www.wallstreetsystems.com

Wall Street Systems – **Empowering** Treasury Trade and Settlement

WALLSTREET SYSTEMS

Wallstreet Suite
WebSuite
Cash Management Connectivity Guide

Version 7.3.16

Information in this document is subject to change without notice and does not represent a commitment on the part of Wall Street Systems. The software and documentation, which includes information contained in any databases, described in this document is furnished under a license agreement or nondisclosure agreement and may only be used or copied in accordance with the terms of the agreement. It is against the law to copy the software or documentation except as specially allowed in the license or nondisclosure agreement. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of Wall Street Systems.

Although Wall Street Systems has tested the software and reviewed the documentation, **Wall Street Systems makes herein no warranty or representation, either expressed or implied, with respect to software or documentation, its quality, performance, marketability, or fitness for a particular purpose. As a result, this software is provided "as is", and in no event will Wall Street Systems be liable for direct, indirect, special, incidental, or consequential damages from any defect in the software or by virtue of providing this documentation,** even if advised of the possibility of such damages. The documentation may contain technical inaccuracies and omissions.

The mention of an activity or instrument in this publication does not imply that all matters relating to that activity or instrument are supported by Wallstreet Suite, nor does it imply that processing of or by that activity or instrument is carried out in any particular way, even if such processing is customary in some or all parts of the industry.

The windows and screen images shown herein were obtained from prototypes during software development. The actual windows and screen images in the software may differ.

First Edition (August 2011)

This edition applies to Wallstreet Suite version 7.3.16 and to all later releases and versions until indicated in new editions or Wall Street Systems communications. Make sure you are using the latest edition for the release level of the Wall Street Systems product.

Contents

Preface **17**

Introduction **17**

How to use this guide **17**

Recommended reading 17

Assumptions 18

Associated documents **18**

1 Overview **19**

1.1 Understanding interfaces **19**

1.2 Defining interface components **20**

1.2.1 Defining formats 20

1.2.2 Connection points 21

1.2.3 Transport mechanisms 22

1.2.4 Security mechanisms 23

1.3 Implementing interfaces **23**

1.3.1 Determining your organization's interface needs 24

1.3.2 Using standard interface components 24

1.3.3 Using CMM tools to create custom interface components 24

1.3.3.1 Web services interface 25

1.3.3.2 Command line interface 25

1.3.3.3 XML-template-based formats 25

1.3.4 Testing interfaces 26

1.3.5 Going live 26

1.4 Setting configuration parameters **26**

1.4.1 Prerequisites 27

1.4.2 Setting configuration parameters using the Configuration Parameters function 27

1.4.3 Setting configuration parameters using the Interfaces Configuration Maintenance function 27

1.4.4 Setting interface configuration parameters 27

1.4.4.1 Attached to SWIFT Network 27

1.4.4.2 BAI Validation on Control Amount 28

1.4.4.3 Citibank Enterprise ID 28

1.4.4.4 Communication Maximum Buffer Size 28

1.4.4.5 Default File Export Directory 29

1.4.4.6 Default File Import Directory 29

1.4.4.7 Enable Shared Service Center 29

1.4.4.8 GnuPG Home Directory 29

1.4.4.9 Import of value date balances enabled 30

1.4.4.10 Maintain SWIFT Template At Payment and Deal Entry 30
1.4.4.11 Maximum String Buffer Process Size 30
1.4.4.12 SafeX File Location 31
1.4.4.13 Static SWIFT Template Data 31
1.4.4.14 SWIFT Duplicate Transaction Handling 31
1.4.4.15 SWIFT MT320 Version 32
1.4.4.16 SWIFT Payment Export Format 32
1.4.4.17 Third-Party Software Location 32
1.5 Opening configuration files 32
1.5.1 Configuration overrides 33
1.5.2 Customization 33
1.5.3 Upgrade 33
1.5.4 Prerequisites 34
1.5.5 Opening configuration files with the Review CMM Configuration Documents function .. 34
1.5.6 Opening configuration files without the Review CMM Configuration Documents function . 35
1.5.7 Returning a configuration file to its default settings 35

2 Managing formats 37
2.1 Configuring the fileimportexportformats.xml file 37
2.1.1 Prerequisites 37
2.1.2 Removing formats from the fileimportexportformats.xml file 37
2.2 Configuring the importfilepattern.xml file 38
2.2.1 Prerequisites 38
2.2.2 Removing format patterns from the importfilepattern.xml file 38
2.2.3 Verifying the business types of bank message import formats 38

3 Managing interchanges (and related data) 41
3.1 Managing transport mechanisms 41
3.1.1 Managing communication protocols 41
3.1.1.1 Prerequisites 41
3.1.1.2 Installing third-party software 41
3.1.1.3 Enabling communication protocols 42
3.1.2 Managing communication protocol parameters 42
3.1.2.1 Prerequisites 42
3.1.2.2 Creating FIN Message Manager protocol parameters 42
3.1.2.3 Creating FTP communication protocol parameters 43
3.1.2.4 Creating MQSeries communication protocol parameters 43
3.1.2.5 Creating SWIFTNet Adaptor communication protocol parameters 44
3.1.2.6 Creating command line communication protocol parameters 45
3.1.2.7 Creating request response communication protocol parameters 45
3.1.2.8 Editing communication protocol parameters 45
3.1.2.9 Deleting communication protocol parameters 46
3.1.3 Completing other tasks 46
3.1.3.1 Prerequisites 46
3.1.3.2 Manually executing communication protocols 46

3.1.3.3 Running processes remotely through communication protocols 46
3.1.4 Entering data in FTP communication protocol parameter file matching fields 47
3.1.4.1 Entering data in the File Match RegExp field 48
3.1.4.2 Entering data in the File Match String field 50
3.1.4.3 Entering data in the File Match Index field 50
3.2 Managing security mechanisms 51
3.2.1 Managing signers 51
3.2.1.1 Prerequisites 51
3.2.1.2 Installing third-party software 51
3.2.1.3 Enabling signers 52
3.2.2 Managing signer parameters 52
3.2.2.1 Prerequisites 52
3.2.2.2 Creating Entrust encryption signer parameters 53
3.2.2.3 Creating Entrust decryption signer parameters 53
3.2.2.4 Creating GnuPG encryption signer parameters 53
3.2.2.5 Creating GnuPG decryption signer parameters 53
3.2.2.6 Creating PGP encryption signer parameters 53
3.2.2.7 Creating PGP decryption signer parameters 54
3.2.2.8 Creating safeX encryption signer parameters 54
3.2.2.9 Creating safeX decryption signer parameters 54
3.2.2.10 Creating command line encryption signer parameters 54
3.2.2.11 Creating command line decryption signer parameters 54
3.2.2.12 Editing signer parameters 55
3.2.2.13 Deleting signer parameters 55
3.3 Managing format processors 55
3.3.1 Prerequisites 55
3.3.2 Creating format processor parameters 56
3.3.3 Editing format processor parameters 56
3.3.4 Deleting format processor parameters 56
3.4 Managing interchanges 56
3.4.1 Prerequisites 57
3.4.2 Creating interchanges 57
3.4.3 Editing interchanges 57
3.4.4 Deleting interchanges 58
3.4.5 Enabling and disabling interchanges 58
3.4.6 Linking interchanges to character sets 58
3.4.7 Optimizing interchanges 59
4 Managing other interface data 61
4.1 Managing transaction subtype mappings 61
4.1.1 Managing internal file codes 61
4.1.1.1 Prerequisites 61
4.1.1.2 Creating internal file codes 62
4.1.1.3 Editing internal file codes 62
4.1.1.4 Deleting internal file codes 62
4.1.2 Managing counterparty file codes 62
4.1.2.1 Prerequisites 63

4.1.2.2 Creating counterparty file codes ... 63
4.1.2.3 Editing counterparty file codes ... 63
4.1.2.4 Deleting counterparty file codes ... 64
4.1.2.5 Managing counterparty file codes for SWIFT ACK and NACK bank messages ... 64
4.1.2.6 Managing mappings ... 65
4.2 Managing branch qualifiers ... 65
4.2.1 Prerequisites ... 65
4.2.2 Creating branch qualifiers ... 65
4.2.3 Editing branch qualifiers ... 66
4.2.4 Deleting branch qualifiers ... 66

5 Using the web services interface ... 67

5.1 SOAP schemas ... 67
5.1.1 Request/response header schema ... 67
5.1.2 Request body schema ... 69
5.1.3 Response body schema ... 69
5.2 Importing messages using the web services interface ... 70
5.2.1 Deploying the web service and creating the adaptor ... 72
5.2.2 Securing the web service ... 72
5.2.2.1 Generating and configuring SSL certificates ... 72
5.2.2.2 Editing the extwsaccessconfig.xml file ... 73
5.2.3 Managing the interchange (and related data) ... 76
5.2.3.1 Managing the interchange (and related data) ... 76
5.2.4 Testing the import process ... 76
5.2.4.1 Testing an accounts payable file import process that uses the web services interface 77
5.2.4.2 Testing a direct debit file import process that uses the web services interface ... 78
5.3 Exporting messages using the web services interface ... 79
5.3.1 Deploying the web service and creating the adaptor ... 80
5.3.2 Securing the web service ... 81
5.3.2.1 Editing the sslconnectionproperties.xml file ... 81
5.3.3 Managing the interchange (and related data) ... 82
5.3.4 Managing the interchange (and related data) ... 82
5.3.5 Testing the export process ... 82
5.3.5.1 Testing a payment file export process that uses the web services interface ... 83
5.3.5.2 Testing a direct debit file export process that uses the web services interface ... 83
5.4 Example SOAP messages ... 83
5.4.1 Initiation request message ... 83
5.4.2 Positive initiation response message ... 83
5.4.3 Negative initiation response message ... 84
5.4.4 Transaction request message ... 85
5.4.5 Positive transaction response message ... 85
5.4.6 Negative transaction response message ... 86
5.5 Example SOAP message construction and consumption ... 86
5.5.1 Binding XML schemas to Java code ... 87
5.5.2 Constructing and consuming SOAP messages ... 89

5.5.2.1 Constructing and consuming SOAP messages for the import process 90
5.5.2.2 Constructing and consuming SOAP messages for the export process 97

6 Using the command line interface 99

6.1 Connecting interface components 99
6.1.1 Connecting transport mechanisms 99
6.1.2 Connecting security mechanisms 100
6.1.3 Connecting format processors 101
6.1.4 Configuring interchanges that use the command line parameters 101

6.2 Example implementation of the command line interface 102
6.2.1 Completing prerequisite steps 103
6.2.2 Configuring the transport mechanism 104
6.2.2.1 Setting up and connecting the transport mechanism 104
6.2.2.2 Testing the transport mechanism 106
6.2.3 Configuring the security mechanisms 106
6.2.3.1 Generating GnuPG keys 106
6.2.3.2 Testing GnuPG keys 107
6.2.3.3 Setting up and connecting the security mechanism 107
6.2.3.4 Testing the security mechanisms 112
6.2.4 Configuring the format processors 113
6.2.4.1 Setting up and connecting format processors 113
6.2.5 Importing bank transaction files using the command line interface 114
6.2.5.1 Creating the bank transaction import interface 114
6.2.5.2 Testing the bank transaction import interface with unencrypted files 115
6.2.5.3 Testing the bank transaction import interface with encrypted files 115
6.2.6 Importing accounts payable files using the command line interface 116
6.2.6.1 Creating the accounts payable import interface 116
6.2.6.2 Testing the accounts payable import interface 117
6.2.7 Exporting payment files using the command line interface 117
6.2.7.1 Creating the payment export interface 117
6.2.7.2 Testing the payment export interface 118

7 Using XML-template-based formats 119

7.1 Creating custom formats 120
7.1.1 Creating custom import formats 121
7.1.1.1 Adding formats to the fileimportexportformats.xml file 121
7.1.1.2 Adding format patterns to the importfilepattern.xml file 123
7.1.1.3 Creating transaction subtype mappings for formats 124
7.1.1.4 Adding formats to the appropriate specification map files 124
7.1.1.5 Creating format parser and mapping files 126
7.1.2 Creating custom export formats 127
7.1.2.1 Adding formats to the fileimportexportformats.xml file 127
7.1.2.2 Creating breakup definitions 127
7.1.2.3 Creating payment aggregation definitions 127
7.1.2.4 Adding formats to the specificationmap.xml file 128
7.1.2.5 Creating format validation and generation files 128

7.2 Testing custom formats 129
7.2.1 Testing custom formats 129
7.2.2 Troubleshooting 130
7.3 Going live 132

Appendix A: Standard formats 133

A.1 Summary of standard banking formats 133
A.2 Forecast import formats 134
A.2.1 Wallstreet XML forecast 134
A.2.1.1 transactions element 135
A.2.1.2 transaction elements 135
A.2.1.3 attribute elements 136
A.2.2 Wallstreet flat file forecast 136
A.2.2.1 Wallstreet flat file forecast header 136
A.2.2.2 Wallstreet flat file forecast body 138
A.3 Transaction import formats 143
A.3.1 Wallstreet XML transaction 143
A.3.1.1 transactions element 144
A.3.1.2 transaction elements 144
A.3.1.3 attribute elements 145
A.3.1.4 component_transaction elements 145
A.3.2 Wallstreet flat file accounts payable 146
A.3.2.1 Wallstreet flat file accounts payable header 146
A.3.2.2 Wallstreet flat file accounts payable body 147
A.3.3 Wallstreet flat file accounts receivable 159
A.3.3.1 Wallstreet flat file accounts receivable 160
A.3.3.2 Wallstreet flat file accounts receivable body 161
A.3.4 Wallstreet flat file direct debit 164
A.3.4.1 Wallstreet flat file direct debit header 165
A.3.4.2 Wallstreet flat file direct debit body 166
A.3.5 EDIFACT PAYEXT 174
A.3.5.1 EDIFACT PAYEXT start of interchange 175
A.3.5.2 EDIFACT PAYEXT start of message 177
A.3.5.3 EDIFACT PAYEXT segment group 1 180
A.3.5.4 EDIFACT PAYEXT segment group 2 181
A.3.5.5 EDIFACT PAYEXT segment group 3 182
A.3.5.6 EDIFACT PAYEXT segment group 4 185
A.3.5.7 EDIFACT PAYEXT segment group 5 190
A.3.5.8 EDIFACT PAYEXT segment group 7 191
A.3.5.9 EDIFACT PAYEXT segment group 8 193
A.3.5.10 EDIFACT PAYEXT segment group 15 195
A.3.5.11 EDIFACT PAYEXT end of message 196
A.3.5.12 EDIFACT PAYEXT end of interchange 197
A.4 Bank message import formats 198
A.4.1 Wallstreet XML bank message 198
A.4.1.1 transactions element 198

A.4.1.2 transaction elements 199
A.4.1.3 attribute elements 199
A.4.2 Wallstreet XML transaction acknowledgement 199
A.4.2.1 transactions element 200
A.4.2.2 transaction elements 200
A.4.3 ISO 20022 XML payment status 201
A.4.3.1 ISO 20022 XML payment status format 201
A.4.3.2 ISO 20022 XML payment status parsing logic 204
A.4.3.3 ISO 20022 XML payment status valid codes 206
A.5 Bank transaction and balance import formats 211
A.5.1 Wallstreet XML bank statement 211
A.5.1.1 bankaccountstatementholder element 212
A.5.1.2 account_statement elements 213
A.5.1.3 account elements 213
A.5.1.4 balance elements 213
A.5.1.5 transaction elements 214
A.5.2 Wallstreet XML bank transaction and balance 214
A.5.2.1 transactions element 215
A.5.2.2 transaction elements 215
A.5.2.3 attribute elements 216
A.5.3 SWIFT MT940 216
A.5.3.1 SWIFT MT940 layout 217
A.5.3.2 SWIFT MT940 header blocks 217
A.5.3.3 SWIFT MT940 main message block 221
A.5.3.4 SWIFT MT940 validation 221
A.5.4 SWIFT MT940 BCS 224
A.5.4.1 SWIFT MT940 BCS layout 225
A.5.4.2 SWIFT MT940 BCS tags 225
A.5.4.3 SWIFT MT940 BCS validation 227
A.6 Deal import formats 229
A.6.1 Wallstreet flat file deal 229
A.6.1.1 Wallstreet flat file deal header 230
A.6.1.2 Wallstreet flat file deal body 231
A.6.1.3 Dates in Wallstreet flat file deal 240
A.7 Transaction export formats 241
A.7.1 SWIFT MT101 241
A.7.1.1 SWIFT MT101 payment scenarios 242
A.7.1.2 SWIFT MT101 layout 243
A.7.1.3 SWIFT MT101 header blocks 244
A.7.1.4 SWIFT MT101 main message block 245
A.7.2 SWIFT MT101 validation 249
A.7.3 SWIFT MT104 250
A.7.3.1 SWIFT MT104 header block 250
A.7.3.2 SWIFT MT104 main message block 251
A.7.4 SWIFT MT210 255
A.7.4.1 SWIFT MT210 header block 255
A.7.4.2 SWIFT MT210 main message block 256

A.7.5 ISO 20022 XML credit transfer ... 257
A.7.5.1 ISO 20022 XML credit transfer format ... 258
A.7.5.2 ISO 20022 XML credit transfer rules ... 265
A.7.6 ISO 20022 XML direct debit transfer ... 267
A.7.6.1 ISO 20022 XML direct debit transfer format ... 268
A.8 Transaction message export formats ... 272
A.8.1 Wallstreet XML payment message ... 272
A.8.1.1 transactions element ... 275
A.8.1.2 transaction elements ... 275
A.8.1.3 debit elements ... 275
A.8.1.4 debit_basics elements ... 276
A.8.1.5 debit_bank_account elements ... 276
A.8.1.6 debit_bank elements ... 276
A.8.1.7 debit_party elements ... 277
A.8.1.8 credit elements ... 277
A.8.1.9 credit_basics elements ... 277
A.8.1.10 credit_bank_account elements ... 278
A.8.1.11 credit_bank elements ... 278
A.8.1.12 intermediary_bank elements ... 279
A.8.1.13 credit_party elements ... 279
A.8.1.14 originating_party elements ... 279
A.8.1.15 additional_info elements ... 280
A.8.1.16 remittance_details elements ... 280
A.8.2 Wallstreet XML receipt message ... 281
A.8.2.1 transactions element ... 283
A.8.2.2 transaction elements ... 284
A.8.2.3 credit elements ... 284
A.8.2.4 credit_basics elements ... 284
A.8.2.5 credit_bank_account elements ... 284
A.8.2.6 credit_bank elements ... 285
A.8.2.7 credit_party elements ... 285
A.8.2.8 debit elements ... 285
A.8.2.9 debit_basics elements ... 286
A.8.2.10 debit_bank_account elements ... 286
A.8.2.11 debit_bank elements ... 287
A.8.2.12 intermediary_bank elements ... 287
A.8.2.13 debit_party elements ... 287
A.8.2.14 originating_party elements ... 288
A.8.2.15 additional_info elements ... 288
A.9 General ledger posting export formats ... 289
A.9.1 Wallstreet XML general ledger posting ... 289
A.9.1.1 transactions element ... 290
A.9.1.2 transaction elements ... 290
A.10 Free formats ... 291
A.10.1 SWIFT MT199 and MT999 ... 291
A.10.1.1 SWIFT MT199 and MT999 header block ... 292
A.10.1.2 SWIFT MT199 and MT999 main message block ... 292

A.11 Wallstreet XML schemas 292
A.11.1 Transaction schema 293
A.11.2 Transaction acknowledgement schema 294
A.11.3 Bank statement schema 294
A.11.4 Payment message schema 296
A.11.5 Receipt message schema 299
A.11.6 General ledger posting schema 301
A.12 Supplemental information for EDIFACT formats 302
A.12.1 EDIFACT Level A character set 302
A.12.1.1 Alphabetic characters (a) 303
A.12.1.2 Numeric characters (n) 303
A.12.1.3 Alphabetic characters (an) 303
A.12.1.4 Element lengths 303
A.13 Supplemental information for SWIFT formats 303
A.13.1 SWIFT field lengths 303
A.13.2 SWIFT field types 304
A.13.3 SWIFT character sets 304
A.13.3.1 SWIFT X character set 304
A.13.3.2 SWIFT Y character set 304
A.13.3.3 SWIFT Z character set 304
A.13.4 SWIFT transaction type identification code mappings 305
A.13.5 SWIFT business transaction code (GVC) mappings 306
A.14 Syntactic validation of incoming and outgoing MX messages 308

Appendix B: Attributes 311

B.1 Import attributes 312
B.1.1 Forecast import attributes 312
B.1.2 Accounts payable import attributes 316
B.1.3 Accounts receivable import attributes 325
B.1.4 Bank transaction import attributes 328
B.1.5 Bank balance import attributes 330
B.1.6 Bank message import attributes 331
B.1.7 Interchange import attributes 332
B.1.8 File import attributes 333
B.2 Export attributes 334
B.2.1 Payment export attributes 334
B.2.2 Receipt export attributes 351
B.2.3 Credit advice export attributes 358
B.2.4 Remittance detail export attributes 359
B.2.5 Bank statement export attributes 360
B.2.6 Company general ledger export attributes 365
B.2.7 Interchange export attributes 366

Appendix C: Handlers 369

C.1 add_to_buffer 371
C.1.1 Parameters 371

C.1.2 Examples ... 371
C.2 attribute_defined_element ... 371
C.2.1 Parameters ... 371
C.2.2 Examples ... 372
C.3 build_mail_message ... 372
C.3.1 Parameters ... 372
C.3.2 Examples ... 372
C.4 build_txn_groups ... 373
C.4.1 Parameters ... 373
C.4.2 Examples ... 373
C.5 charset ... 374
C.5.1 Parameters ... 374
C.5.2 Examples ... 374
C.6 component_transactions ... 375
C.6.1 Parameters ... 375
C.6.2 Examples ... 375
C.7 condition_test_element ... 375
C.7.1 Parameters ... 375
C.7.2 Component handlers ... 376
C.7.3 Examples ... 380
C.8 configure_attribute_container ... 383
C.8.1 Parameters ... 383
C.8.2 Examples ... 383
C.9 context_defined_element ... 383
C.9.1 Parameters ... 384
C.9.2 Examples ... 384
C.10 context_value_iterator ... 385
C.10.1 Parameters ... 385
C.10.2 Examples ... 385
C.11 create_node ... 385
C.11.1 Parameters ... 385
C.11.2 Examples ... 385
C.12 create_object ... 385
C.12.1 Parameters ... 386
C.12.2 Examples ... 386
C.13 create_tree ... 386
C.13.1 Parameters ... 386
C.13.2 Examples ... 386
C.14 echo ... 386
C.14.1 Parameters ... 387
C.14.2 Examples ... 387
C.15 find_parent_node ... 387
C.15.1 Parameters ... 388
C.15.2 Examples ... 388

C.16 find_root_node 388
C.16.1 Parameters 388
C.16.2 Examples 388
C.17 get_system_date 388
C.17.1 Parameters 388
C.17.2 Examples 389
C.18 if 389
C.18.1 Parameters 389
C.18.2 Examples 389
C.19 include 390
C.19.1 Parameters 390
C.19.2 Examples 391
C.20 increment_counter 391
C.20.1 Parameters 391
C.20.2 Examples 391
C.21 insert_chars 391
C.21.1 Parameters 391
C.21.2 Examples 392
C.22 iterate_db_result_set 392
C.22.1 Parameters 392
C.22.2 Examples 392
C.23 jython_script 393
C.23.1 Parameters 393
C.23.2 Examples 393
C.24 log_invalid_transaction 393
C.24.1 Parameters 393
C.24.2 Examples 393
C.25 map_value 394
C.25.1 Parameters 394
C.25.2 Examples 394
C.26 mathFunction 395
C.26.1 Parameters 395
C.26.2 Examples 395
C.27 pad_handler 395
C.27.1 Parameters 396
C.27.2 Examples 396
C.28 parse_delimiter 396
C.28.1 Parameters 396
C.28.2 Examples 397
C.29 parse_fixed_width 397
C.29.1 Parameters 398
C.29.2 Examples 398
C.30 parse_range_width 398
C.30.1 Parameters 398

C.30.2 Examples 399
C.31 parse_xml_tree 399
C.31.1 Parameters 400
C.31.2 Examples 400
C.32 remittance_details_list 400
C.32.1 Parameters 400
C.32.2 Examples 400
C.33 remove_chars 400
C.33.1 Parameters 401
C.33.2 Examples 401
C.34 remove_context_variable 401
C.34.1 Parameters 401
C.34.2 Examples 401
C.35 replace_string 402
C.35.1 Parameters 402
C.35.2 Examples 402
C.36 reset_counter 402
C.36.1 Parameters 402
C.36.2 Examples 402
C.37 save_value_to_context 403
C.37.1 Parameters 403
C.37.2 Examples 403
C.38 send_mail_message 403
C.38.1 Parameters 404
C.38.2 Examples 404
C.39 set_context_variable 404
C.39.1 Parameters 404
C.39.2 Examples 404
C.40 set_node_attribute 405
C.40.1 Parameters 405
C.40.2 Examples 405
C.41 set_value 406
C.41.1 Parameters 406
C.41.2 Examples 406
C.42 sql_query_call 407
C.42.1 Parameters 407
C.42.2 Examples 408
C.43 static_data_element 409
C.43.1 Parameters 409
C.43.2 Examples 409
C.44 store_object 409
C.44.1 Parameters 409
C.44.2 Examples 409
C.45 substring 409

C.45.1 Parameters 410
C.45.2 Examples 410
C.46 traverse_tree 410
C.46.1 Parameters 410
C.46.2 Examples 411
C.47 truncate_handler 412
C.47.1 Parameters 412
C.47.2 Examples 412
C.48 txn_aggregation 413
C.48.1 Parameters 413
C.48.2 Examples 414
C.49 txn_groups_list 414
C.49.1 Parameters 415
C.49.2 Examples 415
C.50 txn_list 415
C.50.1 Parameters 415
C.50.2 Examples 415
C.51 use_attribute_container 416
C.51.1 Parameters 416
C.51.2 Examples 416
C.52 use_buffer_input 416
C.52.1 Parameters 416
C.52.2 Examples 416
C.53 use_node_attribute 417
C.53.1 Parameters 417
C.53.2 Examples 417
C.54 use_node_value 417
C.54.1 Parameters 417
C.54.2 Examples 417
C.55 use_output_device 418
C.55.1 Parameters 418
C.55.2 Examples 418

Appendix D: SSL certificate generation and configuration 419

D.1 Assumptions 420
D.2 Generating SSL certificates 420
D.2.1 Generating SSL certificates 420
D.3 Creating a secure website using IIS 423
D.3.1 Assigning an additional IP address 423
D.3.1.1 Assigning an additional IP address 424
D.3.2 Creating a secure website 424
D.3.2.1 Creating a secure website 424
D.3.3 Configuring the secure website 425
D.3.3.1 Configuring the secure website 425

D.3.4 Creating a virtual folder for content 427
D.3.4.1 Creating a virtual folder for content 427
D.3.5 Configuring two-way SSL 428
D.3.5.1 Importing server and CA certificates 428
D.3.5.2 Assigning the server certificate 431
D.3.6 Configuring one-way SSL 432
D.3.6.1 Configuring one-way SSL 432
D.4 Creating a secure website using Tomcat 433
D.4.1 Creating a secure website using Tomcat 434
D.5 Testing HTTPS configuration 434
D.5.1 Importing client certificates to the web browser 434
D.5.2 Configuring CMM with PKI X509 435
D.5.3 Testing CMM with PKI X509 436

Preface

Introduction

This guide enables WebSuite system administrators to set up the Cash Management Module's interfaces and, where appropriate, to use the Web services interface, the command line interface, and XML-template-based formats to create custom interface components.

Note: The term "interfaces" in this guide refers to interfaces that connect WebSuite to banks and other external systems. For information on the user interface, see the *WebSuite User Guide* and the *WebSuite System Administration Guide*.

WebSuite users and system administrators as well as other interest parties should have a general knowledge of the following:

- Basic cash management
- The bank, enterprise resource planning (ERP), and other systems connected to WebSuite
- The technologies used in setting up interfaces, particularly XML
- The Web browser you are using to access CMM (for example, Microsoft Internet Explorer).

Experience with Wallstreet Suite's other modules—particularly Transaction and Risk Module (TRM) and Accounting Module (ACM)—is beneficial but not necessary.

How to use this guide

To use this guide, follow the recommended reading and be aware of the assumptions defined in this section.

Recommended reading

If you are new to CMM, read *Chapter 1 Overview* on page 19.

If you are responsible for configuring interfaces, read the following chapters in this guide:

- *Chapter 2 Managing formats* on page 37
- *Chapter 3 Managing interchanges (and related data)* on page 41
- *4.1 Managing transaction subtype mappings* on page 61.

If you are responsible for creating or maintaining customized interface components, the following chapters in this guide:

- *Chapter 5 Using the web services interface* on page 67
- *Chapter 6 Using the command line interface* on page 99
- *Chapter 7 Using XML-template-based formats* on page 119.

Note: Connecting to the SWIFT system is described in the *WSS SWIFT Connectivity Guide*.

Assumptions

This guide assumes the following:

- You are using the default menu installed with CMM.
- You have enabled JavaScript by setting the JavaScript Enabled configuration parameter to `True`.
- You are using Microsoft Internet Explorer.
- You have access to the CMM application server.

Associated documents

Associated documents can be accessed from the Help menu of TRM and ACM. All Wallstreet Suite user documentation can be downloaded from the Wallstreet Customer Support site http://www.trema.com/support.

Chapter 1 Overview

An important step in implementing the Cash Management Module (CMM) part of WebSuite for an organization is connecting the module to other financial systems through interfaces.

The creation and maintenance of interfaces is an involved process. It requires the efforts of several parties, including the following:

- Your organization
- Wallstreet or one of its implementation partners
- The external systems to which your organization wants to connect CMM (particularly bank systems).

1.1 Understanding interfaces

Interfaces are a key feature of CMM. They facilitate straight-through processing of information between CMM and your organization's bank, enterprise resource planning, treasury management, and other external systems.

Using interfaces, you can import information to and export information from CMM:

Imports	Exports
• Forecasts • Payments (AP) • Receipts (AR) • Direct debits • Bank transactions • Bank messages • Free format messages • Interest rates • Foreign exchange rates • Static data.	• Payments • Receipts • Credit advices • Direct debits • Letters of credit • Positive payments • Remittance advices • Confirmation messages • Requisition messages • Free format messages • Bank statements • Bank balances • General ledger postings.

Each interface your organization uses in CMM is defined by an interchange. An interchange specifies its interface's type and format as well as how information is transferred between CMM and other financial systems.

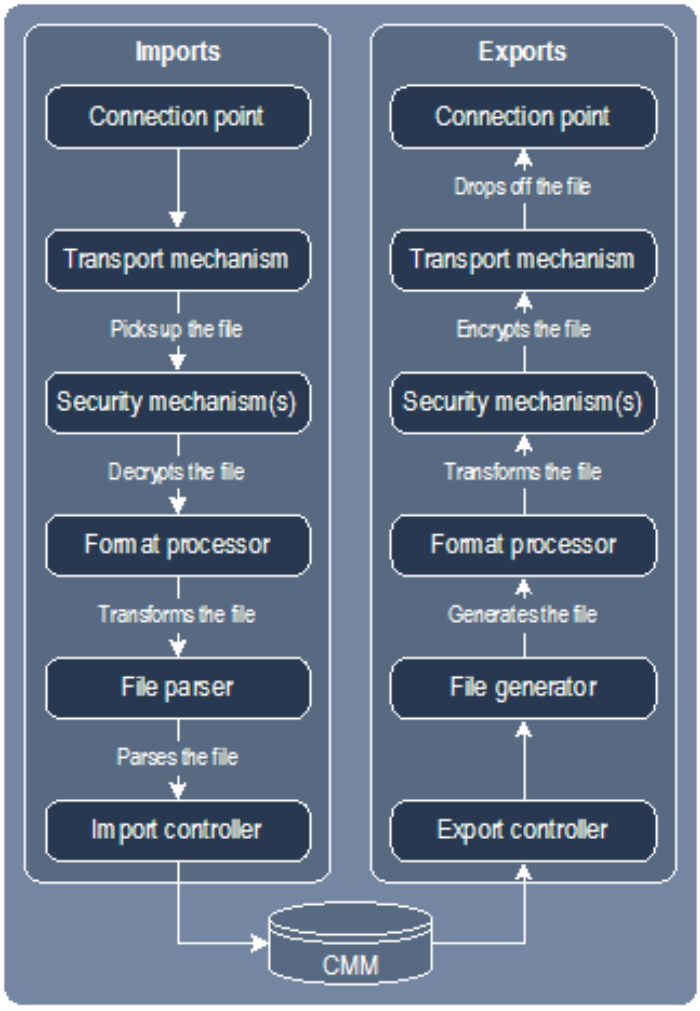
Note: For more information on interchanges, see Chapter 3 Managing interchanges (and related data) on page 41.

1.2 Defining interface components

Each interface in CMM consists of the following components:

- One format
- One connection point
- One transport mechanism
- One or more security mechanisms.

With these components, you can import and export information as presented in the following diagram:



1.2.1 Defining formats

An interface's format specifies the layout of its files. Each format can be one of the following:

- An industry standard, such as SWIFT MT103 for payments
- A variation of an industry standard used by a specific party or in a specific country or region
- A CMM standard
- Proprietary.

A set of default formats are provided with CMM. These formats are specified in a file named `fileimportexportformats.xml`, which is included in the CMM installation. You can add formats to

this file. In addition, you should remove formats from this file that your organization is not planning to use to ensure optimal performance of CMM.

Note: For information on the `fileimportexportformats.xml` file, see 2.1 Configuring the fileimportexportformats.xml file on page 37.

If a format is specified in the `fileimportexportformats.xml` file, you can import or export files in that format. However, if a format is not specified in the `fileimportexportformats.xml` file, you need to transform files in that format to a format in the file before importing or exporting them. You do this using format processor parameters.

Note: For information on format processor parameters, see 3.3 Managing format processors on page 55.

1.2.2 Connection points

A connection point is a location from which CMM picks up files (for imports) or to which CMM drops off files (for exports).

Usually, you define an interface's connection point in its interchange. This is the approach Wallstreet recommends. However, if you do not define a connection point in the interface's interchange, CMM refers to and uses the default import or export folder instead.

The following table presents the default folders for imports:

Interface type	Location
Forecast files	`[DefaultImportFolder]\forecasts\`
Accounts payable files	`[DefaultImportFolder]\[CounterpartyID]\AP\`
Accounts receivable files	`[DefaultImportFolder]\[CounterpartyID]\AR\`
Direct debit files	`[DefaultImportFolder]\[CounterpartyID]\DD\`
Bank transactions	`[DefaultImportFolder]\BankImports\[CounterpartyID]\`
Bank messages	`[DefaultImportFolder]\BankImports\[CounterpartyID]\`
Interest rates	`[DefaultImportFolder]\InterestRates\`
Foreign exchange rates	`[DefaultImportFolder]\FxRates\`
Static data (entities)	`[DefaultImportFolder]\Entity\`
Static data (counterparties)	`[DefaultImportFolder]\Counterparty\`
Static data (bank accounts)	`[DefaultImportFolder]\BankAcct\`

Note: In the above table, `[DefaultImportFolder]` is the default import folder defined by the Default File Import Directory configuration parameter (for example, `C:\cmm\Imports\`) and `[CounterpartyID]` is the ID of the interface's counterparty (for example, `SmithCo`).

The following table presents the default folders for exports:

Interface type	Location
Payment files	`[DefaultExportFolder]\BankPayment\[CounterpartyID]\`
Receipt files	`[DefaultExportFolder]\BankReceipt\[CounterpartyID]\`
Credit advice files	`[DefaultExportFolder]\CreditAdvice\[CounterpartyID]\`

Interface type	Location
Direct debit files	[DefaultExportFolder]\DirectDebit\[CounterpartyID]\
Letter of credit files	[DefaultExportFolder]\Default\[CounterpartyID]\
Positive pay files	[DefaultExportFolder]\PositivePay\[CounterpartyID]\
Remittance advice files	[DefaultExportFolder]\Ra\
Confirmation messages (BANSTA)	[DefaultExportFolder]\Bansta\
Confirmation messages (HTML)	[DefaultExportFolder]\Html\
Requisition messages	[DefaultExportFolder]\Requisition\
Bank statements	[DefaultExportFolder]\Default\[CounterpartyID]\
External general ledger files (code-based)	[DefaultExportFolder]\GL\
External general ledger files (template-based)	[DefaultExportFolder]\Default\[CounterpartyID]\

Note: In the above table, [DefaultExportFolder] is the default export folder defined by the Default File Export Directory configuration parameter (for example, C:\cmm\Exports\) and [CounterpartyID] is the ID of the interface's counterparty (for example, SmithCo).

For information on setting the Default File Import Directory and Default File Export Directory configuration parameters, see 1.4.4 Setting interface configuration parameters on page 27.

Locations defined in interchanges and the previously mentioned configuration parameters must be physical folders accessible to the CMM application server.

1.2.3 Transport mechanisms

A transport mechanism allows CMM to pick up files from or drop off files to the connection point without intervention from external applications.

In CMM, transport mechanisms are called "communication protocols". The following are the communication protocols available for use in the module:

Communication protocol	Reason to use the communication protocol
FTP	You are transporting files through FTP.
MQ Series Communication	You are transporting files through MQSeries.
Generic command line	You are transporting files through a third-party tool connected to CMM by the command line interface.
Request Response	You are transporting files through a third-party tool connected to CMM by the web services interface.
SWIFTNet Adaptor	You are transporting files to and from SWIFTNet.

Note: For information on creating and maintaining parameters for communication protocols, see 3.1 Managing transport mechanisms on page 41.

1.2.4 Security mechanisms

Security mechanisms ensure a file cannot be tampered with while being transported to or from CMM. You can assign one or more security mechanisms to each interface.

In CMM, a security mechanism is called a "signer". Signers either encrypt export files or decrypt import files. In addition, you can use signers to create or verify digital signatures. A digital signature identifies the sender and connects the sender to an exact message. When combined with a digital time stamp, a digital signature can be used to verify the time the message was sent. Anyone who has a party's public key can verify the integrity of that party's digital signature. If a message from the party is altered in any way, the signature does not decrypt properly. This indicate that the message was altered in transit or that copying it from a different message forged the signature.

CMM supports the following third-party tools:

- Entrust
- GnuPG
- PGP
- safeX.

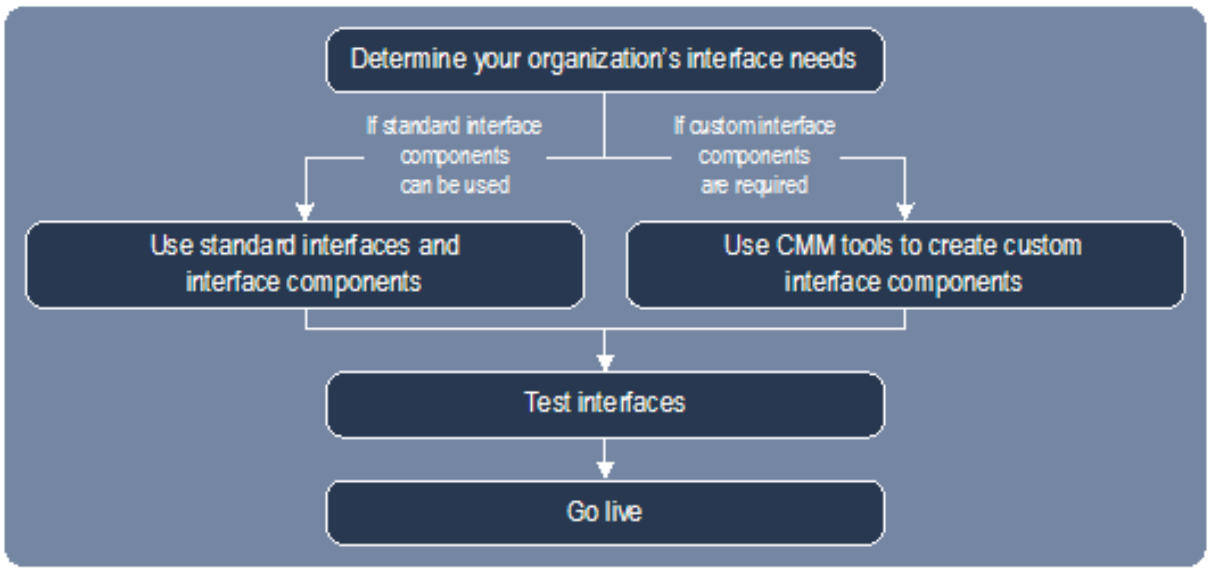
If your organization has installed these tools, it can create and maintain signer parameters in CMM that use them to encrypt, decrypt and sign files.

If you want to use other tools or use these tools in a manner different than how they were implemented in CMM, you can use the command line interface.

Note: For information on creating and maintaining signer parameters, see 3.2 Managing security mechanisms on page 51.

1.3 Implementing interfaces

Implementing interfaces requires the following steps:



1.3.1 Determining your organization's interface needs

The first step in implementing interfaces is to determine your organization's specific interface needs. To do this, ask yourself the following questions:

- Imports
  - What information will you be importing to CMM?
  - What is the source of this information?
  - What is the format of this information?
  - How will the information be transported to CMM?
  - Will security (decryption and digital signatures) be required?
- Exports
  - What information will you be exporting from CMM?
  - What is the destination of this information?
  - What is the format of this information?
  - How will the information be transported from CMM?
  - Will security (encryption and digital signatures) be required?

You may need to work with your organization's banks and other providers as well as Wallstreet or one of its partners to answer these questions.

When you have completed answering these questions, you will have a list of formats, transport mechanisms, and security mechanisms, some of which will be supported by CMM and some of which will not be supported by CMM:

- For the former, you can use standard interface components.

  For more information, see 1.3.2 Using standard interface components on page 24.
- For the latter, you (or a consultant) must use CMM tools to create custom interface components.

  For more information, see 1.3.3 Using CMM tools to create custom interface components on page 24.

1.3.2 Using standard interface components

As noted in 1.2 Defining interface components on page 20, CMM supports a standard set of interface formats, transport mechanisms (or "communication protocols"), and security mechanisms (or "signers").

If these standard interface components meet the needs you identified in 1.3.1 Determining your organization's interface needs on page 24, you can create parameters in CMM that utilize them. You can then create interchanges that refer to those parameters. For more information, see Chapter 3 Managing interchanges (and related data) on page 41.

1.3.3 Using CMM tools to create custom interface components

If one or more of your needs are not satisfied by CMM's standard interface components, you can create custom interface components using the following CMM tools:

Tool	Use to create[o]		
	Formats	Transport mechanisms	Security mechanisms
Web services interface			

Tool	Use to create[o]		
Command line interface	✓	✓	✓
XML-template-based formats	✓		

Each tool uses a different set of web-based technologies and requires varying levels of technical knowledge. If you do not have this knowledge, consider hiring a consultant.

1.3.3.1 Web services interface

The web services interface allows you to connect CMM to an application external to CMM through an adaptor. The application then transforms, secures, or transports files for CMM.

To use an application external to CMM with the web services interface, you must install it in an appropriate location accessible to the CMM application server and create an adaptor to connect it to CMM. Therefore, you require extensive knowledge of the third-party application.

The application can create status messages and pass them on to CMM. These status messages display in CMM's job log. Therefore, exception handling and status reporting for interfaces created through the web services interface are independent of CMM's standard exception handling and status reporting and can be as detailed and user friendly as you require.

For more information on the web services interface, see Chapter 5 Using the web services interface on page 67.

1.3.3.2 Command line interface

The command line interface allows you to use applications external to CMM to transform, secure, and transport files (assuming the applications are executable from the command line).

To use an application external to CMM with the command line interface, you must install it in an appropriate location accessible to the CMM application server and create commands. This requires knowledge of the third-party application. For example, to use Perl to transform files from one format to another, you need know how to install Perl and create commands in it.

After you have installed the application and created commands, you can create communication protocol, signer, or format processor parameters in CMM that reference the application through the command line interface. You can then create interchanges that use these parameters.

For more information on the command line interface, see Chapter 6 Using the command line interface on page 99.

1.3.3.3 XML-template-based formats

Using XML templates, you can specify custom formats for the following types of information:

Imports	Exports
• Forecasts • Payments (AP) • Receipts (AR) • Direct debits • Bank statements • Bank messages.	• Payments • Direct debits • Letters of credit • Preadvices • Credit advices • Bank statements • General ledger postings.

To import files using XML-template-based formats, you must first map attributes from the source external system to attributes in CMM, you can create an XML template based on this mapping.

To export files using XML-template-based formats, you follow a similar process.

For more information on XML-template-based formats, see Chapter 7 Using XML-template-based formats on page 119.

1.3.4 Testing interfaces

Wallstreet recommends you test your interfaces before going live. To do this, set up the interfaces on your organization's CMM test environment. Working with your organization's banks and other external systems, import sample files to the test environment or export sample files from the test environment. If exporting sample files, confirm with the banks or other external systems that they have received and can process the files.

1.3.5 Going live

If your interfaces pass testing, you can set them up in your production environment and go live.

The following tasks require interchanges (using your interfaces) be set up to run successfully:

- Importing static data
- Importing interest rates
- Importing foreign exchange rates
- Importing forecasts
- Importing transactions
- Releasing transactions
- Managing bank messages
- Managing free format bank messages
- Importing bank statements, transactions, and balances
- Exporting bank statements
- Exporting bank balances
- Exporting external general ledger files.

You can schedule these tasks in the Task Scheduler function or manually run them on a one-time basis. For more information, see the *WebSuite User Guide* and the *WebSuite System Administration Guide*.

1.4 Setting configuration parameters

CMM's configuration parameters allow you to configure key components of the module, including default functional settings and file paths. A subset of these configuration parameters is relevant to interfaces and can be set in either the Configuration Parameters function or the Interfaces Configuration Maintenance function.

1.4.1 Prerequisites

The following are prerequisites for opening configuration files:

Category	Tasks
Security	Ensure you have access to the following functions: • FG-0014 Configuration Parameters • FG-0460 View/Edit Interfaces Configuration. For more information, see the *WebSuite System Administration Guide*.

1.4.2 Setting configuration parameters using the Configuration Parameters function

To set a configuration parameter using the Configuration Parameters function:

1. Select **Admin - Utilities - Setup - Configuration Parameters**.
2. In the Configuration Parameters Maintenance [list] page, enter search criteria.
3. Click **Search**.
4. Drill down on the configuration parameter.
5. In the Configuration Parameters Maintenance [editor] page, set the configuration parameter.
6. Click **Save**.

1.4.3 Setting configuration parameters using the Interfaces Configuration Maintenance function

To set a configuration parameter using the Interfaces Configuration Maintenance function:

1. Select **Admin - Static Data - Bank Interfacing - Interfaces Configuration Maintenance.**
2. In the resulting page, set the configuration parameter.
3. Click **Save**.

1.4.4 Setting interface configuration parameters

This section defines the configuration parameters relevant to interfaces. You must set these configuration parameters before configuring interfaces.

1.4.4.1 Attached to SWIFT Network

The Attached to SWIFT Network configuration parameter specifies if specific SWIFT message processing is enabled:

Attribute	Value
Possible values	• `True` Specific SWIFT message processing is enabled because the application server has direct access to the SWIFT network. • `False` Specific SWIFT message processing is not enabled.
Default value	• `False`

Attribute	Value
Editable in	• Configuration Parameters • Interfaces Configuration Maintenance

1.4.4.2 BAI Validation on Control Amount

The BAI Validation on Control Amount configuration parameter specifies if control amount validation during BAI import is enabled:

Attribute	Value
Possible values	• `True` Control amount validation during BAI import is enabled. • `False` Control amount validation during BAI import is disabled.
Default value	• `True`
Editable in	• Configuration Parameters • Interfaces Configuration Maintenance

1.4.4.3 Citibank Enterprise ID

The Citibank Enterprise ID configuration parameter specifies the Citibank message request enterprise ID:

Attribute	Value
Possible values	• [Valid Citibank enterprise ID] The Citibank message request enterprise ID.
Default value	• N/A
Editable in	• Configuration Parameters

1.4.4.4 Communication Maximum Buffer Size

The Communication Maximum Buffer Size configuration parameter specifies the maximum size of the communication processing buffer in bytes:

Attribute	Value
Possible values	• [Valid number] The maximum size of the communication processing buffer in bytes. For example, MQSeries requires string buffers on import and export. If the buffer size is too large, out of memory errors can occur. (During imports, if the buffer size is exceeded, an error displays asking you to increase buffer size to process messages. During exports, if the buffer size is exceeded, CMM splits the messages into smaller files.)
Default value	• `4194304`
Editable in	• Configuration Parameters • Interfaces Configuration Maintenance

1.4.4.5 Default File Export Directory

The Default File Export Directory configuration parameter specifies the default file path CMM references when writing export files:

Attribute	Value
Possible values	• [Valid file path] The default file path CMM references when writing export files.
Default value	• `d:\cmm\Exports`
Editable in	• Configuration Parameters • Interfaces Configuration Maintenance

1.4.4.6 Default File Import Directory

The Default File Import Directory configuration parameter specifies the default file path CMM references when reading import files:

Attribute	Value
Possible values	• [Valid file path] The default file path CMM references when reading import files.
Default value	• `d:\cmm\Imports`
Editable in	• Configuration Parameters • Interfaces Configuration Maintenance

1.4.4.7 Enable Shared Service Center

The Enable Shared Service Center configuration parameter specifies whether the **Shared Service Center** field displays in the Interchanges function for SWIFT format interchanges:

Attribute	Value
Possible values	• `True` The **Shared Service Center** field displays in the Interchanges function for SWIFT format interchanges. • `False` The **Shared Service Center** field does not display in the Interchanges function for SWIFT format interchanges.
Default value	• N/A
Editable in	• Configuration Parameters

1.4.4.8 GnuPG Home Directory

The GnuPG Home Directory configuration parameter specifies the file path in which GnuPG is installed:

Attribute	Value
Possible values	• [Valid file path] The file path in which GnuPG is installed.
Default value	• N/A

Attribute	Value
Editable in	• Configuration Parameters • Interfaces Configuration Maintenance

1.4.4.9 Import of value date balances enabled

The Import of value date balances enabled configuration parameter specifies how to treat tags 64 and 65 in a SWIFT MT940 message:

Attribute	Value
Possible values	• `True` SWIFT MT940 message import: field 64 (Closing Available Balance) and field 65 (Forward Available Balance) are imported into WebSuite and are not recalculated. • `False` SWIFT MT940 message import: field 64 (Closing Available Balance) and field 65 (Forward Available Balance) are ignored by WebSuite which calculates these figures.
Default value	• `True`
Editable in	• Configuration Parameters • Interfaces Configuration Maintenance

1.4.4.10 Maintain SWIFT Template At Payment and Deal Entry

The Maintain SWIFT Templates At Payment and Deal Entry configuration parameter specifies if users can add or edit SWIFT templates during transaction or deal entry:

Attribute	Value
Possible values	• `True` Users can add or edit SWIFT templates during transaction or deal entry. • `False` Users cannot add or edit SWIFT templates during transaction or deal entry.
Default value	• `True`
Editable in	• Configuration Parameters • Interfaces Configuration Maintenance

1.4.4.11 Maximum String Buffer Process Size

The Maximum String Buffer Process Size configuration parameter specifies the maximum size of the string buffer for import/export processing:

Attribute	Value
Possible values	• [Valid whole number greater than 0] The maximum size of the string buffer for import/export processing (defined in kilobytes).
Default value	• `150`
Editable in	• Configuration Parameters • Interfaces Configuration Maintenance

1.4.4.12 SafeX File Location

The SafeX 2.x File Location configuration parameter specifies the file path of the safeX executable:

Attribute	Value
Possible values	• [Valid file path] The file path of the safeX executable.
Default value	• N/A
Editable in	• Configuration Parameters • Interfaces Configuration Maintenance

1.4.4.13 Static SWIFT Template Data

The Static SWIFT Template Data configuration parameter specifies if users can edit the default fields of the SWIFT template:

Attribute	Value
Possible values	• `True` Users can edit the default fields of the SWIFT template. • `False` Users cannot edit the default fields of the SWIFT template.
Default value	• `False`
Editable in	• Configuration Parameters • Interfaces Configuration Maintenance

1.4.4.14 SWIFT Duplicate Transaction Handling

The SWIFT Duplicate Transaction Handling configuration parameter specifies if CMM verifies that the transactions in SWIFT files have not already been imported when import SWIFT files:

Attribute	Value
Possible values	• `True` When importing SWIFT files, CMM verifies that the transactions in the files have not already been imported. If they have, CMM invokes duplicate transaction handling. • `False` When importing SWIFT files, CMM does not verify that the transactions in the files have not already been imported.
Default value	• `False`
Editable in	• Configuration Parameters • Interfaces Configuration Maintenance

1.4.4.15 SWIFT MT320 Version

The SWIFT MT320 Version configuration parameter specifies the version of SWIFT messaging (MT320) used by CMM:

Attribute	Value
Possible values	• `Version 1` CMM uses version 1 of SWIFT messaging (MT320). • `Version 2` CMM uses version 2 of SWIFT messaging (MT320).
Default value	• `Version 2`
Editable in	• Configuration Parameters • Interfaces Configuration Maintenance

1.4.4.16 SWIFT Payment Export Format

The SWIFT Payment Export Format configuration parameter specifies whether CMM uses the MT100 or MT103 format for SWIFT payment exports:

Attribute	Value
Possible values	• `MT100` CMM uses the older MT100 format for SWIFT payment exports. • `MT103` CMM uses the newer MT103 format for SWIFT payment exports.
Default value	• `MT103`
Editable in	• Configuration Parameters • Interfaces Configuration Maintenance

1.4.4.17 Third-Party Software Location

The Third-Party Software Location configuration parameter specifies the file path for third-party software accessed for interface security and communication:

Attribute	Value
Possible values	• [Valid file path] The file path for third-party software accessed for interface security and communication.
Default value	• `D:\cmm\3rdPartyProgDir`
Editable in	• Configuration Parameters • Interfaces Configuration Maintenance

1.5 Opening configuration files

Some interface tasks in CMM require you to open and edit configuration files.

Most configuration files utilize XML and are stored in one of these three locations:

- Configuration files that are relevant to individual application servers are maintained in an `InstallationData` folder for each application server. This folder is located here:

  <install home>\envs\<env>\etc\wss-web\cmm\InstallationData

- Configuration files that are relevant to the web interface as a whole are maintained in a central `ConfigurationData` folder. This folder is located here:

  <install home>\envs\<env>\etc\wss-web\cmm\ConfigurationData

- Default configuration files that contain all of the CMM configuration. This folder is located here:

  <install home>\components\wss-web\websuite\DefaultData\default\AurosConfigData\standard

1.5.1 Configuration overrides

Any configuration file in the `ConfigurationData` or `InstallationData` folder overrides its "twin" configuration file in the default CMM configuration directory (assuming it is in the correct subfolder) without actually overwriting it.

The override order is as follows: the `InstallationData` folder takes precedence over `ConfigurationData` which takes precedence over `DefaultData`.

1.5.2 Customization

Any customization must be done in the `ConfigurationData` or `InstallationData` folder.

If by default, the configuration file to customize is not present in these two folders, the original file must be copied from the `DefaultData` folder and put under `ConfigurationData` folder (if this is to be a global configuration change) or under the `InstallationData` folder (if this is an environment-specific change), respecting the same directory hierarchy.

1.5.3 Upgrade

When upgrading Wallstreet Suite, before applying the old configuration files from the `InstallationData` and `ConfigurationData` folders, you must check them against the new default configurations available in the `InstallationData`/`ConfigurationData`/`DefaultData` folders of the new installation, and merge what has been added between the two versions (for example, a new import format may have been added).

Contact the Support Center if you have any doubts regarding the configuration merging.

You can edit most configuration files using the Review CMM Configuration Documents function. As a result, you do not need access to the application server to edit files.

As of this release, the following configuration files are not available in the Review CMM Configuration Documents function:

- Class map files
- Service definition files
- Object mapping files
- Files that can be modified through editors in the user interface.

In most situations, you do not need to edit these files. In a future release, you may be able to view (but not edit) these files from the Review CMM Configuration Documents function.

1.5.4 Prerequisites

The following are prerequisites for opening configuration files:

Category	Tasks
Security	Ensure you have access to the following function: • FG-0400 Review CMM Configuration. For more information, see the *WebSuite System Administration Guide*.

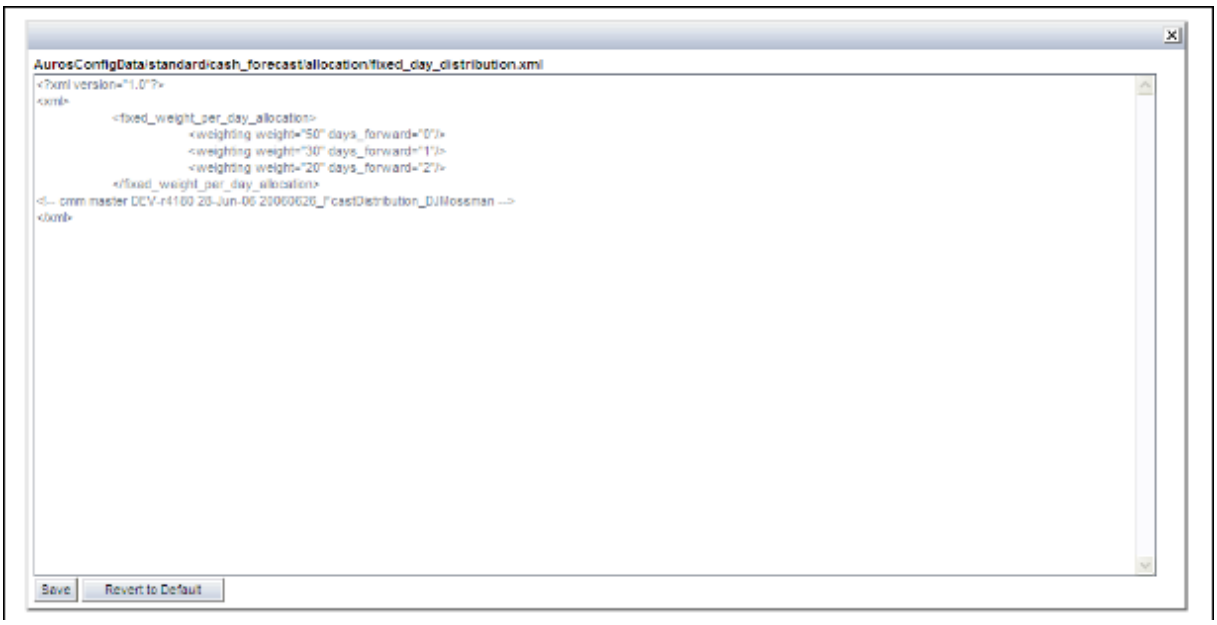
1.5.5 Opening configuration files with the Review CMM Configuration Documents function

To open configuration files with the Review CMM Configuration Documents function:

1. Select **Admin - Utilities - Setup - Review CMM Configuration Documents**.
2. In the Review Configuration Documents page:
   - To view or edit all editable configuration files, or edit a configuration file, select `Installation Config Documents` in the list.
   - To view or edit a subset of the most commonly modified editable configuration files, select `Standard Config Documents` in the list.

   You can also access files in the `Runtime` and `VirtualDirectory` folders:
   - To view a file in the `Runtime` folder, select `Runtime Directory Documents` in the list.
   - To view or edit a file in the `VirtualDirectory` folder, select `Virtual Directory Config Documents` in the list.
3. Navigate to the configuration file you want to open using the bulleted list.

   In the bulleted list, folders are represented by black bulleted list items while configuration files are represented by blue bulleted list items. To view the contents of a folder, click its bulleted list item.
4. Click the configuration file. A window opens, displaying the contents of the configuration file:



5. Edit and save the configuration file as described in this guide.

When you first edit a configuration file in the Review CMM Configuration Documents function, WebSuite creates a version of the file in the `ConfigurationData` folder. This version overrides the default version from the `DefaultData` folder. However, the function allows you to delete the custom version in the `ConfigurationData` folder and resume using the default version in the `DefaultData` folder.

1.5.6 Opening configuration files without the Review CMM Configuration Documents function

To open configuration files without the Review CMM Configuration Documents function (assuming the files is not present already in the `InstallationData` or `ConfigurationData` folder):

1. Open the `DefaultData` folder:

   <install home>\components\wss-web\websuite\DefaultData\default\AurosConfigData\standard

2. Locate and copy the configuration file to the appropriate location in the `ConfigurationData` folder.
3. Open the configuration file in a text editor.
4. Edit and save the configuration file as described in the documentation.

1.5.7 Returning a configuration file to its default settings

To return a configuration file to its default settings:

1. Select **Admin - Utilities - Setup - Review CMM Configuration Documents**.
2. In the Review Configuration Documents page:
   - To view or edit all editable configuration files, or edit a configuration file, select `Installation Config Documents` in the list.
   - To view or edit a subset of the most commonly modified editable configuration files, select `Standard Config Documents` in the list.
3. Navigate to the configuration file you want to edit using the bulleted list.

   In the bulleted list, folders are represented by black bulleted list items while configuration files are represented by blue bulleted list items. To view the contents of a folder, click its bulleted list item.
4. Click the configuration file. A window opens, displaying the contents of the configuration file.
5. Click **Revert to Default**.

   An archived version of the edited file is saved in `[WebSuite installation root]\ConfigurationData\Archive`.

   (If the file is not available in the Review CMM Configuration Documents function, you can return it to its default settings by removing it from the `ConfigurationData` folder.)

Chapter 2 Managing formats

A set of default formats are provided with CMM. You can manage these formats through two configuration files included in the CMM installation:

- `fileimportexportformats.xml` specifies import and export formats supported by CMM
- `importfilepattern.xml` specifies import format patterns supported by CMM.

To ensure optimal performance of CMM, remove all unneeded formats from the `fileimportexportformats.xml` file and all unneeded format patterns from the `importfilepattern.xml` file.

2.1 Configuring the fileimportexportformats.xml file

Like all of CMM's XML-based configuration files, the `fileimportexportformats.xml` file is located in the CMM `DefaultData` folder.

Before you can edit the `fileimportexportformats.xml` file, you must copy it from the CMM `defaultData` folder and place it in the correct location in the `ConfigurationData` folder. See "Opening configuration files" on page 32.

Note: In Chapter 7 Using XML-template-based formats on page 119, you will add custom formats to the file.

2.1.1 Prerequisites

The following are prerequisites for configuring the `fileimportexportformats.xml` file:

Category	Tasks
Security	Ensure you have access to the following function: • FG-0400 Review CMM Configuration. For more information, see the *WebSuite System Administration Guide*.

2.1.2 Removing formats from the fileimportexportformats.xml file

To remove formats from the `fileimportexportformats.xml` file:

1. In the `table_data` element, delete the `row_data` child element of the first format you want to remove.
3. Repeat step 1 for each format you want to remove.
4. Save and close the file.

2.2 Configuring the importfilepattern.xml file

Like the `fileimportexportformats.xml` file, the `importfilepattern.xml` file is located in the CMM `DefaultData` folder.

Before you can edit the `importfilepattern.xml` file, you must copy it from the CMM `DefaultData` folder and place it in the correct location in the `ConfigurationData` folder. See "Opening configuration files" on page 32.

Note: In Chapter 7 Using XML-template-based formats on page 119, you will add custom format patterns to the file.

2.2.1 Prerequisites

The following are prerequisites for configuring the `importfilepattern.xml` file:

Category	Tasks
Security	Ensure you have access to the following function: • FG-0400 Review CMM Configuration. For more information, see the *WebSuite System Administration Guide*.

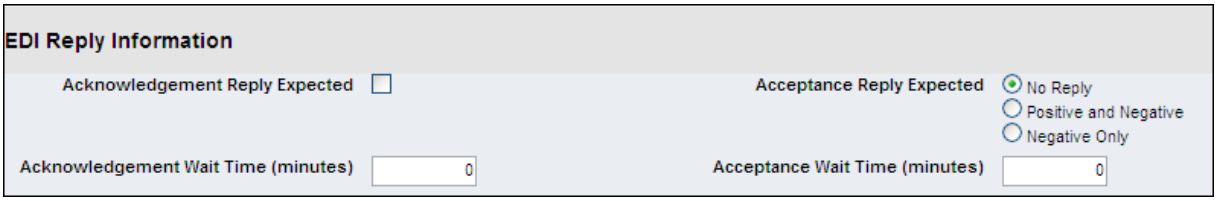
2.2.2 Removing format patterns from the importfilepattern.xml file

To remove format patterns from the `importfilepattern.xml` file:

1. In the `import_file_pattern` element, delete the `pattern` child element of the first format pattern you want to remove.
2. Repeat step 2 for each format pattern you want to remove.
3. Save and close the file.

2.2.3 Verifying the business types of bank message import formats

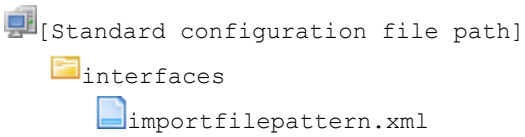
In select export format interchanges, you can specify if your organization expects to receive an acknowledgment and acceptance messages from the receiving bank for transactions released through the interchange:



If you enable these options in the export interchange, you need to ensure the corresponding bank message import formats have the correct business types.

To verify the business type of a bank message import format:

1. Open the following configuration file:



For instructions on opening configuration files, see “Opening configuration files” on page32.

2. In the `import_file_pattern` element, locate the `pattern` child element of the bank message format.
3. Verify that the `business_type` attribute is set properly:
   - `ACKNOWLEDGEMENT` for acknowledgement bank messages
   - `ACCEPTANCE` for acceptance bank messages.
4. Save and close the file.

Chapter 3 Managing interchanges (and related data)

To implement an interface in CMM, you must create parameters for its transport mechanism, security mechanisms, and format processors. You then create an interchange that references these parameters as well as specifying the interface's type, format, and connection point.

3.1 Managing transport mechanisms

A transport mechanism allows CMM to pick up files from or drop off files to the connection point without intervention from external applications. In CMM, transport mechanisms are referred to as "communication protocols".

3.1.1 Managing communication protocols

You can enable the following communication protocols in the Communication Protocols function:

- FIN Message Manager
- FTP (File Transfer Protocol)
- Generic Command Line Communication
- MQSeries
- Native Command Line Communication
- Request Response
- SWIFTNet Adaptor

3.1.1.1 Prerequisites

The following are prerequisites for managing communication protocols:

Category	Tasks
Security	Ensure you have access to the following function: • FG-0096 Communication Protocol Maintenance. For more information, see the *WebSuite System Administration Guide*.

3.1.1.2 Installing third-party software

Before enabling the MQSeries communication protocol, you must install third-party software.

To install third-party software for the MQSeries communication protocol:

1. Purchase a license of MQSeries from IBM.

   CMM currently supports version 6.0 of MQSeries.
2. Install MQSeries in a location accessible to CMM.

   For more information, see the documentation provided by IBM.

3.1.1.3 Enabling communication protocols

To enable a communication protocol:

1. Select **Admin - Static Data - Bank Interfacing - Communication Protocols**.
2. In the Communication Protocol Maintenance List page:
   - To enable the communication protocol, select its **Enabled in CMM** checkbox.
   - To allow the enabled communication protocol to be manually executed through the Task Scheduler or Communication Dispatch functions, select its **Manually Executable** checkbox.
   - To allow the enabled communication protocol to create a remote connection between two servers for remote processes, select its **Remote Connection Enabled** checkbox.
3. Click **Save**.

   If you cleared the **Enabled in CMM** checkbox of a previously enabled communication protocol, CMM does not delete or otherwise change any parameters that reference that communication protocol. To discontinue using the communication protocol, you must remove all communication protocol parameters and interchanges that reference it.

3.1.2 Managing communication protocol parameters

When a communication protocol is enabled, it displays in the Communication Protocol Parameters function. As a result, you can create parameters that reference the communication protocol and then use those parameters in interchanges.

3.1.2.1 Prerequisites

The following are prerequisites for managing communication protocol parameters:

Category	Tasks
Static data	Ensure the following static data are available: • Counterparties . For more information, see the *WebSuite User Guide*.
Interfaces	Ensure the following task has been completed: • 3.1.1 Managing communication protocols on page 41.
Security	Ensure you have access to the following function: • FG-0038 Communications Protocol Parameters Maintenance. For more information, see the *WebSuite System Administration Guide*.

3.1.2.2 Creating FIN Message Manager protocol parameters

To route SWIFT FIN messages via TRMSwift so that they are available in the Fin Message Admin application:

1. Select **Admin - Static Data - Bank Interfacing - Communication Protocol Parameters**.
2. In the Communication Protocols page, select the **Fin Message Manager** option button.
3. Click **Edit Parameter**.
4. In the Communication - Fin Message Manager Adapter page, click **New Entry**.
5. In the Communication - Fin Message Manager Adapter page, set the **Protocol Type** to FIN, and select the correct **Internal BIC** and **External BIC** as declared in the ESIAdapter Client Relationship Editor.
6. Click **Save**.

3.1.2.2.1 Using the richer XML FIN message format

There are two formats available for the FIN message(s) produced by CMM to be sent to ESIAdapter: text or XML depending on the Format Specification selected in the corresponding interchange.

To change Format Specification, follow the steps in *3.4.3 Editing interchanges* on page 57, then:

- Select a different **Format Specification**: the XML versions are prefixed `FIN_` (text versions are prefixed `STD_`). If you route SWIFT FIN messages via TRMSwift, so that they are available in the FIN Message Admin application, then you get the most benefit from using the XML version. You will see that FIN message(s) released using this set of format specifications will populate the **FIN Sequence** and **Subfield** columns of the FIN Message Field view of the FIN Message Admin. This allows full usage of the FIN Message Admin through rules and actions, as this complementary metadata enables you to identify the business reason of a line in a FIN message instead of only using technical identifiers like the **Order ID**. This makes the whole process more efficient.

To route your FIN message(s) towards ESIAdapter via TRMSwift:

- Select `Fin Message Manager` as the **Communication Type**, under the **Interchange Plugins** group. For more information, see *3.4.3 Editing interchanges* on page 57.

3.1.2.3 Creating FTP communication protocol parameters

To create an FTP communication protocol parameter:

1. Select **Admin - Static Data - Bank Interfacing - Communication Protocol Parameters**.
2. In the Communication Protocols page, select the **FTP - File Transfer Protocol** option button.
3. Click **Edit Parameter**.
4. In the Communication - FTP [list] page, click **New Entry**.
5. In the Communication - FTP [editor] page, create the communication protocol parameter.

   The following is an example FTP communication protocol parameter:
6. Click **Save**.

3.1.2.4 Creating MQSeries communication protocol parameters

The number of MQSeries queues (communication protocol parameter sets) that is required depends on the number of interchanges configured in the system. Typically each of these interchanges uses a distinct communication protocol parameter set with each set defining its own queue either for sending or receiving.

CMM itself does not impose any restraints on the MQSeries naming convention; rules for naming thus depends on MQSeries itself, and other allied systems that consume and produce messages.

To create an MQSeries communication protocol parameter:

1. Select **Admin - Static Data - Bank Interfacing - Communication Protocol Parameters**.
2. In the Communication Protocols page, select the **MQ Series Communication** option button.
3. Click **Edit Parameter**.
4. In the Communication - MQ Series [list] page, click **New Entry**.
5. In the Communication - MQ Series [editor] page, create the communication protocol parameter.
6. Click **Save**.

When an interchange with an MQSeries communication protocol parameter encounters an error during import, it attempts to post the transaction data to a queue. You can then view the transaction data in the queue. To facilitate this, you must create an MQSeries submit communication protocol parameter with the following value in the **Parameter Set Name** field:

```
MQ SERIES ERROR STREAM (DO NOT CHANGE NAME)
```

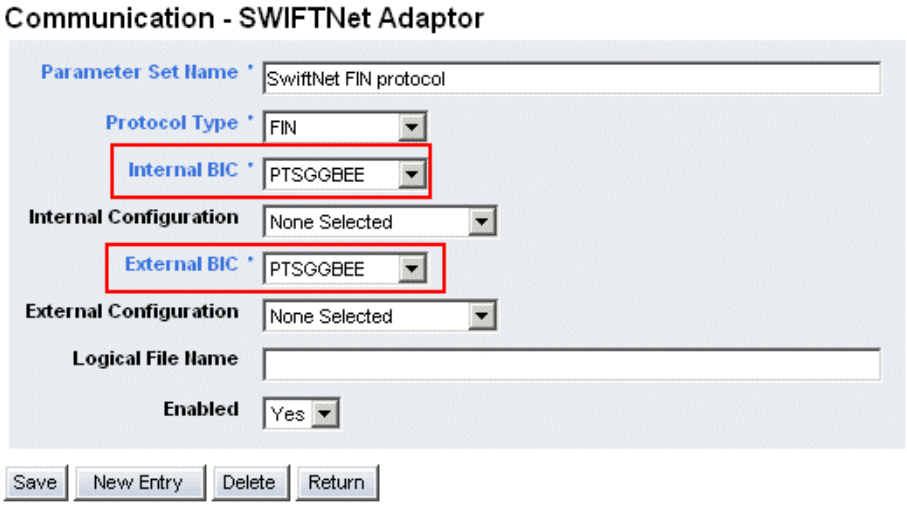
3.1.2.5 Creating SWIFTNet Adaptor communication protocol parameters

To create a SWIFTNet Adaptor communication protocol parameter:

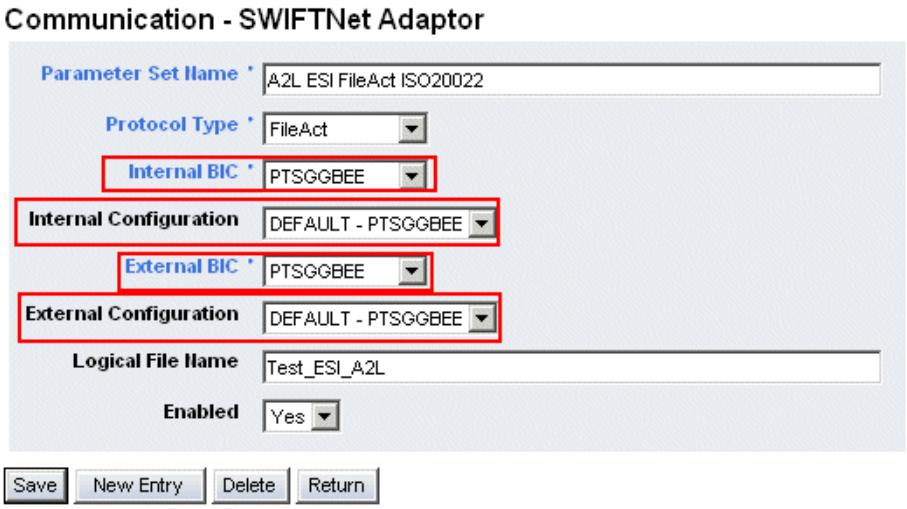
1. Select **Admin - Static Data - Bank Interfacing - Communication Protocol Parameters**.
2. In the Communication Protocols page, select the **SWIFTNet Adaptor** option button.
3. Click **Edit Parameter**.
4. In the Communication - SWIFTNet Adaptor [list] page, click **New Entry**.
5. In the Communication - SWIFTNet Adaptor [editor] page, create the communication protocol parameter.
6. Click **Save**.

   The following are examples of SWIFTNet Adaptor communication protocol parameters:

   FIN



   FileAct



Note that the Internal and External configuration setup is not required for the FIN protocol type. Internal and external BICs are set in the ESIAdapter Client Relationship Editor (also known as ESIAdapter Editor), see the *Wallstreet Suite SWIFT Connectivity Guide*.

3.1.2.6 Creating command line communication protocol parameters

To create a command line communication protocol parameter:

1. Select **Admin - Static Data - Bank Interfacing - Communication Protocol Parameters**.
2. In the Communication Protocols page, select the **Generic Command Line Communication** option button.
3. Click **Edit Parameter**.
4. In the Communication - Generic Command Line [list] page, click **New Entry**.
5. In the Communication - Generic Command Line [editor] page, create the communication protocol parameter.

   For more information on command line communication protocol parameters, see 6.1 Connecting interface components on page 99.
6. Click **Save**.

3.1.2.7 Creating request response communication protocol parameters

To create a request response communication protocol parameter:

1. Select **Admin - Static Data - Bank Interfacing - Communication Protocol Parameters**.
2. In the Communication Protocols page, select the **Request Response** option button.
3. Click **Edit Parameter**.
4. In the Communication - Request Response [list] page, click **New Entry**.
5. In the Communication - Request Response [editor] page, create the communication protocol parameter.

   For more information on request response communication protocol parameters, see 5.2.3 Managing the interchange (and related data) on page 76 and 5.3.3 Managing the interchange (and related data) on page 82.
6. Click **Save**.

3.1.2.8 Editing communication protocol parameters

To edit a communication protocol parameter:

1. Select **Admin - Static Data - Bank Interfacing - Communication Protocol Parameters**.
2. In the Communication Protocols page, select the appropriate communication protocol's option button.
3. Click **Edit Parameter**.
4. In the communication protocol's parameter list page, drill down on the communication protocol parameter.
5. In the communication protocol's parameter editor page, edit the communication protocol parameter.

   For descriptions of the controls on this page, see the appropriate Creating section.
6. Click **Save**.

3.1.2.9 Deleting communication protocol parameters

To delete a communication protocol parameter:

1. Select **Admin - Static Data - Bank Interfacing - Communication Protocol Parameters**.
2. In the Communication Protocols page, select the appropriate communication protocol's option button.
3. Click **Edit Parameter**.
4. In the communication protocol's parameter list page, drill down on the communication protocol parameter.
5. In the communication protocol's parameter editor page, click **Delete**.
6. In the resulting dialog, click **OK**.

3.1.3 Completing other tasks

In addition to managing communication protocol parameters, you can complete the following tasks after enabling communication protocols:

- Manually execute communication protocols
- Run processes remotely through communication protocols.

3.1.3.1 Prerequisites

The following are prerequisites for completing other tasks:

Category	Tasks
Interfaces	Ensure the following task has been completed: • 3.1.1 Managing communication protocols on page 41.
Security	Ensure you have access to the following functions: • FG-0215 Communication Dispatch • FG-0303 Remote Process Maintenance. For more information, see the *WebSuite System Administration Guide*.

3.1.3.2 Manually executing communication protocols

If you selected a communication protocol's **Manually Executable** checkbox in the Communication Protocols function, you can manually execute that communication protocol by scheduling the appropriate Communication task in the Task Scheduler function. For more information, see the *WebSuite System Administration Guide*.

You can also manually execute communication protocols on a one-time basis using the Communication Dispatch function:

1. Select **Admin - Utilities - Bank Interfacing - Communication Dispatch**.
2. In the Communication Dispatch page, select the checkboxes of communication protocols you want to manually execute.
3. Click **Dispatch Messages**.

3.1.3.3 Running processes remotely through communication protocols

If you selected a communication protocol's **Remote Connection Enabled** checkbox in the Communication Protocols function, you can run processes remotely through that communication protocol in the Remote Processes function.

Note: For information on specific processes, contact Wallstreet.

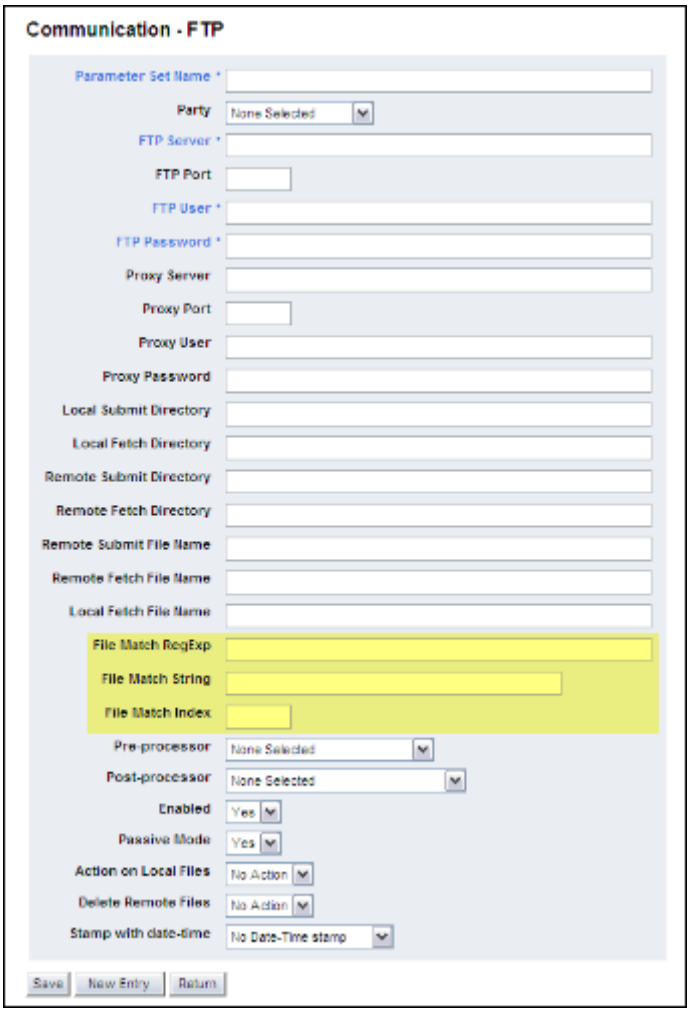
To run a process remotely through a communication protocol:

1. Select **Admin - Utilities - Setup - Remote Processes**.
2. In the Remote Process Maintenance page, drill down on the process.
3. In the Remove [Process] Communication Protocol page, select the communication protocol's **Remote** option button.
4. Click **Save**.
5. In the communication protocol's parameter editor page, create a communication protocol parameter for the process.

   For descriptions of the controls on this page, see 3.1.2 Managing communication protocol parameters on page 42.
6. Click **Save**.

3.1.4 Entering data in FTP communication protocol parameter file matching fields

If you are creating or editing an FTP communication protocol parameter in the Communication Protocol Parameters function, three fields related to file matching display on the Communication – FTP [editor] page:



3.1.4.1 Entering data in the File Match RegExp field

The **File Match RegExp** field specifies the file or files CMM is to retrieve from the FTP server. You need to enter a valid regular expression in this field so that CMM can recognize which file or files to retrieve from the FTP server. Therefore, it is important to update this field if there are any changes to the standard file names or folder structure on the FTP server.

3.1.4.1.1 Regular expression syntax

The following table presents syntax you can use in regular expressions:

Syntax	Purpose
.	Matches any character except a new line.
^	Matches the start of the string and, in multiline mode, also matches immediately after each new line.
$	Matches the end of the string and, in multiline mode, also matches before a new line. Example: fo matches fo and food, but fo$ only matches fo.
*	Causes a match of zero or more repetitions of the preceding regular expression. Example: ab* matches a, ab, or abbb... (a followed by two or more bs).
+	Causes a match of one or more repetitions of the preceding regular expression. Example: ab+ matches ab or abbb... (a followed by two or more bs) but does not match a (a followed by no bs).
?	Causes a match of zero or one repetition of the preceding regular expression. Example: ab? matches a or ab. Adding ? after the qualifiers (*, +, and ?) makes them perform the match in a minimal fashion, where as few characters as possible are matched. Examples: If <.*> is matched against <H1>title</H>, it matches the entire string. If <.*?> is matched against <H1>title</H>, it only matches <H1>.
{*m*, *n*}	Causes a match from m to n repetitions of the preceding regular expression, attempting to match as many repetitions as possible. Example: a{3,5} matches from three to five a characters. Omitting n specifies an infinite upper bound. Example: a{3} matches from three to infinity a characters.
{*m*, *n*}?	Causes a match from m to n repetitions of the preceding regular expression, attempting to match as few repetitions as possible. Examples: If a{3,5} is matched against aaaaaa, it matches five a characters. If a{3,5}? is matched against aaaaaa, it only matches three a characters.

Syntax	Purpose
`[]`	Indicates a set of characters. Characters can be listed individually, or a range of characters can be indicated by listing two characters separated by a hyphen. Special characters are not active inside sets. Examples: [akm$] matches any of the characters a, k, m, or $. [a-z] matches any lowercase letter. [a-zA-Z0-9] matches any letter or digit. Character classes, such as \w or \S, are also accepted inside a range. If you want to include a closing bracket or hyphen inside a set, precede it with a back slash or place it as the first character. Example: []] matches]. You can also match the characters not within a range by including a ^ as the first character of the set. Example: [^5] matches any character except 5 (^ elsewhere simply matches the ^ character.)
A|*B*	Creates a regular expression that matches either A or B (where A and B are regular expressions). This can be used inside groups as well. To match |, use \\| or enclose it inside a character class (for example, [|]).
\	Escapes special characters, enabling you to match characters such as asterisk and question mark. Example: * matches *. Back slashes (\) can also signal special sequences: • *number* matches the contents of the group of the same number. Groups are numbered starting from one. Example: (.+)\1 matches the the or 55 55 but not the end. This special sequence can only be used with numbers 1 to 99. If the first digit of a number is 0 or the number is three digits long, it will not be interpreted as a group match. • \A matches only at the start of the string • \b matches the empty string, but only at the beginning or end of a word. A word is defined as a sequence of alphanumeric characters, so the end of a word is indicated by white space or a non-alphanumeric character. Inside a character range, \b represents the backspace character for compatibility with Python's string literals. • \B matches the empty string, but only when it is not at the beginning or end of a word. • \d matches any decimal digit. This is equivalent to the set [0-9]. • \D matches any non-digit character. This is equivalent to the set [^0-9]. • \s matches any white space character. This is equivalent to the set [\t\n\r\f\v]. • \S matches any non-white space character. This is equivalent to the set [^ \t\n\r\f\v]. • \w matches any alphanumeric character. This is equivalent to the set [a-zA-Z0-9_]. • \W matches any non-alphanumeric character. This is equivalent to the set [^a-zA-Z0-9_]. • \Z matches only at the end of the string. • \\ matches a literal back slash.

Syntax	Purpose
(*A*)	Matches the regular expression A inside the parentheses and indicates the start and end of a group. The content of a group can be retrieved after a match has been performed and can be matched later in the string with the *number* special sequence. To match (and), use \(and \) or enclose them inside character classes (for example, [(] and [)]).

3.1.4.1.2 Example

The following string:

```
(.*\s+)(\d+\s+)(\S+\s+)(\d+)(\s+\w{3}\s+\d{1,2}\s+\d{1,2}:?\d{0,2}\s+)([a-z,A-Z]
*\.?[B,b][A,a][I,i])\s
```

Matches:

```
-rw-r--r-- 1 treasury 2439 Aug 30 09:01 CitiDEBAI
```

The following table explains why this is the case:

No.	Regular expression	Matching text	Explanation
1	`(.*\s+)`	`-rw-r--r--`	This expression matches any characters for zero or more repetitions followed by one or more white space characters.
2	`(\d+\s+)`	`1`	This expression matches any decimal digits for one or more repetitions followed by one or more white space characters.
3	`(\S+\s+)`	`treasury`	This expression matches any non-white space character for one or more repetitions followed by one or more white space characters.
4	`(\d+)`	`2439`	This expression matches any decimal digits for one or more repetitions.
5	`(\s+\w{3}\s+\d{1,2}\s+\d{1,2}:?\d{0,2}\s+)`	`Aug 30 09:01`	This expression matches any one or more white space characters followed by any three alphanumeric characters, followed by one or more white space characters, followed by one to two decimal digits, followed by one or more white space characters, followed by one to two decimal digits, followed by zero or one colons, followed by zero to two decimal digits, followed by one or more white space characters.
6	`([a-z,A-Z]*\.?[B,b][A,a][I,i])\s`	`CitiDEBAI`	This expression matches any letter for zero or more repetitions, followed by zero or one periods, followed by BAI or bai, followed by any white space character.

3.1.4.2 Entering data in the File Match String field

Wallstreet recommends you enter the following value in the **File Match String** field:

```
(.*\s+)(\d+\s+)(\S+\s+)(\d+)(\s+\w{3}\s+\d{1,2}\s+\d{1,2}:?\d{0,2}\s+)([a-z,A-Z]*\.?
[B,b][A,a][I,i])\s
```

3.1.4.3 Entering data in the File Match Index field

After you have entered a value in the **File Match RegExp** field, you need to enter a corresponding value in the **File Match Index** field. This field indicates which group matches the file name.

If you entered the example described in 3.1.4.1 Entering data in the File Match RegExp field on page 48 in the **File Match RegExp** field, enter `6` in the **File Match Index** field because the sixth group, `([a-z,A-Z]*\.?[B,b][A,a][I,i])\s`, matches the file name, `CitiDEBAI`.

3.2 Managing security mechanisms

A security mechanism ensures a file cannot be tampered with while being transported to or from CMM. In CMM, security mechanisms are referred to as "signers".

3.2.1 Managing signers

You can enable the following signers in the Signers function:

- Entrust
- GnuPG
- PGP
- safeX
- Command line.

3.2.1.1 Prerequisites

The following are prerequisites for managing signers:

Category	Tasks
Security	Ensure you have access to the following function: • FG-0183 Signer Maintenance. For more information, see the *WebSuite System Administration Guide*.

3.2.1.2 Installing third-party software

Before enabling the Entrust, GnuPG, PGP, and safeX signers, you must install third-party software.

To install third-party software for the Entrust signers:

1. Download the current version of the CMM application from the UIS section of the Wallstreet support site.

   The Wallstreet support site is password protected. If you do not have a password for the site, contact Wallstreet.

2. Copy the `websuite-entrust.jar` file to the `$JAVA_HOME\jre\lib\ext` directory. This is the same JDK that is used to run Websuite.
3. Take the XCC file which was obtained from Citibank, rename the file to `entrust.xcc` and copy it to the `$JAVA_HOME\jre\lib\ext` directory.
4. Take the file `enttoolkit.jar` which is a component of the Entrust Toolkit V7 obtained from Entrust, and copy it to the `%JAVA_HOME%\jre\lib\ext` directory.

To install third-party software for the GnuPG signers:

1. Download GnuPG from the GnuPG website.

   CMM currently supports version 1.2.1 of GnuPG.

2. Install GnuPG in a location accessible to CMM.

   For more information, see the documentation provided on the GnuPG website.

3. Log into CMM and set the GnuPG Home Directory configuration parameter.

   For information on setting this configuration parameter, see 1.4.4 Setting interface configuration parameters on page 27.

To install third-party software for the PGP signers:

1. Purchase a license of McAfee E-Business Server from McAfee Inc.

CMM currently supports version 7.0 of McAfee E-Business Server.

2. Install McAfee E-Business Server in …\3rdPartyProgDir\pgp\.

   For more information, see the documentation provided by McAfee Inc.

To install third-party software for the safeX signers:

1. Purchase a license of safeX from R&L AG.

   CMM currently supports version 2.1 of safeX.

2. Install safeX in a location accessible to CMM.

   For more information, see the documentation provided by R&L AG.

3. Log into CMM and set the SafeX 2.x File Location configuration parameter.

   For information on setting this configuration parameter, see 1.4.4 Setting interface configuration parameters on page 27.

3.2.1.3 Enabling signers

To enable a signer:

1. Select **Admin - Static Data - Bank Interfacing - Signers**.
2. In the Signer Maintenance List page, select the signer's **Installed** checkbox.
3. Click **Save**.

   If you cleared the **Installed** checkbox of a previously enabled signer, CMM does not delete or otherwise change any parameters that reference that signer. To discontinue using the communication protocol, you must remove all signer parameters and interchanges that reference it.

3.2.2 Managing signer parameters

When a signer is enabled, it displays in the Signer Parameters function. As a result, you can create parameters that reference the signer and then use those parameters in interchanges.

3.2.2.1 Prerequisites

The following are prerequisites for managing signer parameters:

Category	Tasks
Static data	Ensure the following static data are available: • Counterparties. For more information, see the *WebSuite User Guide*.
Interfaces	Ensure the following task has been completed: • 3.2.1 Managing signers on page 51.
Security	Ensure you have access to the following function: • FG-0184 Signer Parameter Maintenance. For more information, see the *WebSuite System Administration Guide*.

3.2.2.2 Creating Entrust encryption signer parameters

To create an Entrust encryption signer parameter:

1. Select **Admin - Static Data - Bank Interfacing - Signer Parameters**.
2. In the Signer Parameter Maintenance page, drill down on **Generic Entrust Encryption**.
3. In the Generic Entrust Encryption [list] page, click **New Entry**.
4. In the Generic Entrust Encryption [editor] page, create the signer parameter.

   The following is an example Entrust encryption signer parameter:
5. Click **Save**.

3.2.2.3 Creating Entrust decryption signer parameters

To create an Entrust decryption signer parameter:

1. Select **Admin - Static Data - Bank Interfacing - Signer Parameters**.
2. In the Signer Parameter Maintenance page, drill down on **Generic Entrust Decryption**.
3. In the Generic Entrust Decryption [list] page, click **New Entry**.
4. In the Generic Entrust Decryption [editor] page, create the signer parameter.
5. Click **Save**.

3.2.2.4 Creating GnuPG encryption signer parameters

To create a GnuPG encryption signer parameter:

1. Select **Admin - Static Data - Bank Interfacing - Signer Parameters**.
2. In the Signer Parameter Maintenance page, drill down on **Generic GnuPG Encryption**.
3. In the Generic GnuPG Encryption [list] page, click **New Entry**.
4. In the Generic GnuPG Encryption [editor] page, create the signer parameter.
5. Click **Save**.

3.2.2.5 Creating GnuPG decryption signer parameters

To create a GnuPG decryption signer parameter:

1. Select **Admin - Static Data - Bank Interfacing - Signer Parameters**.
2. In the Signer Parameter Maintenance page, drill down on **Generic GnuPG Decryption**.
3. In the Generic GnuPG Decryption [list] page, click **New Entry**.
4. In the Generic GnuPG Decryption [editor] page, create the signer parameter.
5. Click **Save**.

3.2.2.6 Creating PGP encryption signer parameters

To create a PGP encryption signer parameter:

1. Select **Admin - Static Data - Bank Interfacing - Signer Parameters**.
2. In the Signer Parameter Maintenance page, drill down on **Generic PGP Encryption**.
3. In the Generic PGP Encryption [list] page, click **New Entry**.
4. In the Generic PGP Encryption [editor] page, create the signer parameter.
5. Click **Save**.

3.2.2.7 Creating PGP decryption signer parameters

To create a PGP decryption signer parameter:

1. Select **Admin - Static Data - Bank Interfacing - Signer Parameters**.
2. In the Signer Parameter Maintenance page, drill down on **Generic PGP Decryption**.
3. In the Generic PGP Decryption [list] page, click **New Entry**.
4. In the Generic PGP Decryption [editor] page, create the signer parameter.
5. Click **Save**.

3.2.2.8 Creating safeX encryption signer parameters

To create a safeX encryption signer parameter:

1. Select **Admin - Static Data - Bank Interfacing - Signer Parameters**.
2. In the Signer Parameter Maintenance page, drill down on **Generic safeX Encryption**.
3. In the Generic SafeX Encryption [list] page, click **New Entry**.
4. In the Generic SafeX Encryption [editor] page, create the signer parameter.
5. Click **Save**.

3.2.2.9 Creating safeX decryption signer parameters

To create a safeX decryption signer parameter:

1. Select **Admin - Static Data - Bank Interfacing - Signer Parameters**.
2. In the Signer Parameter Maintenance page, drill down on **Generic SafeX Decryption**.
3. In the Generic SafeX Decryption [list] page, click **New Entry**.
4. In the Generic SafeX Decryption [editor] page, create the signer parameter.
5. Click **Save**.

3.2.2.10 Creating command line encryption signer parameters

To create a command line encryption signer parameter:

1. Select **Admin - Static Data - Bank Interfacing - Signer Parameters**.
2. In the Signer Parameter Maintenance page, drill down on **Generic Command Line Encryption**.
3. In the Generic Command Line Encryption [list] page, click **New Entry**.
4. In the Generic Command Line Encryption [editor] page, create the signer parameter.

   For more information on command line encryption signer parameters, see 6.1 Connecting interface components on page 99.
5. Click **Save**.

3.2.2.11 Creating command line decryption signer parameters

To create a command line decryption signer parameter:

1. Select **Admin - Static Data - Bank Interfacing - Signer Parameters**.
2. In the Signer Parameter Maintenance page, drill down on **Generic Command Line Decryption**.
3. In the Generic Command Line Decryption [list] page, click **New Entry**.
4. In the Generic Command Line Decryption [editor] page, create the signer parameter.

For more information on command line decryption signer parameters, see 6.1 Connecting interface components on page 99.

5. Click **Save**.

3.2.2.12 Editing signer parameters

To edit a signer parameter:

1. Select **Admin - Static Data - Bank Interfacing - Signer Parameters**.
2. In the Signer Parameter Maintenance page, drill down on the appropriate signer.
3. In the signer's parameter list page, drill down on the signer parameter.
4. In the signer's parameter editor page, edit the signer parameter.

   For descriptions of the controls on this page, see the appropriate Creating section.
5. Click **Save**.

3.2.2.13 Deleting signer parameters

To delete a signer parameter:

1. Select **Admin - Static Data - Bank Interfacing - Signer Parameters**.
2. In the Signer Parameter Maintenance page, drill down on the appropriate signer.
3. In the signer's parameter list page, drill down on the signer parameter.
4. In the signer's parameter editor page, click **Delete**.
5. In the resulting dialog, click **OK**.

3.3 Managing format processors

When importing or exporting a file, CMM can transform it from one format to another according to the format processor parameters in the file's interchange:

- For an import process, the format processor transforms the data file from a user-specified format to an appropriate CMM import format. It then stores the results in a specified output file so that CMM can retrieve the file and continue processing.
- For an export process, the format processor transforms the data file from the appropriate CMM export format to a user-specified format that can be processed by an external system such as a bank. It stores the results in the specified output data file so that CMM can retrieve the file and continue processing.

3.3.1 Prerequisites

The following are prerequisites for managing format parameters:

Category	Tasks
Static data	Ensure the following static data are available: • Counterparties. For more information, see the *WebSuite User Guide*.
Security	Ensure you have access to the following function: • FG-0359 Format Processor Parameters. For more information, see the *WebSuite System Administration Guide*.

3.3.2 Creating format processor parameters

To create a format processor parameter:

1. Select **Admin - Static Data - Bank Interfacing - Command Line Processors**.
2. In the Format Processor Parameters page, drill down on the appropriate format processor.
3. In the Command Line Format Processor [list] page, click **New Entry**.
4. In the Command Line Format Processor [editor] page, create the format processor parameter.
5. Click **Save**.

3.3.3 Editing format processor parameters

To edit a format processor parameter:

1. Select **Admin - Static Data - Bank Interfacing - Command Line Processors**.
2. In the Format Processor Parameters page, drill down on the appropriate format processor.
3. In the Command Line Format Processor [list] page, drill down on the format processor parameter.
4. In the Command Line Format Processor [editor] page, edit the format processor parameter.
5. Click **Save**.

3.3.4 Deleting format processor parameters

To delete a format processor parameter:

1. Select **Admin - Static Data - Bank Interfacing - Command Line Processors**.
2. In the Format Processor Parameters page, drill down on the appropriate format processor.
3. In the Command Line Format Processor [list] page, drill down on the format processor parameter.
4. In the Command Line Format Processor [editor] page, click **Delete**.
5. In the resulting dialog, click **OK**.

3.4 Managing interchanges

When you have an interface's format, connection point, communication protocol parameter, signer parameters, and format processor parameters, you can create an interchange for that interface. Afterwards, you can assign the interchange to import and export parameter sets in the Task Scheduler function.

This section documents a general procedure for creating and maintaining interchanges.

Note: Each interchange has a unique name. If you do not specify the name, CMM creates it using the values in the **Business Function**, **Priority Status**, **Format Specification**, **Domestic/Cross-Border**, **Entity**, and **Counterparty** (or **Bank**) controls.

3.4.1 Prerequisites

The following are prerequisites for configuring interchanges:

Category	Tasks
Static data	Ensure the following static data are available: • Currencies • Payment methods • Entities • Counterparties • Banks • Cash flow types. For more information, see the *WebSuite User Guide*.
Interfaces	Ensure the following tasks have been completed: • Chapter 2 Managing formats on page 37 • 3.1.2 Managing communication protocol parameters on page 42 • 3.2.2 Managing signer parameters on page 52 • 3.3 Managing format processors on page 55.
Security	Ensure you have access to the following functions: • FG-0175 Interchanges • FG-0400 Review CMM Configuration. For more information, see the *WebSuite System Administration Guide*.

3.4.2 Creating interchanges

To create an interchange:

1. Select **Admin - Static Data - Bank Interfacing - Interchanges**.
2. In the Interchange - Criteria Selection page, click **Search**.
3. In the Interchange Maintenance List page, click **New Entry**.
4. In the Select Interchange Type - Criteria Selection page, enter an import/export type, business function, and format category.
5. Click **Continue**.
6. In the Interchange Maintenance page, create the interchange.
7. Click **Save**.

3.4.3 Editing interchanges

To edit an interchange:

1. Select **Admin - Static Data - Bank Interfacing - Interchanges**.
2. In the Interchange - Criteria Selection page, enter search criteria.
3. Click **Search**.
4. In the Interchange Maintenance List page, drill down on the interchange.
5. In the Interchange Maintenance page, edit the interchange.

   For descriptions of the controls on this page, see the appropriate Creating section.

(You can edit the interchange's import/export type, business function, and format category by clicking **Modify Interchange Type**.)

6. Click **Save**.

3.4.4 Deleting interchanges

To delete an interchange:

1. Select **Admin - Static Data - Bank Interfacing - Interchanges**.
2. In the Interchange - Criteria Selection page, enter search criteria.
3. Click **Search**.
4. In the Interchange Maintenance List page, select the interchange's checkbox.
5. Click **Delete**.

3.4.5 Enabling and disabling interchanges

To enable or disable an interchange:

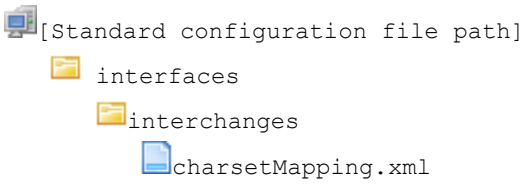
1. Select **Admin - Static Data - Bank Interfacing - Interchanges**.
2. In the Interchange - Criteria Selection page, enter search criteria.
3. Click **Search**.
4. In the Interchange Maintenance List page, select the interchange's checkbox.
5. Do one of the following:
   - To enable the interchange, click **Enable**.
   - To disable the interchange, click **Disable**.

3.4.6 Linking interchanges to character sets

By default, all interchanges in CMM use the UTF-8 character set. If you need to import or export data containing characters not in UTF-8, you can edit the `charsetMapping.xml` file.

To link interchanges to character sets:

1. Open the following configuration file:

   For instructions on opening configuration files, see "Opening configuration files" on page32.

2. Do the following:
   - To define a default character set other than UTF-8, change the value of the `charset` attribute in the default `interchange` element.
   - To override the default character set for a specific interchange, create an `interchange` element for that interchange. The following is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<interchanges>
    <interchange interchange_id="default" charset="UTF-8"/>
   <interchange interchange_id="138" charset="ISO-8859-1"/>
```

```
    </interchanges>
```

3. Save and close the file.

3.4.7 Optimizing interchanges

When you edit an import interchange (AP, DD, AR) and do not select a format type, you can allot extra computer resources to the interchange by checking the **Multithreaded for parsing** checkbox. This activates the parsing of the files in a multithreaded mode.

When this checkbox is checked, multithreaded parsing only occurs when the file being parsed has a minimum number of lines.

This minumum number is set in the runtime_parameters.xml file, located in the `etc/wss-web/cmm/InstallationData/installation folder`. The entry below defines the maximum number of lines in a file before splitting.

```
<name name="maxTransactionNumberInStandardFormatImportBeforeSplittingFile">
  <description/>
  <value>100</value>
</name>
```

The purpose of this parameter is to ensure that you do not assign computer resources needlessly for a small file. If your interchange is configured as multithreaded, but the file that you are going to import is too small (number of lines is less than the parameter) the system parses the file in a single thread mode.

Chapter 4 Managing other interface data

The following are data outside of interchanges that are relevant to the setup and configuration of interfaces:

- Transaction subtype mappings
- Branch qualifiers.

4.1 Managing transaction subtype mappings

All bank files contain unique codes to classify their contents. CMM refers to these codes as counterparty file codes. You can map each counterparty file code to an internal file code called a transaction subtype. This allows you to track where information from each file will appear and behave within CMM. For example, during the bank transaction import process, CMM matches the internal file codes to their mapped counterparty file codes and displays the transaction information in cash records, the cash flow forecast, and the general ledger.

4.1.1 Managing internal file codes

CMM has a standard list of internal file codes. You can add to this list to suit your needs. The standard internal file codes are numbered 1 to 999. The internal file codes you enter are numbered 1,000 or higher.

You cannot delete an internal file code that is mapped to counterparty file codes. You must remove the counterparty file code mapping first.

4.1.1.1 Prerequisites

The following are prerequisites for managing internal file codes:

Category	Tasks
Static data	Ensure the following static data are available: • Cash flow types. For more information, see the *WebSuite User Guide*.
Security	Ensure you have access to the following function: • FG-0085 Transaction Subtype Mapping. For more information, see the *WebSuite System Administration Guide*.

4.1.1.2 Creating internal file codes

To create an internal file code:

1. Select **Admin - Static Data - Bank Interfacing - Transaction Subtype Mapping**.
2. In the Transaction Sub Type Maintenance page, click **Internal File Codes**.
3. In the Internal File Code - Criteria Selection page, enter search criteria.
4. Click **Search**.
5. In the Internal File Code List page, click **New Entry**.
6. In the Internal File Code Maintenance page, create the internal file code.
7. Click **Save**.

4.1.1.3 Editing internal file codes

To edit an internal file code:

1. Select **Admin - Static Data - Bank Interfacing - Transaction Subtype Mapping**.
2. In the Transaction Sub Type Maintenance page, click **Internal File Codes**.
3. In the Internal File Code - Criteria Selection page, enter search criteria.
4. Click **Search**.
5. In the Internal File Code List page, drill down on the internal file code.
6. In the Internal File Code Maintenance page, edit the internal file code.

   Only the Cash Flow Type list is editable for internal file codes with IDs less than 1,000. (The internal file codes are Wallstreet defined.)
7. Click **Save**.

4.1.1.4 Deleting internal file codes

To delete an internal file code:

1. Select **Admin - Static Data - Bank Interfacing - Transaction Subtype Mapping**.
2. In the Transaction Sub Type Maintenance page, click **Internal File Codes**.
3. In the Internal File Code - Criteria Selection page, enter search criteria.
4. Click **Search**.
5. In the Internal File Code List page, drill down on the internal file code.

   Internal file codes with IDs less than 1,000 are Wallstreet defined and cannot be deleted.
6. In the Internal File Code Maintenance page, click **Delete**.
7. In the resulting dialog, click **OK**.

4.1.2 Managing counterparty file codes

CMM needs counterparty file codes to interpret the information within the files and map it to internal file codes.

When you enter a counterparty file code, you can map it to an internal file code. You can later view the counterparty file codes mapped to an internal file code in the Internal File Code List page.

Each counterparty file code associates a file format and a counterparty. Each counterparty file code can be mapped to only one internal file code under the same counterparty and file format.

Note: In the Internal File Code List page, you can view counterparty file codes mapped to an internal file code and create, edit, and delete counterparty file codes.

4.1.2.1 Prerequisites

The following are prerequisites for managing counterparty file codes:

Category	Tasks
Static data	Ensure the following static data are available: • Counterparties. For more information, see the *WebSuite User Guide*.
Interfaces	Ensure the following task has been completed: • 4.1.1 Managing internal file codes on page 61.
Security	Ensure you have access to the following function: • FG-0085 Transaction Subtype Mapping. For more information, see the *WebSuite System Administration Guide*.

4.1.2.2 Creating counterparty file codes

To create a counterparty file code:

1. Select **Admin - Static Data - Bank Interfacing - Transaction Subtype Mapping**.
2. In the Transaction Sub Type Maintenance page, click **Counterparty File Codes**.
3. In the Import Export Type Selection page, select the appropriate import or export type in the **Import Export Type** list.
4. Click **Continue**.
5. In the Counterparty File Code - Criteria Selection page, enter search criteria.
6. Click **Search**.
7. In the Counterparty File Code List page, click **New Entry**.
8. In the Counterparty File Code Maintenance page, create the counterparty file code.
9. Click **Save**.

4.1.2.3 Editing counterparty file codes

To edit a counterparty file code:

1. Select **Admin - Static Data - Bank Interfacing - Transaction Subtype Mapping**.
2. In the Transaction Sub Type Maintenance page, click **Counterparty File Codes**.
3. In the Import Export Type Selection page, select the appropriate import or export type in the **Import Export Type** list.
4. Click **Continue**.
5. In the Counterparty File Code - Criteria Selection page, enter search criteria.
6. Click **Search**.
7. In the Counterparty File Code List page, drill down on the counterparty file code.
8. In the Counterparty File Code Maintenance page, edit the counterparty file code.
9. Click **Save**.

4.1.2.4 Deleting counterparty file codes

To delete a counterparty file code:

1. Select **Admin - Static Data - Bank Interfacing - Transaction Subtype Mapping**.
2. In the Transaction Sub Type Maintenance page, click **Counterparty File Codes**.
3. In the Import Export Type Selection page, select the appropriate import or export type in the **Import Export Type** list.
4. Click **Continue**.
5. In the Counterparty File Code - Criteria Selection page, enter search criteria.
6. Click **Search**.
7. In the Counterparty File Code List page, drill down on the counterparty file code.
8. In the Counterparty File Code Maintenance page, click **Delete**.
9. In the resulting dialog, click **OK**.

4.1.2.5 Managing counterparty file codes for SWIFT ACK and NACK bank messages

If your organization imports SWIFT ACK bank messages, it needs to create a counterparty file code with the following values:

Attribute	Value
Import/export type	`Bank Messages Import`
Counterparty	`Default`
File format	`SWIFT_ACK - Bank Messages Import`
File code [1]	`007`
File code description	[Appropriate description]
Internal file code [2]	`Bank Message Accepted`
Code domain	`Messages`

Table notes:

1. All file codes provided by the external party must be stored with leading zeros (for example, `007` rather than `7`).
2. This assumes the Bank Message Accepted internal file code has already been created.

The counterparty file code for SWIFT NACK bank messages is provided by SWIFT in the `405:` tag of the file, but there may be bank-specific codes as well. These need to be defined as counterparty file codes in theTransaction Subtype Mapping function and mapped to the Bank Message Rejected internal file code if your organization imports SWIFT NACK bank messages.

4.1.2.6 Managing mappings

To manage mappings:

1. Select **Admin - Static Data - Bank Interfacing - Transaction Subtype Mapping**.
2. In the Transaction Sub Type Maintenance page, click **Internal File Codes**.
3. In the Internal File Code - Criteria Selection page, enter search criteria.
4. Click **Search**.
5. In the Internal File Code List page, click **View** in the internal file code's row.
6. In the Internal File Code Mapping List page:
   - To create a new counterparty file code, click **Add Mapping**... and follow the procedure in Creating counterparty file codes.
   - To edit an existing counterparty file code, drill down on it and follow the procedure in Editing counterparty file codes.
   - To delete an existing counterparty file code, drill down on it and follow the procedure in Deleting counterparty file codes.

4.2 Managing branch qualifiers

Branch qualifiers are required for EDIFACT formats. You can create one branch qualifier for each country in CMM.

4.2.1 Prerequisites

The following are prerequisites for managing branch qualifiers:

Category	Tasks
Static data	Ensure the following static data are available: • Countries. For more information, see the *WebSuite User Guide*.
Security	Ensure you have access to the following function: • FG-0004 Branch Qualifiers. For more information, see the *WebSuite System Administration Guide*.

4.2.2 Creating branch qualifiers

To create a branch qualifier:

1. Select **Admin - Static Data - Bank Interfacing - Branch Qualifiers**.
2. In the Branch Qualifiers List page, click **New Entry**.
3. In the Bank Branch Qualifier Maintenance page, create the branch qualifier.
4. Click **Save**.

4.2.3 Editing branch qualifiers

To edit a branch qualifier:

1. Select **Admin - Static Data - Bank Interfacing - Branch Qualifiers**.
2. In the Branch Qualifiers List page, drill down on the branch qualifier.
3. In the Bank Branch Qualifier Maintenance page, edit the branch qualifier.
4. Click **Save**.

4.2.4 Deleting branch qualifiers

To delete a branch qualifier:

1. Select **Admin - Static Data - Bank Interfacing - Branch Qualifiers**.
2. In the Branch Qualifiers List page, drill down on the branch qualifier.
3. In the Bank Branch Qualifier Maintenance page, Click **Delete**.
4. In the resulting dialog, click **OK**.

Chapter 5 Using the web services interface

CMM supports two types of web service:

Name	Use
Internal	Integration with the other modules of Wallstreet Suite.
External	Transferring of information between CMM and external applications.

The web services interface utilizes the second of these two types and allows you to connect CMM to an application external to CMM through an adaptor. The application then transforms, secures, or transports messages for CMM.

You can use applications connected to CMM through the web services interface for both import and export processes. To use such applications, you must install them in locations accessible to the CMM application server and create adaptors to connect them to CMM. Therefore, you require extensive knowledge of the applications as well as web services technology in general.

The web services interface utilizes the following technologies:

Name	Use
Web services [1]	Transfer messages between CMM and external applications
SOAP messages	Contain messages' contents
SSL over HTTP (HTTPS)	Secure messages

Table notes:

1. The web services utilized by the interface are intended to act as a transport protocol for importing data to and exporting data from CMM. The web services utilized by the interface are not compliant with Web Services Description Language (WSDL).

5.1 SOAP schemas

Three schemas define the structure of SOAP messages that can be transmitted between CMM and external applications through the web services interface:

- Request/response header (`requestresponseheader.xsd`)
- Request body (`requestbody.xsd`)
- Response body (`responsebody.xsd`).

For example SOAP messages that are compliant with these schemas, see *5.4 Example SOAP messages* on page 83.

5.1.1 Request/response header schema

This schema is used for constructing and consuming the header of both request and response SOAP messages:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--@author 2004-Nov-09 GWang Created-->
<xsd:schema
 targetNamespace="http://www.trema.com/externalinterface/XMLschema"
 elementFormDefault="unqualified"
 attributeFormDefault="unqualified"
 jaxb:version="1.0"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns="http://www.trema.com/externalinterface/XMLschema"
 xmlns:jaxb="http://java.sun.com/xml/ns/jaxb">
    <xsd:annotation>
       <xsd:documentation xml:lang="en">
          CMM request SOAP header to external interface clients
          Copyright 2004 trema.com. All rights reserved.
       </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="trema_web_service_header">
       <xsd:complexType>
          <xsd:sequence>
             <!--Contains the destination servlet URL-->
             <xsd:element name="end_point_url" type="xsd:anyURI"/>
             <xsd:element name="message_type" type="xsd:string"/>
             <xsd:element name="additional_properties"
            type="additionalPropertyType" minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
       </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="additionalPropertyType">
       <xsd:sequence>
          <xsd:element name="name" type="xsd:string"/>
          <xsd:element name="value" type="xsd:string"/>
       </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

You need to provide three mandatory input parameters when constructing the SOAP element of `trema_web_service_header` for the following elements:

- `end_point_url`: The web service endpoint. For the import process, this is a published web service URL from CMM.
- `message_type`: The type of the SOAP request. Valid values are `initiation` and `transaction`.
- `additional_properties`: Additional properties about the type of the SOAP request, such as the name-value pair `transaction_function=2` where `2` is the import/export type ID:

ID	Description	Import/export
2	AP File Import	I
6	AR File Import	I
8	Bank Account Import	I
10	Bank Messages Import	I
1	Bank Transaction Import	I
9	Counterparty Bank Import	I
7	Counterparty Import	I
27	Direct Debit Import	I
24	Entity Import	I

ID	Description	Import/export
38	Forecast Import	I

5.1.2 Request body schema

This schema is used for constructing and consuming the body of the request SOAP message:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- @author 2004-Nov-09 GWang Created -->
<xsd:schema
 targetNamespace="http://www.trema.com/externalinterface/XMLschema"
 elementFormDefault="unqualified"
 attributeFormDefault="unqualified"
 jaxb:version="1.0"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.trema.com/externalinterface/XMLschema"
 xmlns:jaxb="http://java.sun.com/xml/ns/jaxb">
    <xsd:annotation>
       <xsd:documentation xml:lang="en">
          CMM initiated request SOAP body to external interface client or external
         client initiated request SOAP body to CMM
          Copyright 2004 trema.com. All rights reserved.
       </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="trema_web_service_request">
       <xsd:complexType>
          <xsd:sequence>
             <xsd:element ref="format"/>
             <!--this is the destination bank's identification, which is mutually
            recognizable between CMM and the external interface client and is used
            to uniquely identify the destination bank-->
             <xsd:element name="interchange_receiver_id" type="xsd:string"/>
             <!--this is the id assigned to CMM as an identification, which is
            mutually recognizable between CMM and the external interface client-->
             <xsd:element name="interchange_sender_id" type="xsd:string"/>
             <!--this is the batch id cross referenced between CMM and the external
            client to identify the all the transaction messages in one process-->
             <xsd:element name="interchange_control_batch_id" type="xsd:string"/>
             <!--this is the transaction message sequence number cross referenced
            between CMM and the external client-->
             <!--This number is unique to each message within one interchange
            control batch id-->
             <xsd:element name="message_sequence_number"
            type="xsd:nonNegativeInteger"/>
            <xsd:element name="message_generation_datetime" type="xsd:dateTime"/>
          </xsd:sequence>
       </xsd:complexType>
    </xsd:element>
    <!--this is the format code mutually recognizable between CMM and external
   client-->
    <xsd:element name="format" type="xsd:string">
    </xsd:element>
    <!--put the actual transaction message in SOAP attachment part-->
</xsd:schema>
```

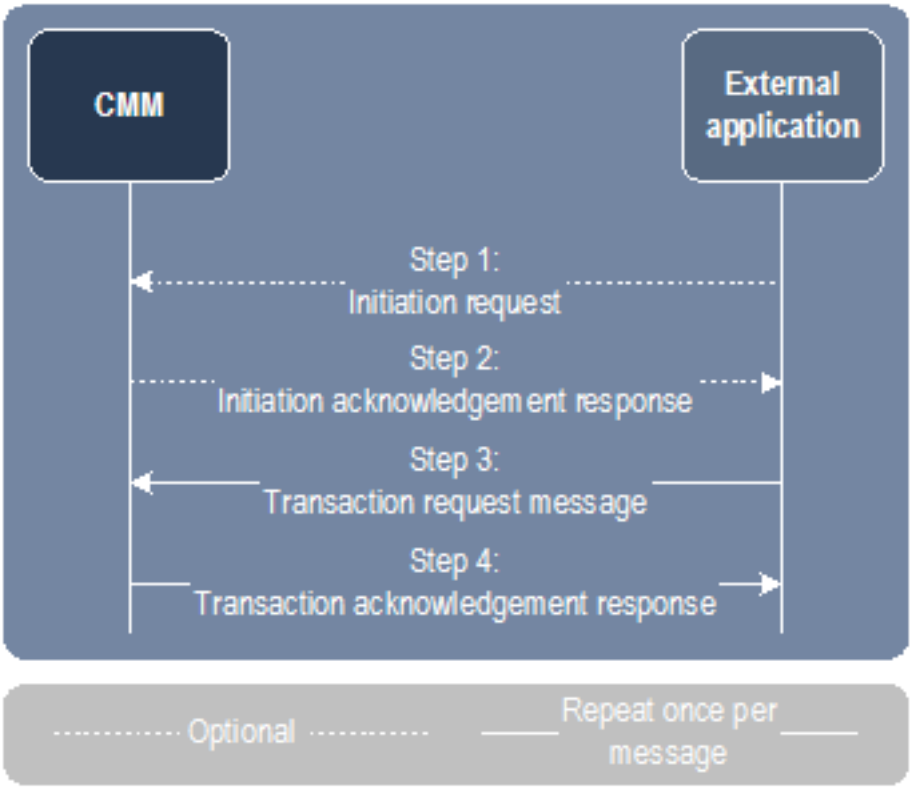
5.1.3 Response body schema

This schema is used for constructing and consuming the body of the response SOAP message:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- @author 2004-Nov-09 GWang Created -->
<xsd:schema
 targetNamespace="http://www.trema.com/externalinterface/XMLschema"
 elementFormDefault="unqualified"
 attributeFormDefault="unqualified"
 jaxb:version="1.0"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns="http://www.trema.com/externalinterface/XMLschema"
 xmlns:jaxb="http://java.sun.com/xml/ns/jaxb">
    <xsd:annotation>
       <xsd:documentation xml:lang="en">
          external interface client response SOAP body to CMM initiated request or
         CMM response SOAP body to external interface client initiated request
          Copyright 2004 trema.com. All rights reserved.
       </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="trema_web_service_response">
       <xsd:complexType>
          <xsd:sequence>
          <!--this is the batch id cross referenced between CMM and the external
         client to identify the all the transaction messages in one process-->
          <xsd:element name="interchange_control_batch_id" type="xsd:string"/>
          <!--this is the transaction message sequence number cross referenced
         between CMM and the external client-->
          <!--This number is unique to each message within one interchange control
         batch id-->
          <xsd:element name="message_sequence_number"
         type="xsd:nonNegativeInteger"/>
          <xsd:element name="message_generation_datetime" type="xsd:dateTime"/>
          <xsd:element name="request_success" type="xsd:boolean"/>
          <!-- this element presents only if the message type is initiation in the
         header and the request_success is true-->
          <xsd:element name="max_number_transactions_permessage"
         type="xsd:nonNegativeInteger" minOccurs ="0"/>
          <!-- this element presents only if the request_success is true-->
          <xsd:element name="rejection_reason" type="xsd:string" minOccurs="0"/>
          </xsd:sequence>
       </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

5.2 Importing messages using the web services interface

The following diagram and table present the steps involved when you import messages from an external application to CMM using the web services interface:



Number	Description	Subsequent action
1	The external application sends an initiation request SOAP message to CMM. The message contains basic information about the subsequent transaction messages. It consists of a header, defined by the request/response header schema, and a body, defined by the request body schema.	None
2	CMM sends an initiation acknowledgement response SOAP message to the external application. The message indicates if CMM is ready to receive the transaction messages and the maximum number of transactions per message it will allow. It consists of a header, defined by the request/response header schema, and a body, defined by the response body schema.	If the message is positive, the process continues to step 3. If the message is negative, the process stops.
3	The external application sends a transaction request SOAP message to CMM. CMM then validates the message and imports the message's transaction data. The message consists of a header, defined by the request/response header schema, and a body, defined by the request body schema. The body contains the transaction data as an attachment in a format that CMM can support.	None
4	CMM sends a transaction acknowledgement response SOAP message to the external application. The message consists of a header, defined by the request/response header schema, and a body, defined by the response body schema.	If the message is positive, the process returns to step 3. (The process repeats steps 3 and 4 until all messages have been imported.) If the message is negative, the process stops.

For examples of the messages sent between CMM and the external application, see *5.4 Example SOAP messages* on page 83.

5.2.1 Deploying the web service and creating the adaptor

CMM is deployed with a web service in the `ews` folder of the same web container as the application. For example, if CMM's URL is `http://treasury.acme-co.com:8080/cmm`, the web service's URL is `http://treasury.acme-co.com:8080/cmm/ews`.

You need to create an adaptor to do the following:

- Construct and consume SOAP messages (including attachments).

  The SOAP messages must be compliant with CMM's schemas.

- Call the Web service through its URL.

You can write the adaptor in a language of your choice as long as the adaptor can complete the above two tasks.

For demonstration purpose, Wallstreet has implemented an example import adaptor in Java. It is contained in the following zip file, which is included in every CMM release's package:

`external interfaces cmmext-client-cmm-[Release].zip`

Where `[Release]` is the release's version and build numbers (for example, `7.1.5.0-b0001`).

To use the example import adaptors, you need to extract the zip file from the package of the current CMM release your organization is using and run it in an application outside of CMM.

The example import adaptor's class is `com.trema.cmmext.service.CaOutboundMessageInitiatorDemoImpl`. To use this adaptor, you must create a request response communication protocol parameter and interchange (see *Chapter 3 Managing interchanges (and related data)* on page 41). You can then import messages.

5.2.2 Securing the web service

You can secure data at the following levels:

Name	Description
Transport	Authentication, which involves verifying the identity of a user, device, or other entity in a computer system, usually as a prerequisite to allowing access to resources in a system.
Message	Content encryption and digital signatures.

In CMM, the transport level is addressed through the web service, while the message level is addressed through the signer parameters assigned to interchanges. (For more information on interchanges, see *3.4 Managing interchanges* on page 56.)

To secure a web service for an import process, you need to complete the following steps:

1. Generate and configure SSL certificates.
2. Edit the `extwsaccessconfig.xml` file.

Note: The technology CMM uses for web services security is client-certificate authentication over HTTP/SSL, which needs to be enabled on the client side and authenticated on the server side.

5.2.2.1 Generating and configuring SSL certificates

The first step in securing a web service for an import process is to generate and configure SSL certificates.

For instructions on generating and configuring SSL certificates, see *Appendix D SSL certificate generation and configuration* on page 419.

5.2.2.2 Editing the extwsaccessconfig.xml file

The second step in securing a web service for an import process is to edit the `extwsaccessconfig.xml` file.

The `extwsaccessconfig.xml` file defines whether SSL over HTTPS is enabled or not and, if it is, the validation to be completed on the request URL and client certificate to ensure a request is from a trusted source. If a request passes the validation, CMM invokes the web service; otherwise, CMM filters out the request and returns a SOAP fault message.

Note: The `extwsaccessconfig.xml` file is for external web services only. There is a separate but nearly identical file for internal web services, `intwsaccessconfig.xml`.

The following is an example of the `extwsaccessconfig.xml` file:

```
◦
<strong_authentication>
    <mapping login_id="…" ssl_enable="true">
       <validate_request id="RequestURL" name="req_url" pattern="^https:.*"/>
          <certificate id="javax.servlet.request.X509Certificate"
         issuer_dn_pattern="(OU=C([a-zA-Z]*) A([a-zA-Z]*),)">
             <value id="SubjectDN" classname="alterna.appserver.security.cert.
            CaX509CertificateStringValueParser" pattern="=(\w*)@" />
             …
          </certificate>
          …
    </mapping>
    …
</strong_authentication>
```

As this example shows, the `extwsaccessconfig.xml` consists of several elements:

- `strong_authentication`: This is the root element of the document. It has a single child element that describes the mapping configuration, `mapping`. It has no attributes.
- `mapping`: This element is used to define the validation rules. A request is only valid if it passes the `validate_request` and `certificate` validation rules defined in this element.

  Attributes:

Name	Description	Default value (if any)
`ssl_enable`	The value of this attribute tells the web service filter whether to perform client request certificate validation or not. • If the value is `true`, the validation is enabled and requests fail if they do not pass the rules defined in this configuration file. • If the value is `false`, the validation is disabled and requests go through without validation against the rules defined in this configuration file. You can set this attribute to false in testing environments if you only want to ensure that messages follow through HTTP regardless of security. Wallstreet recommends that you set this attribute to true in production environments.	`true`

Child elements:

- `validate request`
- `certificate`.

- `validate_request`: This element defines the validation rule for attributes of the client request. One typical attribute you should validate is `RequestURL`. Other attributes that you may want to validate include `RemoteHost` and `RemoteAddr`.

  You can define the `validate_request` element multiple times within the parent element (`mapping`), provided each element has a unique value for the `id` attribute. A request is validated against each item defined in the `validate_request` element's attributes, and if it passes validation against all items, it is passed on to certificate validation.

  Attributes:

Name	Description	Default value (if any)
`id`	The ID to be valid for the request. Acceptable values: • `RequestURL` • `RemoteHost` • `RemoteAddr`.	
`name`	A name corresponding to the ID. There is no restriction on the name as long as it is meaningful to the user. If not specified, it is defaulted to id.	
`pattern`	The regular expression pattern used to validate the value from request for the ID specified. Wallstreet recommends that you follow the regular expression guidelines from Sun when constructing or consuming your own pattern and that you test it before going to production.	

- `certificate`: This element defines the validation rule for the client certificate provided in the request.

  In most cases you only need to define one of these elements in the parent element (`mapping`) since most likely you are dealing with only one issuer. However, you are allowed to define multiple instances of this element provided each element has a different value for the `issuer_dn_pattern` attribute. Certificates from requests are validated against each `certificate` rule (per `issuer_dn_pattern`). A request passes validation if it passes any of the `certificate` rules.

  Attribute:

Name	Description	Default value (if any)
`id`	The unique ID that identifies the type of certificate to be validated. The current implementation supports the following types: • `javax.servlet.request.X509Certificate`. This is the key attribute in the HTTP request in the secured connection.	[Fixed value]

Name	Description	Default value (if any)
issure_dn_pattern	The regular expression pattern used to validate the issuer of the certificate. Wallstreet recommends that you follow the regular expression guidelines from Sun when constructing or consuming your own pattern and that you test it before going to production. In addition, the pattern should be constructed so that it returns only a single group instead of multiple groups. If it is constructed to return multiple groups, the last one is returned.	

Child elements:

- value.

- value: This element defines the validation rule for the individual attributes of the certificate such as SubjectDN and SerialNumber.

  The individual value of the certificate as specified by the ID will be validated against the pattern defined. To pass the certificate validation, it has to pass each individual value validation. The certificate validation fails if any individual validation fails.

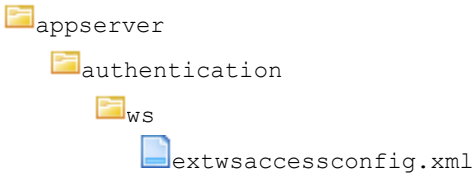
  Attributes:

Name	Description	Default value (if any)
id	The ID attribute that tells where the information is retrieved from the certificate. Acceptable values: • SubjectDN • IssuerDN • SigAlgName • SigAlgOID • SerialNumber. In most situations, you use SubjectDN as the value for this attribute.	
classname	The class name.	alterna.appserver.security.cert.CaX509Certificate StringValueParser
pattern	The regular expression pattern used to validate the attribute of the certificate as defined in the id attribute. Wallstreet recommends that you follow the regular expression guidelines from Sun when constructing or consuming your own pattern and that you test it before going to production. In addition, the pattern should be constructed so that it returns only a single group instead of multiple groups. If it is constructed to return multiple groups, the last one is returned.	

To edit the extwsaccessconfig.xml file:

1. Open the following configuration file:

    [Standard configuration file path]



For instructions on opening configuration files, see "Opening configuration files" on page32.

2. Edit the file.
3. Save and close the file.

5.2.3 Managing the interchange (and related data)

After deploying and securing a web service, and creating an adaptor for an import process, you can create the interchange (and related data) for that import process.

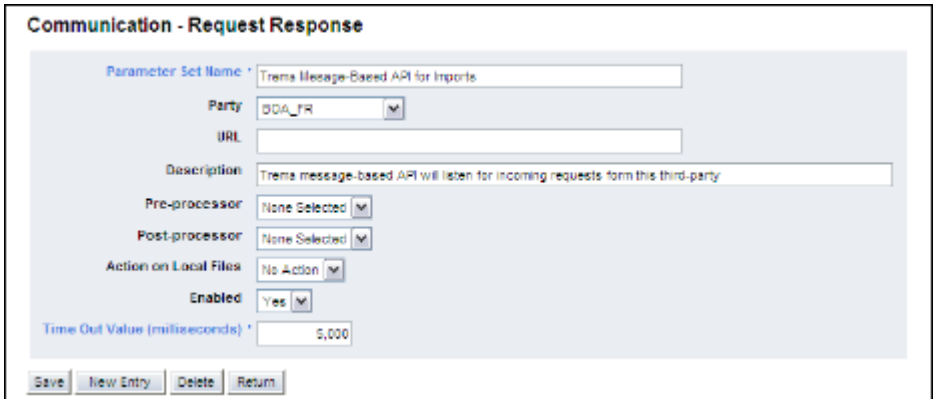
5.2.3.1 Managing the interchange (and related data)

To manage the interchange (and related data):

1. In the Communication Protocols function, ensure the Request Response communication protocol is enabled and manually executable.

   For instructions on enabling communication protocols, see *3.1.1 Managing communication protocols* on page 41.

2. In the Communication Protocol Parameters function, create a request response communication protocol parameter. The following is an example:



   Do not specify a value in the **URL** field, as the external application sends requests to the web service on the CMM side for import processes.

   Though **Time Out Value** is a required field, CMM does not use it for request response communication protocols. Therefore, the value you enter it is not particularly relevant.

   For instructions on creating communication protocol parameters, see *3.1.2 Managing communication protocol parameters* on page 42.

3. In the Interchanges function, create an interchange that references the request communication protocol parameter you created in step 2.

   For instructions on creating interchanges, see *3.4 Managing interchanges* on page 56.

5.2.4 Testing the import process

After deploying and securing a web service, creating an adaptor, and setting up the interchange and related components, you should test the import process to ensure it works properly.

This section includes two examples to show how to test an import process. The first example is for an accounts payable file import process, while the second is for a direct debit file import process.

5.2.4.1 Testing an accounts payable file import process that uses the web services interface

To test an accounts payable file import process that uses the web services interface:

1. Using the external application, generate an accounts payable file.
2. Using the adaptor, call the web service and transfer the accounts payable file to CMM using SOAP messages.

   You can use the example import adaptor provided with CMM for this step. For more information on the example import adaptor, see *5.2.1 Deploying the web service and creating the adaptor* on page 72.
3. Review the job log:
   - If the import was successful, its status in the job log will be "Successful" and the log file will reflect this:

     ```
     ◦
     07:00</message_generation_datetime>
     <request_success xmlns="">true</request_success>
     <max_number_transactions_permessage
     ◦
     ```

   - If the import was not successful because of a duplicate file, its status in the job log will be "Duplicate File" and the log file will reflect this:

     ```
     ◦
     07:00</message_generation_datetime>
     <request_success xmlns="">false</request_success>
     <max_number_transactions_permessage
     xmlns="">0</max_number_transactions_permessage>
     <rejection_reason xmlns="">Unexpected Exception: importFile( BOA_FR
     http://localhost/cmm/tws ) failed - message = Duplicate File - File has been
     imported.</rejection_reason>
     </trema_web_service_response>
     </soapenv:Body>
     </soapenv:Envelope>
     transaction request of 2 to http://localhost/cmm/tws is rejected due to:
     Unexpected Exception: importFile( BOA_FR http://localhost/cmm/tws ) failed -
     message = Duplicate File - File has been imported.
     ◦
     ```

   - If the import was not successful because CMM could not locate an appropriate interchange, the log file will reflect this:

     ```
     ◦
     07:00</message_generation_datetime>
     <request_success xmlns="">false</request_success>
     <max_number_transactions_permessage
     xmlns="">0</max_number_transactions_permessage>
     <rejection_reason xmlns="">Unexpected Exception: There is no request response
     interchange configured for the import type of: 2</rejection_reason>
     </trema_web_service_response>
     </soapenv:Body>
     </soapenv:Envelope>
     transaction request of 2 to http://localhost/cmm/tws is rejected due to:
     Unexpected Exception: There is no request response interchange configured for
     the import type of: 2
     ◦
     ```

   For instructions on reviewing the job log, see the *WebSuite System Administration Guide*.

5.2.4.2 Testing a direct debit file import process that uses the web services interface

To test a direct debit file import process that uses the web services interface:

1. Using the external application, generate a direct debit file.
2. Using the adaptor, call the web service and transfer the direct debit file to CMM using SOAP messages.

   You can use the example import adaptor provided with CMM for this step. For more information on the example import adaptor, see *5.2.1 Deploying the web service and creating the adaptor* on page 72.
3. Review the job log:
   - If the import was successful, its status in the job log will be "Successful" and the log file will reflect this:

     ```
     ◦
     07:00</message_generation_datetime>
     <request_success xmlns="">true</request_success>
     <max_number_transactions_permessage
     ◦
     ```

   - If the import was not successful because of a duplicate file, its status in the job log will be "Duplicate File" and the log file will reflect this:

     ```
     ◦
     07:00</message_generation_datetime>
     <request_success xmlns="">false</request_success>
     <max_number_transactions_permessage
     xmlns="">0</max_number_transactions_permessage>
     <rejection_reason xmlns="">Unexpected Exception: importFile( BOA_FR
     http://localhost/cmm/tws ) failed - message = Duplicate File - File has been
     imported.</rejection_reason>
     </trema_web_service_response>
     </soapenv:Body>
     </soapenv:Envelope>
     transaction request of 2 to http://localhost/cmm/tws is rejected due to:
     Unexpected Exception: importFile( BOA_FR http://localhost/cmm/tws ) failed -
     message = Duplicate File - File has been imported.
     ◦
     ```

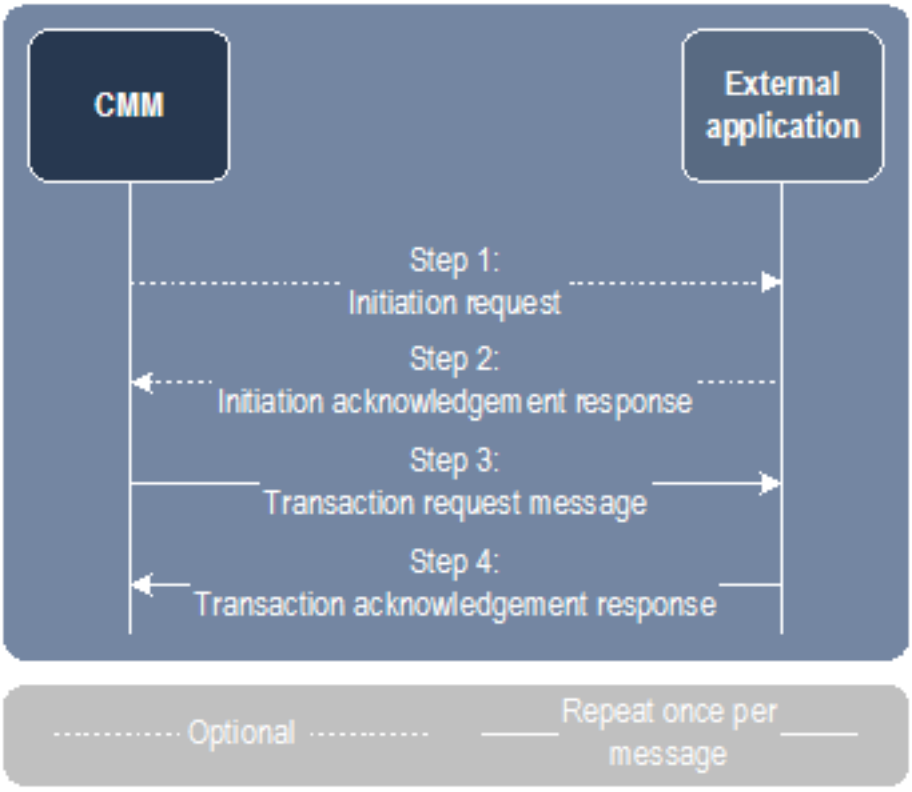
   - If the import was not successful because CMM could not locate an appropriate interchange, the log file will reflect this:

     ```
     ◦
     07:00</message_generation_datetime>
     <request_success xmlns="">false</request_success>
     <max_number_transactions_permessage
     xmlns="">0</max_number_transactions_permessage>
     <rejection_reason xmlns="">Unexpected Exception: There is no request response
     interchange configured for the import type of: 2</rejection_reason>
     </trema_web_service_response>
     </soapenv:Body>
     </soapenv:Envelope>
     transaction request of 2 to http://localhost/cmm/tws is rejected due to:
     Unexpected Exception: There is no request response interchange configured for
     the import type of: 2
     ◦
     ```

   For instructions on reviewing the job log, see the *WebSuite System Administration Guide*.

5.3 Exporting messages using the web services interface

The following diagram and table present the steps involved when you export messages from CMM to an external application using the web services interface:



Number	Description	Subsequent action
1	CMM sends an initiation request SOAP message to the external application. The message contains basic information about the subsequent transaction messages. It consists of a header, defined by the request/response header schema, and a body, defined by the request body schema.	None
2	The external application sends an initiation acknowledgement response SOAP message to CMM. The message indicates if the external application is ready to receive the transaction messages and the maximum number of transactions per message it will allow. It consists of a header, defined by the request/response header schema, and a body, defined by the response body schema.	If the message is positive, the process continues to step 3. If the message is negative, the process stops.
3	CMM sends a transaction request SOAP message to the external application. The message consists of a header, defined by the request/response header schema, and a body, defined by the request body schema. The body contains the transaction data as an attachment in a format that CMM can support.	None

Number	Description	Subsequent action
4	The external application sends a transaction acknowledgement response SOAP message to CMM. The message consists of a header, defined by the request/response header schema, and a body, defined by the response body schema.	If the message is positive, the process returns to step 3. (The process repeats steps 3 and 4 until all messages have been exported.) If the message is negative, the process stops.

For examples of the messages sent between CMM and the external application, see *5.4 Example SOAP messages* on page 83.

5.3.1 Deploying the web service and creating the adaptor

You need to deploy the web service on the external application's side. The specific location (URL) depends on the external application.

Wallstreet provides a demonstration web service (`com.trema.cmmext.service.CaExtRequestResponseServlet`). If you want to use demonstration web service, you need to deploy it in your J2EE web container by mapping the servlet to the `web.xml` file:

```
◦
<!-- External Interface Client Web Service -->
<servlet>
    <servlet-name>ExternalInterfaceWebService</servlet-name>
    <display-name>ExternalInterfaceWebService</display-name>

   <servlet-class>com.trema.cmmext.service.CaExtRequestResponseServlet</servlet-cla
   ss>
</servlet>
◦
<!-- External Interface Client Web Service Servelet Mapping-->
<servlet-mapping>
    <servlet-name>ExternalInterfaceWebService</servlet-name>
    <url-pattern>/ews/*</url-pattern>
    <load-on-startup>20</load-on-startup>
</servlet-mapping>
◦
```

In addition, you need to create an adaptor to do the following:

- Construct and consume SOAP messages (including attachments).

  The SOAP messages must be compliant with CMM's schemas.

- Call the web service through its URL.

You can write the adaptor in a language of your choice as long as the adaptor can complete the above two tasks.

For demonstration purpose, Wallstreet has implemented an example export adaptor in Java. It is contained in the following zip file, which is included in every CMM release's package:

`external interfaces cmmext-client-cmm-[Release].zip`

Where `[Release]` is the release's version and build numbers (for example, `7.1.5.0-b0001`).

To use the example export adaptors, you need to extract the zip file from the package of the current CMM release your organization is using and run it in an application outside of CMM.

The example export adaptor's class is `com.trema.cmmext.service.CaInboundSOAPServiceDemoImpl`. To use this adaptor, you must create a request response communication protocol parameter and interchange (see *Chapter 3 Managing interchanges (and related data)* on page 41). You can then export messages.

5.3.2 Securing the web service

Because the web service resides on the external application side for exports, how you secure the web service varies.

If you want to secure the web service on the external application side in a manner similar to how the web service on the CMM side is secured, Wallstreet provides a demonstration implementation of CaDemoSSLConnectionPropertyBuilder. You can set up your own implementation of CaDemoSSLConnectionPropertyBuilder and return it from CaRequestResposeServiceProvider.

If you choose to secure the web service on the external application side in a manner similar to how the web service on the CMM side is secured, you need to edit the `sslconnectionproperties.xml` file on the CMM side.

The `sslconnectionproperties.xml` file allows you configure the SSL certificate properties per target URL. If the certificate properties of your target URL are specified in this file, the SSL connection for your target URL is enabled whenever you make a connection to that URL and is removed once the connection is released.

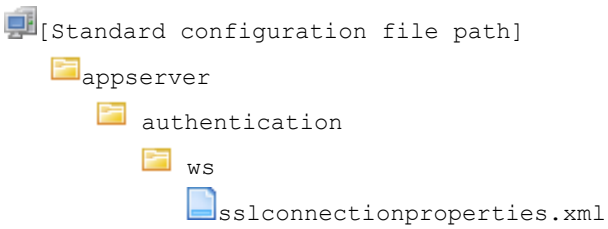
The following is an example of the `sslconnectionproperties.xml` file:

```
°
<ssl_connection_properties>
    <target_ws_url value="https://localhost:8143/cmm//jws/tes" description="the end
   point url of the target web service, this should be in https protocol">
       <!--properties of the key store-->
       <keystoreFile value="C:/security/alterna.p12" description="the full path of
      the key store file"/>
       <keystoreType value="PKCS12" description="this should be the Java supported
      type only (PKCS12,JKS)"/>
       <keystorePass value="trema.ca" description="this is the password of the key
      store"/>
       <!--properties of the trusted CA-->
       <truststoreFile value="C:/security/myCA.p12" description="the full path of
      the trust store file"/>
      <truststoreType value="PKCS12" description="this should be the Java supported
      type only (PKCS12,JKS)"/>
      <truststorePass value="trema.ca" description="this is the password of the
      trust store"/>
       <debug value="false" description="for debug purpose, set to true while you
      doing test"/>
    </target_ws_url>
    <!-- additional ssl connection properties for other target_ws_url can be added
   here -->
   °
</ssl_connection_properties>
```

5.3.2.1 Editing the sslconnectionproperties.xml file

To edit the `sslconnectionproperties.xml` file:

1. Open the following configuration file:



For instructions on opening configuration files, see "Opening configuration files" on page32.

2. Edit the file.
3. Save and close the file.

5.3.3 Managing the interchange (and related data)

After deploying and securing a web service, and creating an adaptor for an export process, you can configure the interchange (and related components) for that export process.

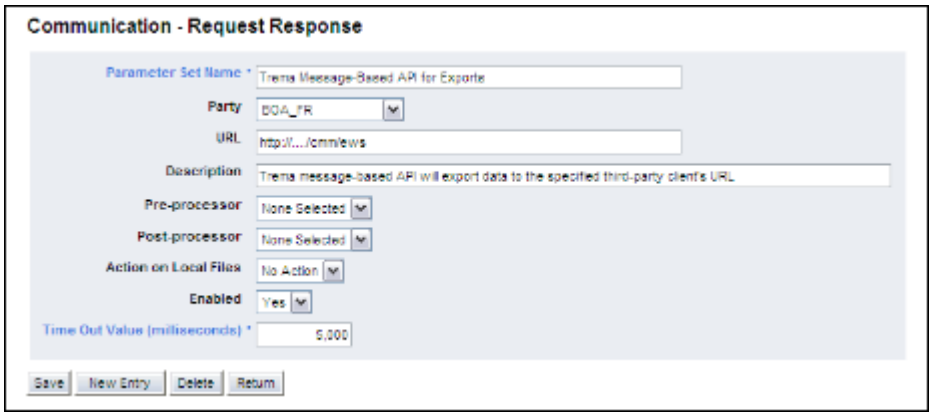
5.3.4 Managing the interchange (and related data)

To manage the interchange (and related data):

1. In the Communication Protocols function, ensure the Request Response communication protocol is enabled and manually executable.

   For instructions on enabling communication protocols, see *3.1.1 Managing communication protocols* on page 41.

2. In the Communication Protocol Parameters function, create a request response communication protocol parameter. The following is an example:



   You need to specify a value in the **URL** field. Specifically, you need to enter the URL of the web service on the external application side.

   Though **Time Out Value** is a required field, CMM does not use it for request response communication protocols. Therefore, the value you enter it is not particularly relevant.

   For instructions on creating communication protocol parameters, see *3.1.2 Managing communication protocol parameters* on page 42.

3. In the Interchanges function, create an interchange that references the request communication protocol parameter you created in step 2.

   For instructions on creating interchanges, see *3.4 Managing interchanges* on page 56.

5.3.5 Testing the export process

After deploying and securing a web service, creating an adaptor, and setting up the interchange and related components, you should test the export process to ensure it works properly.

This section includes two examples to show how to test an export process. The first example is for a payment file export process, while the second is for a direct debit file export process.

5.3.5.1 Testing a payment file export process that uses the web services interface

Test the export process by releasing the payments you previously imported and then checking the jog log. The payment files will be exported to the designated output folder.

For instructions on releasing payments and reviewing the jog log, see the *WebSuite User Guide* and the *WebSuite System Administration Guide*.

5.3.5.2 Testing a direct debit file export process that uses the web services interface

Test the export process by releasing the direct debits you previously imported and then checking the jog log. The direct debit files will be exported to the designated output folder.

For instructions on releasing direct debits and reviewing the jog log, see the *WebSuite User Guide* and the *WebSuite System Administration Guide*.

5.4 Example SOAP messages

This section presents example SOAP messages that are compliant with the schemas documented in *5.1 SOAP schemas* on page 67.

5.4.1 Initiation request message

The following is an example of an initiation request message:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
    <soapenv:Header>
       <tws_wrapper soapenv:mustUnderstand="0">
          <trema_web_service_header xmlns="http://www.trema.com/externalinterface/
         XMLschema">
             <end_point_url xmlns="">Http://localhost:1108/cmm</end_point_url>
             <message_type xmlns="">initiation</message_type>
             <additional_properties xmlns="">
                <name xmlns="">transaction_function</name>
                <value xmlns="">1</value>
             </additional_properties>
          </trema_web_service_header>
       </tws_wrapper>
    </soapenv:Header>
    <soapenv:Body>
       <trema_web_service_request xmlns="http://www.trema.com/externalinterface/
      XMLschema" xmlns:ns1="http://www.trema.com/externalinterface/XMLschema">
          <format xmlns="">FINSTA</format>
          <interchange_receiver_id xmlns=""/>
          <interchange_sender_id xmlns="">test request id</interchange_sender_id>
          <interchange_control_batch_id xmlns="">108</interchange_control_batch_id>
          <message_sequence_number xmlns="">1</message_sequence_number>
          <message_generation_datetime xmlns="">2005-01-06T14:36:04.671-07:00
         </message_generation_datetime>
          </trema_web_service_request>
    </soapenv:Body>
</soapenv:Envelope>
```

5.4.2 Positive initiation response message

The following is an example of a positive initiation response message:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header>
       <tws_wrapper soapenv:mustUnderstand="0">
          <trema_web_service_header xmlns="http://www.trema.com/externalinterface/
         XMLschema">
              <end_point_url xmlns="">Http://localhost:1108/cmm</end_point_url>
              <message_type xmlns="">initiation</message_type>
              <additional_properties xmlns="">
                 <name xmlns="">transaction_function</name>
                 <value xmlns="">1</value>
              </additional_properties>
           </trema_web_service_header>
       </tws_wrapper>
    </soapenv:Header>
    <soapenv:Body>
       <trema_web_service_response xmlns="http://www.trema.com/externalinterface/
      XMLschema" xmlns:ns1="http://www.trema.com/externalinterface/XMLschema">
          <interchange_control_batch_id xmlns="">108</interchange_control_batch_id>
          <message_sequence_number xmlns="">1</message_sequence_number>
          <message_generation_datetime xmlns="">2005-01-06T14:36:05.609-07:00
         </message_generation_datetime>
          <request_success xmlns="">true</request_success>
          <max_number_transactions_permessage xmlns="">5000
         </max_number_transactions_permessage>
       </trema_web_service_response>
    </soapenv:Body>
</soapenv:Envelope>
```

5.4.3 Negative initiation response message

The following is an example of a negative initiation response message:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
    <soapenv:Header>
       <tws_wrapper soapenv:mustUnderstand="0">
          <trema_web_service_header xmlns="http://www.trema.com/externalinterface/
         XMLschema">
              <end_point_url xmlns="">Http://localhost:1108/cmm</end_point_url>
              <message_type xmlns="">initiation</message_type>
              <additional_properties xmlns="">
                 <name xmlns="">transaction_function</name>
                 <value xmlns="">1</value>
              </additional_properties>
           </trema_web_service_header>
       </tws_wrapper>
    </soapenv:Header>
    <soapenv:Body>
       <trema_web_service_response xmlns="http://www.trema.com/externalinterface/
      XMLschema" xmlns:ns1="http://www.trema.com/externalinterface/XMLschema">
          <interchange_control_batch_id xmlns="">108</interchange_control_batch_id>
          <message_sequence_number xmlns="">1</message_sequence_number>
          <message_generation_datetime xmlns="">2005-01-06T14:36:05.609-07:00
         </message_generation_datetime>
          <request_success xmlns="">false</request_success>
          <rejection_reason xmlns="">Missing segment of BUS</rejection_reason>
       </trema_web_service_response>
    </soapenv:Body>
```

```
</soapenv:Envelope>
```

5.4.4 Transaction request message

The following is an example of a transaction request message:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
    <soapenv:Header>
       <tws_wrapper soapenv:mustUnderstand="0">
          <trema_web_service_header xmlns="http://www.trema.com/externalinterface/
         XMLschema">
             <end_point_url xmlns="">Http://localhost:1108/cmm</end_point_url>
             <message_type xmlns="">transaction</message_type>
             <additional_properties xmlns="">
                <name xmlns="">transaction_function</name>
                <value xmlns="">5</value>
             </additional_properties>
          </trema_web_service_header>
       </tws_wrapper>
    </soapenv:Header>
    <soapenv:Body>
       <trema_web_service_request xmlns="http://www.trema.com/externalinterface/
      XMLschema" xmlns:ns1="http://www.trema.com/externalinterface/XMLschema">
          <format xmlns="">CRG_PAYMUL</format>
          <interchange_receiver_id xmlns=""/>
          <interchange_sender_id xmlns="">test request id</interchange_sender_id>
          <interchange_control_batch_id xmlns="">108</interchange_control_batch_id>
          <message_sequence_number xmlns="">1</message_sequence_number>
          <message_generation_datetime xmlns="">2005-01-06T14:36:05.796-07:00
        </message_generation_datetime>
       </trema_web_service_request>
    </soapenv:Body>
</soapenv:Envelope>
------=_Part_0_25222452.1105047365843
Content-Type: text/plain
Content-Transfer-Encoding: binary
Content-Id: <trema_message_content_id>
UNB+UNOA:1+5563295673:ZZ+5013546080448+030708:1341+6'UNH+2021+PAYMUL:D:96A:UN:...----
--=_Part_0_25222452.1105047365843--
```

5.4.5 Positive transaction response message

The following is an example of a positive transaction response message:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header>
       <tws_wrapper soapenv:mustUnderstand="0">
          <trema_web_service_header xmlns="http://www.trema.com/externalinterface/
         XMLschema">
             <end_point_url xmlns="">Http://localhost:1108/cmm</end_point_url>
             <message_type xmlns="">transaction</message_type>
             <additional_properties xmlns="">
                <name xmlns="">transaction_function</name>
                <value xmlns="">5</value>
             </additional_properties>
          </trema_web_service_header>
       </tws_wrapper>
```

```
    </soapenv:Header>
    <soapenv:Body>
       <trema_web_service_response xmlns="http://www.trema.com/externalinterface/
      XMLschema" xmlns:ns1="http://www.trema.com/externalinterface/XMLschema">
          <interchange_control_batch_id xmlns="">108</interchange_control_batch_id>
          <message_sequence_number xmlns="">1</message_sequence_number>
          <message_generation_datetime xmlns="">2005-01-06T14:36:06.515-07:00
         </message_generation_datetime>
          <request_success xmlns="">true</request_success>
       </trema_web_service_response>
    </soapenv:Body>
</soapenv:Envelope>
```

5.4.6 Negative transaction response message

The following is an example of a negative transaction response message:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
    <soapenv:Header>
       <tws_wrapper soapenv:mustUnderstand="0">
          <trema_web_service_header xmlns="http://www.trema.com/externalinterface/
         XMLschema">
             <end_point_url xmlns="">Http://localhost:1108/cmm</end_point_url>
             <message_type xmlns="">transaction</message_type>
             <additional_properties xmlns="">
                <name xmlns="">transaction_function</name>
                <value xmlns="">5</value>
             </additional_properties>
          </trema_web_service_header>
       </tws_wrapper>
    </soapenv:Header>
    <soapenv:Body>
       <trema_web_service_response xmlns="http://www.trema.com/externalinterface/
      XMLschema" xmlns:ns1="http://www.trema.com/externalinterface/XMLschema">
          <interchange_control_batch_id xmlns="">108</interchange_control_batch_id>
          <message_sequence_number xmlns="">1</message_sequence_number>
          <message_generation_datetime xmlns="">2005-01-06T14:36:06.515-07:00
         </message_generation_datetime>
          <request_success xmlns="">false</request_success>
          <max_number_transactions_permessage xmlns="">0
         </max_number_transactions_permessage>
          <rejection_reason xmlns="">Missing segment of BUS</rejection_reason>
       </trema_web_service_response>
    </soapenv:Body>
</soapenv:Envelope>
```

5.5 Example SOAP message construction and consumption

This section documents the steps required to construct and consume SOAP messages that can be transported with the CMM web services interface using the SAAJ and JAXB APIs.

Note: This is not the only way that you can create and manipulate SOAP messages in Java—it is an example of one way to do it. If you are more comfortable with other technologies, you can use them instead.

SAAJ (SOAP with Attachments API for Java) contains APIs for creating and populating SOAP messages that may or may not contain attachments. It also contains APIs for sending point-to-point, non-provider-based, request-and-response SOAP messages.

JAXB (Java Architecture for XML Binding) provides a convenient way to bind an XML schema to a representation in Java code. This makes it easy for you to incorporate XML data and processing functions in applications based on Java technology without having to know much about XML itself.

CMM's demo web services implementations use JAXB API to bind an XML schema to a representation in Java code and SAAJ API to attach a transactional document to a SOAP message before sending it to the web service endpoint. To transmit a SOAP message to a web service endpoint, the implementations use HttpClient API to handle both HTTP and HTTPS protocols, minimizing side effects on the client environment.

Note: This section does not include steps on how to transport SOAP messages via HttpClient.

5.5.1 Binding XML schemas to Java code

When communicating with CMM web services, a SOAP message needs to be compliant with CMM's web service schemas for both request and response SOAP messages. These schema files are provided to you in the `cmmext-client-[Release].jar` file.

Because CMM uses JAXB API to bind an XML schema to Java code, the module also provide a pre-compiled version of these Java classes in the `cmmext-client-[Release].jar` file. The sources are located in the `cmmext-client-[Release]-src.jar` file. These Java classes are generated by an XJC tool provided by Sun. To learn how to use this tool, visit the Sun website.

The following are additional libraries required for running the XJC tool:

- `jwsdp-1_4_jaxb-jaxb-xjc-1_4-TEST-DEPENDENCY.jar`
- `jwsdp-1_4_jaxb-xercesImpl-1_4-TEST-DEPENDENCY.jar`.

You can copy these Java archive files from the CMM application folder or the `cmm.war` file.

When running the XJC tool, you need to provide three input parameters:

Name	Description
`-xmlschema`	Physical path of the schema file.
`-p`	Fully qualified package name in which you want to generate the classes. If the schema file is `requestresponseheader.xsd`, the desired package name would be `com.trema.cmmext.message.bindings.requestresponseheader`.
`-d`	Physical path of the folder in which you want to output Java classes.

The following is a sample output log:

```
parsing a schema...
compiling a schema...
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\UnmarshallingEv
entHandler.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\ErrorHandlerAda
ptor.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\NamespaceContex
t2.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\ContentHandlerA
daptor.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\UnmarshallerImp
l.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\XMLSerializable
.java
```

com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\ValidatorImpl.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\NamespaceContextImpl.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\GrammarInfo.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\ValidationContext.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\PrefixCallback.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\ValidatableObject.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\Util.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\UnmarshallingEventHandlerAdaptor.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\MSVValidator.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\UnmarshallableObject.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\SAXMarshaller.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\InterningUnmarshallerHandler.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\GrammarInfoImpl.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\GrammarInfoFacade.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\MarshallerImpl.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\Discarder.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\XMLSerializer.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\UnmarshallingContext.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\SAXUnmarshallerHandlerImpl.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\
AbstractUnmarshallingEventHandlerImpl.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\ValidatingUnmarshaller.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\DefaultJAXBContextImpl.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\runtime\SAXUnmarshallerHandler.java
com\trema\cmmext\message\bindings\requestresponseheader\AdditionalPropertyType.java
com\trema\cmmext\message\bindings\requestresponseheader\ObjectFactory.java
com\trema\cmmext\message\bindings\requestresponseheader\TremaWebServiceHeader.java
com\trema\cmmext\message\bindings\requestresponseheader\TremaWebServiceHeaderType.java
com\trema\cmmext\message\bindings\requestresponseheader\bgm.ser
com\trema\cmmext\message\bindings\requestresponseheader\jaxb.properties
com\trema\cmmext\message\bindings\requestresponseheader\impl\AdditionalPropertyTypeImpl.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\JAXBVersion.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\TremaWebServiceHeaderImpl.java
com\trema\cmmext\message\bindings\requestresponseheader\impl\TremaWebServiceHeaderTypeImpl.java

Wallstreet has generated Java classes for the following schema files:

- Request/response header
- Request SOAP body
- Response SOAP body.

For information on these schema files, see *5.1 SOAP schemas* on page 67.

5.5.2 Constructing and consuming SOAP messages

A SOAP message is made up of a SOAP envelope and zero or more attachments. The SOAP envelope is then made up of a SOAP header and a SOAP body. SOAP attachments allow the SOAP message to contain not only XML data but also non-XML data such as a JPEG files. SOAP attachments use the MIME multipart as a container for these non-XML data.

The SAAJ API provides the `SOAPMessage` class to represent a SOAP message, the `SOAPPart` class to represent the SOAP part, the `SOAPEnvelope` interface to represent the SOAP envelope, and so on.

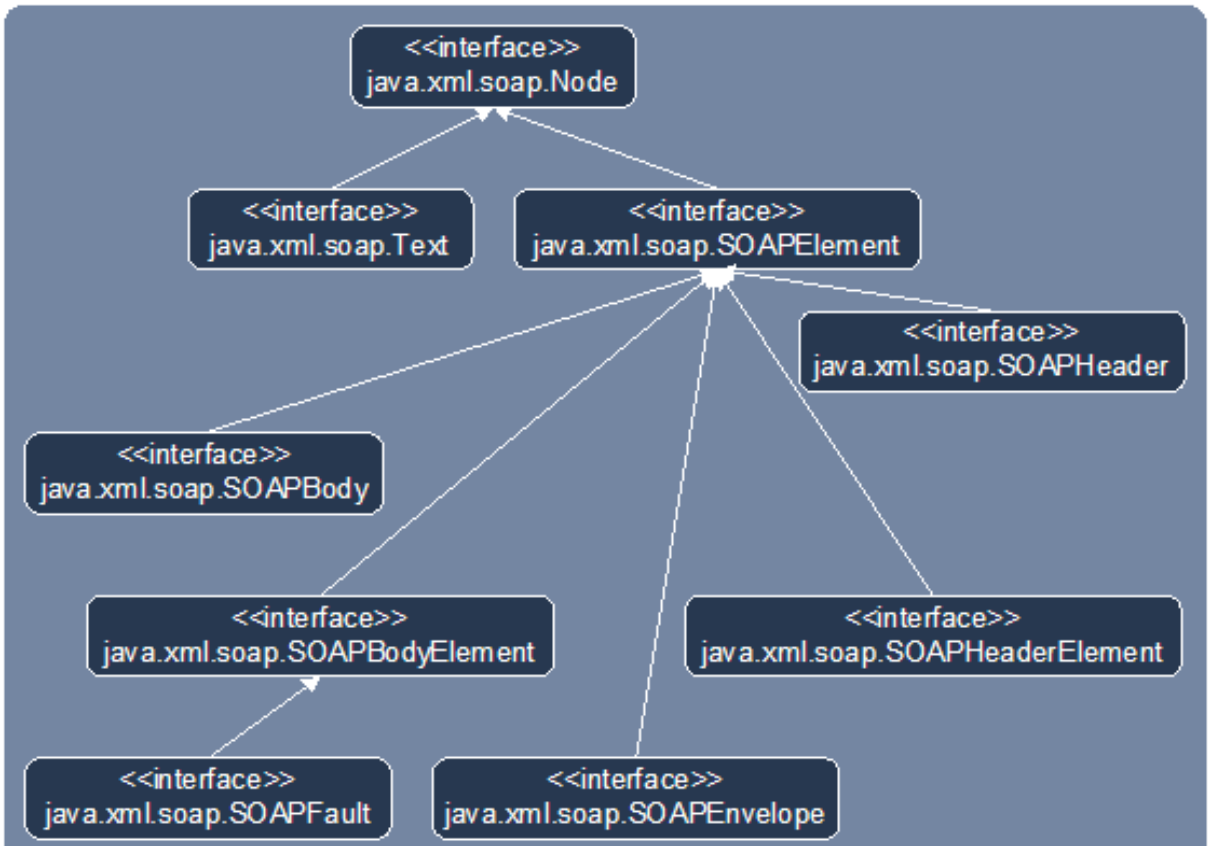
When you create a new `SOAPMessage` object, it automatically has the parts that are required to be in a SOAP message. In other words, a new `SOAPMessage` object has a `SOAPPart` object that contains a `SOAPEnvelope` object. The `SOAPEnvelope` object, in turn, automatically contains an empty `SOAPHeader` object followed by an empty `SOAPBody` object. If you do not need the `SOAPHeader` object, as it is optional, you can delete it. The rationale for having it automatically included is that more often than not you will need it, so it is more convenient to have it provided. The `SOAPHeader` object may contain one or more headers with information about the sending and receiving parties. The `SOAPBody` object, which always follows the `SOAPHeader` object if there is one, provides a simple way to send information intended for the ultimate recipient. For example, if there is a `SOAPFault` object, it must be in the `SOAPBody` object.

A SOAP message may include one or more attachment parts in addition to the SOAP part. The SOAP part may contain only XML content. As a result, if any of the content of a message is not in XML format, it must occur in an attachment part. For example, if you want your message to contain a binary file, your message must have an attachment part for it. An attachment part can contain any kind of content, so it can contain data in XML format as well.

The SAAJ API provides the `AttachmentPart` class to represent the attachment part of a SOAP message. A `SOAPMessage` object automatically has a `SOAPPart` object and its required subelements, but because `AttachmentPart` objects are optional, you have to create and add them yourself.

SAAJ API belongs to the `javax.xml.soap.*` package. `SOAPConnection` provides request/response SOAP message exchange. `SOAPMessage` creates and populates SOAP messages (consisting of `SOAPPart` and `AttachmentPart`).

The following diagram presents the relationships across these different components:



Note the following assumptions:

- The `getMsgUtil()` method returns an instance of `CaExtMessageUtil`, a utility class that is responsible for constructing or consuming a SOAP message through SAAJ and DOM API.
- The `getJAXBUtil()` method returns an instance of `CaExtJAXBUtil`, a utility class that is responsible for creating the appropriate DOM document through JAXB API. This includes the methods to marshall a Java object into a DOM document and unmarshall a DOM document or element into an appropriate Java object.
- The `getTransportService()` method returns an instance of an `IaSOAPRequestResponseService` interface. For the import process, `CaExtSOAPTransportService` would be the instance. For the export process, `CaInboundSOAPServiceDemoImpl` would be the instance.

5.5.2.1 Constructing and consuming SOAP messages for the import process

To construct and consume SOAP messages for the import process:

1. Create an initiation request SOAP message to CMM:

   ```
   SOAPMessage initiation_request = createRequest("initiation");
   ```

   Although this step is optional, it is highly recommended to send the initiation request to CMM to ensure CMM is ready to receive the transactional messages. This is a simple SOAP message that tests the connection to CMM web services.

   As the request SOAP messages for both initiation and transaction processes need to be created, it would be a good idea to factor this out in a separate method:

   ```
   private SOAPMessage createRequest(String msg_type) throws Exception
   {
       SOAPMessage req= getMsgUtil().createEmptySOAPMessage();
   ```

```
    populateHeader(req, msg_type);
    populateBody(req, msg_type);
    return req;
}
```

`CaExtMessageUtil` provides an API to create an empty SOAP message from a default implementation of a `MessageFactory` class:

```
public SOAPMessage createEmptySOAPMessage() throws Exception
{
    MessageFactory f = MessageFactory.newInstance();
    return f.createMessage();
}
```

A `SOAPMessage` object is required to have certain elements, and as stated previously, the SAAJ API simplifies things for you by returning a new `SOAPMessage` object that already contains these elements. So the message, which was created in the preceding line of code, automatically has the following:

- A `SOAPPart` object that contains
  - A `SOAPEnvelope` object that contains
    - An empty `SOAPHeader` object
    - An empty `SOAPBody` object.

For CMM to understand this SOAP message, the `SOAPHeader` and `SOAPBody` objects need to be populated based on the XML schema files provided previously:

```
private void populateHeader(SOAPMessage req, String msg_type)throws Exception
{
    String url = getCMMWebServiceURL();
    String txn_funct = getTransactionFunction();
    // create TREMA SOAPHeader using the API in JAXB Utility
    TremaWebServiceHeader trema_header = getJAXBUtil().createTremaHeader(url,
   msg_type,txn_funct);
    String class_path = getClassPath(TremaWebServiceHeader.class);
    // convert a Java object into a TREMA Header XML document using the API in
   JAXB Utility Document
   trema_header_doc = getJAXBUtil().marshall(trema_header,class_path);
    // add TREMA Header XML document to the SOAP message using the API in
   CaExtMessageUtil
    getMsgUtil().addTremaHeader(req,trema_header_doc);
}
```

Similarly, for the `SOAPBody` element:

```
private void populateBody(SOAPMessage req, String msg_type) throws Exception
{
    //body may be populated differently based on the msg_type
    // create an empty TREMA Request Body using the API in JAXB Utility
    TremaWebServiceRequest req_body = getJAXBUtil().createTremaRequestBody();
    // populate the request body
    req_body.setFormat(getFormat());
    req_body.setInterchangeSenderId(getInterchangeSenderID());
    req_body.setInterchangeReceiverId(getInterchangeReceiverID());
    req_body.setInterchangeControlBatchId(getInterchangeBatchID());
    req_body.setMessageSequenceNumber(getMessageSequenceNumber());
    req_body.setMessageGenerationDatetime(getMessageGenerationTime());
    String class_path = getClassPath(TremaWebServiceRequest.class);
    // convert TREMA Request Boday to a XML document using the API in JAXB Utility
   Document
   req_body_doc = getJAXBUtil().marshall(req_body,class_path);
    // add TREMA Request Body XML document into a SOAP message using the API in
   Msg Util
```

```
    getMsgUtil().addTremaBody(req,req_body_doc);
}
```

To print the SOAP message to the standard output, you can use the following code fragment:

```
getMsgUtil().printMessage(initiation_request);
```

You first need enable the debug mode by calling `getMsgUtil().setDebug(true).`

The result of the SOAP message that would be sent to CMM should be similar to this:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001
XMLSchema-instance">
    <soapenv:Header>
       <tws_wrapper soapenv:mustUnderstand="0">
          <tws:trema_web_service_header
         xmlns="http://www.trema.com/externalinterface/ XMLschema"
         xmlns:tws="http://www.trema.com/externalinterface/XMLschema">
             <end_point_url
            xmlns="">http://localhost:1234/cmm/ews/</end_point_url>
             <message_type xmlns="">initiation</message_type>
             <additional_properties xmlns="">
                <name>transaction_function</name>
                <value>2</value>
             </additional_properties>
          </tws:trema_web_service_header>
       </tws_wrapper>
    </soapenv:Header>
    <soapenv:Body>
       <ns1:trema_web_service_request
      xmlns="http://www.trema.com/externalinterface/ XMLschema"
      xmlns:ns1="http://www.trema.com/externalinterface/XMLschema">
          <ns1:format>PAYEXT97</ns1:format>
          <interchange_receiver_id xmlns="">test receiver
         id</interchange_receiver_id>
          <interchange_sender_id xmlns="">test sender id</interchange_sender_id>
          <interchange_control_batch_id
         xmlns="">batch-1</interchange_control_batch_id>
          <message_sequence_number xmlns="">0</message_sequence_number>
          <message_generation_datetime xmlns="">2006-04-04T11:20:30.296-06:00
         </message_generation_datetime>
       </ns1:trema_web_service_request>
    </soapenv:Body>
</soapenv:Envelope>
```

2. Transmit an initiation SOAP message to CMM:

```
SOAPMessage initiation_response =
getTransportService().onMessage(initiation_request);
```

When the initiation SOAP message is fully populated and ready to send, use the API in `IaSOAPRequestResponseService` to send a SOAP message to CMM. It can be sent through a secured or non-secured channel.

3. Receive a response SOAP message from CMM.

When CMM has processed the request, it sends back a response SOAP message to indicate the status of the process. This could be a FAULT SOAP message when CMM has encountered a fatal error during processing. The reason of the fault message is stored in the fault string element:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
    <soapenv:Body>
       <soapenv:Fault>
          <faultcode>soapenv:Server</faultcode>
```

```
          <faultstring>Failed on security validation, check your request security
         setting, then discuss it with your web service provider</faultstring>
          <faultactor>urn:trema-webserivce</faultactor>
          <detail/>
       </soapenv:Fault>
    </soapenv:Body>
</soapenv:Envelope>
```

When a response SOAP message is returned, use the API in `CaExtMessageUtil` to check whether it is a FAULT SOAP message:

```
if(getMsgUtil().hasSOAPFault(initiation_response))
{
    String ex_msg = "Initiation Response is SOAPFault: ";
    // you can custom your error handling here
    throw new Exception(ex_msg +
   getMsgUtil().getSOAPFaultDetails(initiation_response));
}
```

When CMM has processed the request without encountering any fatal error, it will send back a response SOAP message that would be similar to this:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
    <soapenv:Header>
       <tws_wrapper soapenv:mustUnderstand="0">
          <trema_web_service_header
         xmlns="http://www.trema.com/externalinterface/ XMLschema"
         xmlns:tws="http://www.trema.com/externalinterface/XMLschema">
             <end_point_url
            xmlns="">http://localhost:1234/cmm/ews/</end_point_url>
             <message_type xmlns="">initiation</message_type>
             <additional_properties xmlns="">
                <name>transaction_function</name>
                <value>2</value>
             </additional_properties>
          </trema_web_service_header>
       </tws_wrapper>
    </soapenv:Header>
    <soapenv:Body>
       <trema_web_service_response
      xmlns="http://www.trema.com/externalinterface/ XMLschema"
      xmlns:ns1="http://www.trema.com/externalinterface/XMLschema">
          <interchange_control_batch_id
         xmlns="">batch-1</interchange_control_batch_id>
          <message_sequence_number xmlns="">0</message_sequence_number>
          <message_generation_datetime xmlns="">2006-04-05T09:58:41.564-06:00
         </message_generation_datetime>
         <request_success xmlns="">true</request_success>
          <max_number_transactions_permessage xmlns="">1000
         </max_number_transactions_permessage>
       </trema_web_service_response>
    </soapenv:Body>
</soapenv:Envelope>
```

Note:

- The request and response SOAP messages share the same `SOAPHeader` element.
- The status of the process can be located in the `request_success` element having a boolean value of either true or false.
- When the status of the process is false, the rejection reason will be populated in the `rejection_reason` element. This error can also be reviewed in CMM using the job log.
- The value of the `max_number_transactions_permessage` element is not applicable in the import process as you can send CMM as many transactions per message as you wish. This element is more applicable for the export process when you can only process a limited number of transactions at a time. This way you can specify how many transactions per message you would like to receive from CMM.
- Currently, CMM does not provide details of the errors at the transaction level in the response SOAP message. This will be addressed in a future release.

You can use the APIs in JAXB utility and SOAP message utility to extract the values in the `SOAPHeader` and `SOAPBody` elements:

```
TremaWebServiceHeader ini_trema_header = getTremaHeader(initiation_response);
TremaWebServiceResponse ini_trema_response_body =
getTremaResponseBody(initiation_response);
if(ini_trema_response_body.isRequestSuccess())
 {
    // you can start sending transaction data to CMM
    processTransactions(ini_trema_header,ini_trema_response_body);
}
else
{
    // when there was a failure, you need to handle the error accordingly
    handleNegativeResponse(ini_trema_header,ini_trema_response_body);
}
private TremaWebServiceHeader getTremaHeader(SOAPMessage response)throws
Exception
{
    Element e_trema_header = getMsgUtil().getTremaHeader(response);
    String class_path = getClassPath(TremaWebServiceHeader.class);
    return (TremaWebServiceHeader)
   getJAXBUtil().unmarshall(e_trema_header,class_path);
}
private TremaWebServiceResponse getTremaResponseBody(SOAPMessage response) throws
Exception
{
    Element e_body = getMsgUtil().getTremaResponseBody(response);
    String class_path = getClassPath(TremaWebServiceResponse.class);
    return (TremaWebServiceResponse) getJAXBUtil().unmarshall(e_body,class_path);
}
```

4. Create a transaction SOAP request to CMM:

```
SOAPMessage request = createRequest("transaction");
```

Similar to the initiation request, you can use the same `createRequest()` method to create a default SOAP message to CMM.

5. Attach a transactional data stream to the request SOAP message:

```
CaSOAPMessageAttachUtil.attachToMessage(request, transaction_msg_content,
"ISO-8859-1");
```

Once the default request SOAP message has been created, you can attach the transaction data stream to the SOAP message using the API in `CaSOAPMessageAttachUtil`.

Note:

- To preserve the data integrity of the attachment message, you should set the character set of this attachment to ISO-8859-1. This way the I/O operations treat the input stream as a binary stream and not as a character stream. The encrypted or Unicode data stream is not be corrupted during the I/O processing.
- You should only attach a single transactional data stream to the SOAP message, which may contain more than one transaction's data.

Below is a sample implementation of the `attachToMessage()` method in `CaSOAPMessageAttachUtil`:

```
public static void attachToMessage(SOAPMessage soap_msg, Object
transaction_msg_content, String charset) throws Exception
{
     // to preserve the data sending the attachment as binary
     AttachmentPart attachment = soap_msg.createAttachmentPart();
     if ( transaction_msg_content instanceof InputStream )
    {
         String content_type = "application/octet-stream";
         attachment = soap_msg.createAttachmentPart(transaction_msg_content,
        content_type);
         attachment.setMimeHeader("Content-Type", content_type + "; charset=" +
        charset);
     }
     else if ( transaction_msg_content instanceof File )
    {
         File file = (File)transaction_msg_content;
         DataHandler dh = new DataHandler(new FileDataSource(file));
         attachment = soap_msg.createAttachmentPart(dh);
         String content_type = "application/octet-stream";
         attachment.setMimeHeader("Content-Type", content_type + "; charset=" +
        charset);
     }
     else
    {
         String content_type = "application/octet-stream";
         String str = (String)transaction_msg_content;
         InputStream is = new ByteArrayInputStream(str.getBytes(charset));
         attachment = soap_msg.createAttachmentPart(is, content_type);
         attachment.setMimeHeader("Content-Type", content_type + "; charset=" +
        charset);
     }
     attachment.setContentId(TRANACTION_ATTACHMENT_CONTENT);
     soap_msg.addAttachmentPart(attachment);
}
```

The result of the transaction request SOAP message should look similar to this:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
     <soapenv:Header>
        <tws_wrapper soapenv:mustUnderstand="0">
           <tws:trema_web_service_header
          xmlns="http://www.trema.com/externalinterface/ XMLschema"
          xmlns:tws="http://www.trema.com/externalinterface/XMLschema">
              <end_point_url
             xmlns="">http://localhost:1234/cmm/ews/</end_point_url>
              <message_type xmlns="">transaction</message_type>
              <additional_properties xmlns="">
                 <name>transaction_function</name>
                 <value>2</value>
```

```
            </additional_properties>
         </tws:trema_web_service_header>
      </tws_wrapper>
   </soapenv:Header>
   <soapenv:Body>
      <ns1:trema_web_service_request
     xmlns="http://www.trema.com/externalinterface/ XMLschema"
     xmlns:ns1="http://www.trema.com/externalinterface/XMLschema">
         <ns1:format>PAYEXT97</ns1:format>
         <interchange_receiver_id xmlns="">test receiver
        id</interchange_receiver_id>
         <interchange_sender_id xmlns="">test sender id</interchange_sender_id>
         <interchange_control_batch_id
        xmlns="">batch-1</interchange_control_batch_id>
         <message_sequence_number xmlns="">1</message_sequence_number>
         <message_generation_datetime xmlns="">2006-04-05T09:58:41.939-06:00
        </message_generation_datetime>
      </ns1:trema_web_service_request>
   </soapenv:Body>
</soapenv:Envelope>
---GgtA4s1Zac7_w7Y_ZQnQ_D8T_g4okiNkJ Content-Disposition: form-data;
name="transaction_message_attachment"; filename="attachment" Content-Type:
application/octet-stream; charset=ISO-8859-1 Content-Transfer-Encoding: binary
HEADER ROUTING/FUH_TEST_ALL
°
---GgtA4s1Zac7_w7Y_ZQnQ_D8T_g4okiNkJ--
```

6. Transmit a request SOAP message with attachment to CMM:

```
SOAPMessage transaction_response = getTransportService().onMessage(request);
```

Similarly, use the API in `IaSOAPRequestResponseService` to send a SOAP message to CMM.

7. Receive a response SOAP message from CMM.

As in step 3, CMM sends back a response SOAP message once it has processed the request:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
    <soapenv:Header>
       <tws_wrapper soapenv:mustUnderstand="0">
          <trema_web_service_header
         xmlns="http://www.trema.com/externalinterface/ XMLschema"
         xmlns:tws="http://www.trema.com/externalinterface/XMLschema">
             <end_point_url
            xmlns="">http://localhost:1234/cmm/ews/</end_point_url>
             <message_type xmlns="">transaction</message_type>
             <additional_properties xmlns="">
                <name>transaction_function</name>
                <value>2</value>
             </additional_properties>
          </trema_web_service_header>
       </tws_wrapper>
    </soapenv:Header>
    <soapenv:Body>
       <trema_web_service_response
      xmlns="http://www.trema.com/externalinterface/ XMLschema"
      xmlns:ns1="http://www.trema.com/externalinterface/XMLschema">
          <interchange_control_batch_id
         xmlns="">batch-1</interchange_control_batch_id>
          <message_sequence_number xmlns="">0</message_sequence_number>
```

```
            <message_generation_datetime xmlns="">2006-04-05T09:58:46.267-06:00
           </message_generation_datetime>
           <request_success xmlns="">false</request_success>
            <max_number_transactions_permessage xmlns="">0
           </max_number_transactions_permessage>
            <rejection_reason xmlns="">Unexpected Exception: There is no request
           response interchange configured for the import type of:
           2</rejection_reason>
         </trema_web_service_response>
      </soapenv:Body>
   </soapenv:Envelope>
```

8. Repeat step 4 for each transaction data message.

5.5.2.2 Constructing and consuming SOAP messages for the export process

The construction and consumption of SOAP messages for the export process is very similar to the one for the import process except that you are listening for the incoming requests from CMM. Once you have received a SOAP message from CMM, you are responsible for extracting the attachments from the SOAP message using the API in `CaSOAPMessageAttachUtil`.

Therefore, this guide does not document the steps for creating SOAP messages in the export process.

Chapter 6 Using the command line interface

Parties outside of Wallstreet's R&D department can create custom interfaces using technologies of their choice. The command line interface allows these parties to connect their custom interfaces to CMM. This allows your organization to implement interfaces independently of the release schedule for CMM.

6.1 Connecting interface components

Using the command line interface, you can connect interface components developed outside of CMM to CMM. The command line interface supports the following types of interface components:

- Transport mechanisms
- Security mechanisms
- Format processor.

After creating parameters for these components, you can use the parameters in interchanges.

6.1.1 Connecting transport mechanisms

To connect transport mechanisms:

1. In the Communication Protocols function, ensure the Generic Command Line communication protocol is enabled and manually executable.

   In a standard installation, the Generic Command Line communication protocol is enabled and manually executable by default.

   For instructions on enabling communication protocols, see 3.1.1 Managing communication protocols on page 41.

2. Using a technology of your choice, create a batch file for the transport mechanism you want to connect to CMM.

   The technology you use to create the batch file must be installed on the same computer as the CMM application server.

   Wallstreet recommends that you use Ant to execute the batch file, as Ant allows you to be cross-platform compliant. (Ant has an `<exec>` task that allows different commands to be executed based on the operating system being used.)

   When you execute the batch file, CMM sets the following session environment variables:

   - `ENV_LOCAL_PATH`
   - `ENV_REMOTE_PATCH`
   - `ENV_FILE_FILTERS`.

When you execute the batch file for an export process, CMM sets the following additional session environment variables:

- ENV_SOURCE_FILENAME
- ENV_SOURCE_FILE
- ENV_SOURCE_DIR.

The batch file must pass the correct exit code to CMM so that the module can log the code in the database. CMM will interpret the code as follows:

- "0" represents success
- Any other value represents failure.

3. In the Communication Protocol Parameters function, create a command line communication protocol parameter for the transport mechanism.

   For instructions on creating communication protocol parameters, see 3.1.2 Managing communication protocol parameters on page 42.

4. Repeat steps 2 to 3 for each transport mechanism you want to connect to CMM.

6.1.2 Connecting security mechanisms

To connect security mechanisms:

1. In the Signers function, ensure the Generic Command Line Encryption and Generic Command Line Decryption signers are enabled.

   In a standard installation, the Generic Command Line Encryption and Generic Command Line Decryption signers are enabled by default.

   For instructions on enabling signers, see 3.2.1 Managing signers on page 51.

2. Using a technology of your choice, create a batch file for the security mechanism you want to connect to CMM.

   The technology you use to create the batch file must be installed on the same computer as the CMM application server.

   Wallstreet recommends that you use Ant to execute the batch file, as Ant allows you to be cross-platform compliant. (Ant has an <exec> task that allows different commands to be executed based on the operating system being used.)

   When you execute the batch file, CMM sets the following session environment variables:

   - ENV_INPUT_FILENAME
   - ENV_INPUT_FILE
   - ENV_INPUT_DIR
   - ENV_OUTPUT_FILENAME
   - ENV_OUTPUT_FILE
   - ENV_OUTPUT_DIR.

   The bath file must pass the correct exit code to CMM so that the module can log the code in the database. CMM will interpret the code as follows:

   - "0" represents success
   - Any other value represents failure.

3. In the Signer Parameters function, create a command line signer parameter for the security mechanism.

For instructions on creating signer parameters, see 3.2.2 Managing signer parameters on page 52.

4. Repeat steps 2 to 3 for each security mechanism you want to connect to CMM.

6.1.3 Connecting format processors

To connect format processors:

1. Using a technology of your choice, create a batch file for the format processor you want to connect to CMM.

   The technology you use to create the batch file must be installed on the same computer as the CMM application server.

   Wallstreet recommends that you use Ant to execute the batch file, as Ant allows you to be cross-platform compliant. (Ant has an `<exec>` task that allows different commands to be executed based on the operating system being used.)

   When you execute the batch file, CMM sets the following session environment variables:

   - `ENV_INPUT_FILENAME`
   - `ENV_INPUT_FILE`
   - `ENV_INPUT_DIR`
   - `ENV_OUTPUT_FILENAME`
   - `ENV_OUTPUT_FILE`
   - `ENV_OUTPUT_DIR`.

   The batch file must pass the correct exit code to CMM so that the module can log the code in the database. CMM will interpret the code as follows:

   - "0" represents success
   - Any other value represents failure.

2. In the Command Line Processors function, create a command line format processor parameter for the format processor.

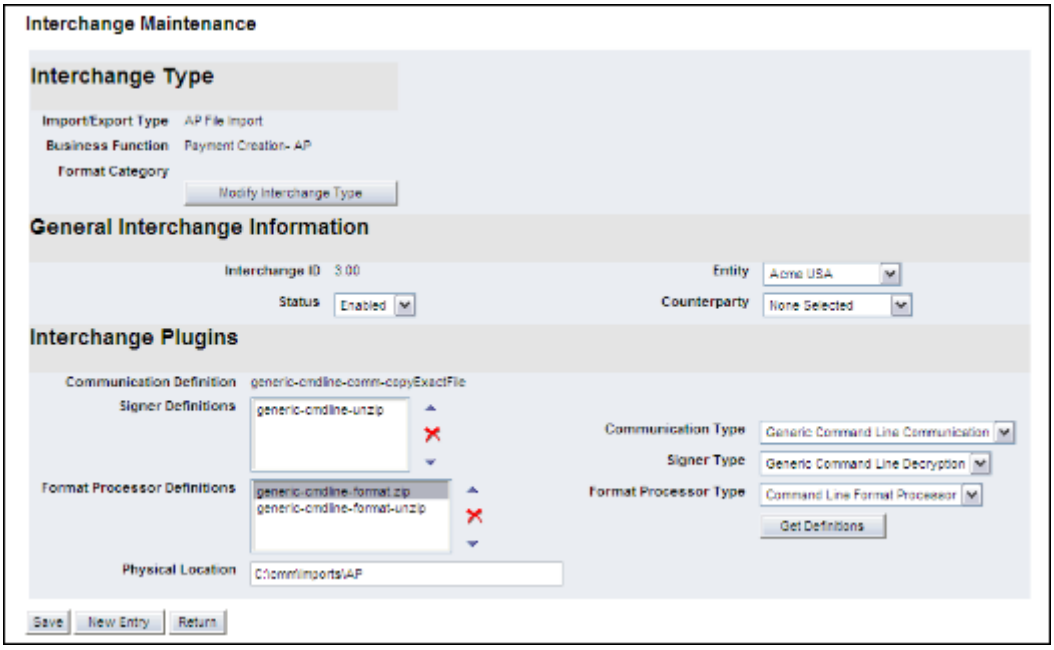
   For instructions on creating format processor parameters, see 3.3 Managing format processors on page 55.

3. Repeat steps 1 to 2 for each format processor you want to connect to CMM.

6.1.4 Configuring interchanges that use the command line parameters

After you have created command line communication protocol, signer, and format processor parameters, you can use them in interchanges.

The following is an example interchange:



When CMM imports accounts payable files using this example interchange, it does the following:

- Copies the source accounts payable files from the remote server using the communication command line (`generic-cmdline-comm-copyExactFile`)
- Places the files in the specified folder (`C:\cmm\Imports\AP`)
- Decrypts the files using the zip tool (`generic-cmdline-unzip`)
- Transforms the files using `generic-cmdline-format.zip` and `generic-cmdline-format-unzip`
- Imports the files' data into CMM.

Note: You do not necessarily have to use command line parameters for all components of an interchange. For example, you could use command line parameters for the communication and format processor components but not the security component.

For detailed instructions on managing interchanges, see 3.4 Managing interchanges on page 56.

6.2 Example implementation of the command line interface

This section presents an example implementation of the command line interface. It shows you the steps involved in setting up and testing the following interfaces created with the command line interface:

Interface	Description
Bank transaction import	CMM retrieves files from a remote FTP server (Remote FTP Server 1) and places them in a temporary folder. It then removes the folders from Remote FTP Server 1 and send them to another remote FTP server (Remote FTP Server 2). This example will support two scenarios: • The files are not encrypted. • The files are encrypted by GnuPG, then signed by GnuPG, and finally compressed by a zip utility.

Interface	Description
Account payable import	CMM retrieves files from Remote FTP Server 1 and places copies in Remote FTP Server 2 before importing them. The files are encrypted by GnuPG, then signed by GnuPG, and finally compressed by a zip utility. The format processor first compresses the files and then decompresses them. This places the files back into their original format so that they can be imported into CMM. Note: In practice, format processors are used to reformat files in foreign data formats to formats that are recognized and supported by CMM.
Payment export	Once the imported payments have been processed as required by CMM, released, and written to files, CMM encrypts, signs, and zips the files before sending them to the bank via FTP. The format processor first compresses the files and then decompresses them. This places the files back into their original format so that they can be exported to the bank. Note: In practice, format processors are used to reformat files in CMM formats into formats that are recognized and supported by banks.

The examples in this section are for demonstration purposes. Therefore, if you decide to create the examples by following the procedures in this section, do so in a test environment rather than a production environment.

6.2.1 Completing prerequisite steps

Before creating the example documented in the subsequent procedures of this section, you must complete the following prerequisite steps:

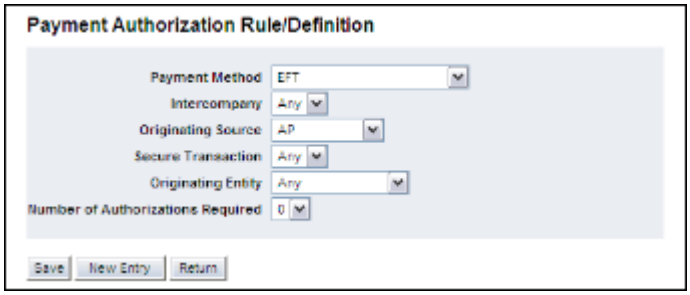
- Install Apache Ant 1.6.0 in `C:\cmm\3rdPartyProgDir\`.

  Apache Ant is available from the Apache website.
- Install GnuPG 1.2.1 in `C:\cmm\3rdPartyProgDir\gpg\bin`.

  GnuPG is available from the GnuPG website.
- Create the following parties:
  - Entities
    - Ent1
  - External banks
    - BOA_FR
    - BOA_US.

  For instructions on creating parties, see the *WebSuite User Guide*.
- Create the following bank accounts:
  - 1019567 (held by Ent1 at BOA_FR)
  - 1010125 (held by Ent1 at BOA_US).

  For instructions on creating bank accounts, see the *WebSuite User Guide*.
- Create the following transaction authorization rule:



For instructions on creating transaction authorization rules, see the *WebSuite System Administration Guide*.

6.2.2 Configuring the transport mechanism

The first step in creating the example implementation of the command line interface is to set up, connect, and test a transport mechanism.

6.2.2.1 Setting up and connecting the transport mechanism

To set up and connect the transport mechanism:

1. Create the following two folders on the remote server:
   - `RemoteFetchFolder`
   - `RemoteFetchFolder2`.
2. Create a user named `ftpuser` with full access to these folders.

   Full access is required because the transport mechanism will copy and delete files in `RemoteFetchFolder` and paste them in `RemoteFetchFolder2`.
3. Open the `cmdline-plugins.xml` file (located in …\3rdPartyProgDir\cmdline\).
4. Edit the marked attributes' values:

```
◦
<!-- Move Files from remote Server 1 to remote Server 2 -->
<target name="moveFiles" description="Move all files matching the regular
expression pattern from remote server 1 to remote server 2">
    <antcall target="showEnvironments">
       <param name="param_env_pattern" value="ENV_"/>
    </antcall>
    <!-- declaration of local properties -->
    <property name="tmp_folder" value="tmp_files_copied_from_server1"/>
    <property name="remote_server1"value="localhost"/>
```

Enter the name of Remote Server 1.

Enter the value of Remote Server 1.

```
 <property name="remote_server2"value="localhost"/>
```

Enter the name of Remote Server 2.

Enter the value of Remote Server 2.

```
 <property name="remote_server2_path"
value="/cmdline/Imports/RemoteFetchFolder2"/>
 <!-- empty the temporary folder -->
 <delete dir="${env.ENV_LOCAL_PATH}/${tmp_folder}" includeemptydirs="true"/>
 <!-- create a temporary folder to store the files being fetched from remote
server 1 -->
 <mkdir dir="${env.ENV_LOCAL_PATH}/${tmp_folder}"/>
```

```
 <!-- get all files matching the regular expression pattern from remote server
1 and store them in ${tmp_folder} -->
 <ftp action="get" server="${remote_server1}" userid="ftpuser"
password="$testING1"depends="yes" verbose="true"
remotedir="${env.ENV_REMOTE_PATH}">
```

Enter the password of ftpuser.

```
    <fileset dir="${env.ENV_LOCAL_PATH}/${tmp_folder}">
       <include name="${env.ENV_FILE_FILTERS}"/>
    </fileset>
 </ftp>
 <!-- delete all files being copied from remote server 1 -->
 <ftp action="del" server="${remote_server1}" userid="ftpuser"
password="$testING1"depends="yes" verbose="true"
remotedir="${env.ENV_REMOTE_PATH}">
```

Enter the password of ftpuser.

```
    <fileset dir="${env.ENV_LOCAL_PATH}/${tmp_folder}">
       <include name="${env.ENV_FILE_FILTERS}"/>
    </fileset>
 </ftp>
 <!-- copy all files being copied from remote server 1 in the temporary folder
to remote server 2 -->
<ftp action="put" server="${remote_server2}" userid="ftpuser"
password="$testING1"depends="yes" verbose="true"
remotedir="${remote_server2_path}">
```

Enter the password of ftpuser.

```
    <fileset dir="${env.ENV_LOCAL_PATH}/${tmp_folder}">
      <include name="${env.ENV_FILE_FILTERS}"/>
    </fileset>
 </ftp>
 <!-- copy all files to CMM import folder from the temporary folder -->
 <copy todir="${env.ENV_LOCAL_PATH}">
    <fileset dir="${env.ENV_LOCAL_PATH}/${tmp_folder}"/>
 </copy>
 <!-- empty the temporary folder -->
 <delete dir="${env.ENV_LOCAL_PATH}/${tmp_folder}" includeemptydirs="true"/>
</target>
°
```

For the purposes of this example, Remote Server 1 and Remote Server 2 are identical. In a production environment, you would change attributes 3 and 4 with the actual host name of the remote FTP servers.

You are responsible for protecting the `cmdline-plugins.xml` file so that unauthorized users cannot open it and access user names and passwords from it.

5. Save and close the file.
6. Create a communication protocol parameter for the transport mechanism:

7. Disable all other communication protocol parameters.

   For instructions on creating and disabling communication protocol parameters, see 3.1.2 Managing communication protocol parameters on page 42.

6.2.2.2 Testing the transport mechanism

To test the transport mechanism:

1. Place the following sample files (provided by Wallstreet) in `RemoteFetchFolder`:
   - `BT_BAI_template_enc_sign_zip.txt`
   - `BT_BAI_template2_enc_sign_zip.txt`.
2. Delete any existing files in `RemoteFetchFolder2`.
3. Select **Admin - Utilities - Bank Interfacing - Communication Dispatch**.
4. In the Communication Dispatch page, select the **Generic Command Line Communication** checkbox and clear all other checkboxes.
5. Click **Dispatch Message**.
6. Review the job log.
7. Confirm that the same files were moved from `RemoteFetchFolder` to `RemoteFetchFolder2`.

6.2.3 Configuring the security mechanisms

The second and third steps in creating the example implementation of the command line interface are to generate and test GnuPG key and to set up, configure, and test security mechanisms.

Note: CMM includes a built-in GnuPG file signer. However, for the purposes of the example, you will using the command line interface to connect an external one to CMM.

6.2.3.1 Generating GnuPG keys

To generate GnuPG keys:

1. Open a command prompt.
2. Navigate to …\3rdPartyProgDir\gpg\bin:

   ```
   $> cd C:\cmm\3rdPartyProgDir\gpg\bin
   ```

3. List the public key ring:

   ```
   $> gpg --homedir . --list-keys
   ```

4. Generate two different GnuPG keys:


```
$> gpg --homedir . --gen-key
```

5. List the secret keys:

```
$> gpg --homedir . --list-secret-keys
.\secring.gpg
-------------
sec 1024D/05D08C89 2005-09-07 Phu Vuong (key does not expire)
<phu.vuong@trema.com>
ssb 1024g/559A0D5D 2005-09-07
sec 1024D/BA045DB0 2005-09-07 Phu Vuong (Test Key 2 does not expire)
<phu.vuong@trema.com>
ssb 1024g/FA71BB85 2005-09-07
```

6.2.3.2 Testing GnuPG keys

To test the GnuPG keys:

1. Open a command prompt.
2. Navigate to …\3rdPartyProgDir\gpg\bin:

```
$> cd C:\cmm\3rdPartyProgDir\gpg\bin
```

3. Encrypt the test file from the sender test key 1 to the recipient test key 2:

```
$> gpg --homedir . --output C:\temp\gpg\BT_BAI_template_enc.txt --local-user
0x05D08C89 --recipient 0xBA045DB0 --encrypt C:\temp\gpg\BT_BAI_template.txt
```

4. Sign the encrypted file:

```
$> gpg --homedir . --output C:\temp\gpg\BT_BAI_template_enc_sign.txt --local-user
0x05D08C89 --recipient 0xBA045DB0 --sign C:\temp\gpg\BT_BAI_template_enc.txt
```

5. Verify the signed test file:

```
$> gpg --homedir . --output C:\temp\gpg\BT_BAI_template_enc_unsign.txt --verify
C:\temp\gpg\BT_BAI_template_enc_sign.txt
```

6. Decrypt the signed test file:

```
$> gpg --homedir . --output C:\temp\gpg\BT_BAI_template_enc_unsign.txt --decrypt
C:\temp\gpg\BT_BAI_template_enc_sign.txt
```

6.2.3.3 Setting up and connecting the security mechanism

To set up and connect the security mechanism:

1. Open the `cmdline-plugins.xml` file (located in …\3rdPartyProgDir\cmdline\).
2. Confirm the following sections are available in the file and configure them as appropriate:
   - GnuPG encryption

```
◦
<target name="gpg_enc" description="Generic GnuPG function to encrypt, sign or
both sign and encrypt the file">
    <antcall target="showEnvironments">
       <param name="param_env_pattern" value="ENV_"/>
    </antcall>
    <!-- required input parameters -->
    <fail unless="param.gpg.op"/>
    <fail unless="param.input.file"/>
    <fail unless="param.output.file"/>
    <fail unless="param.sender.keyid"/>
    <fail unless="param.passphrase"/>
    <fail unless="param.receiver.keyid"/>
    <echo message="GnuPG Home = ${env.GPG_HOME}"/>
    <echo message="param.gpg.op = ${param.gpg.op}"/>
```

```
    <echo message="param.input.file = ${param.input.file}"/>
    <echo message="param.output.file = ${param.output.file}"/>
    <echo message="param.sender.keyid = ${param.sender.keyid}"/>
    <echo message="param.receiver.keyid = ${param.receiver.keyid}"/>
    <echo message=""/>
    <if>
       <equals arg1="${param.gpg.op}" arg2="--sign" />
       <then>
          <property name="__recipient_key" value=""/>
       </then>
       <else>
          <property name="__recipient_key" value="--recipient
         ${param.receiver.keyid}"/>
       </else>
    </if>
    <property name="__gpg_cmd_args" value="--batch --no-tty --yes
   --no-secmem-warning -- homedir ${env.GPG_HOME} --passphrase-fd 0 --output
   ${param.output.file} --local- user ${param.sender.keyid} ${__recipient_key}
   ${param.gpg.op} ${param.input.file}"/>
    <echo message="gpg ${__gpg_cmd_args}"/>
    <exec executable="gpg" inputstring="${param.passphrase}">
       <arg line="${__gpg_cmd_args}"/>
    </exec>
</target>
 …
```

- GnuPG encryption and signing (from the command line)

```
◦
<target name="gpg.encrypt" description="Encrypt the file with GnuPG">
    <antcall target="gpg_enc">
       <param name="param.gpg.op" value="--encrypt"/>
    </antcall>
</target>
<target name="gpg.sign" description="Sign the file with GnuPG">
    <antcall target="gpg_enc">
       <param name="param.gpg.op" value="--sign"/>
    </antcall>
</target>
<target name="gpg.sign.encrypt" description="Sign and encrypt the file with
GnuPG">
    <antcall target="gpg_enc">
       <param name="param.gpg.op" value="--sign --encrypt"/>
    </antcall>
</target>
 …
```

- GnuPG encryption and signing (from CMM)

```
◦
<target name="cmm.gpg.encrypt" description="Encrypt the file with GnuPG
calling from CMM">
    <antcall target="gpg_enc">
       <param name="param.gpg.op" value="--encrypt"/>
       <param name="param.input.file" value="${env.ENV_INPUT_FILE}"/>
       <param name="param.output.file" value="${env.ENV_OUTPUT_FILE}"/>
      <param name="param.sender.keyid" value="0x05D08C89"/>
       <param name="param.passphrase" value="phu.vuong"/>
       <param name="param.receiver.keyid" value="0xBA045DB0"/>
    </antcall>
</target>
<target name="cmm.gpg.sign" description="Sign the file with GnuPG calling from
CMM">
```

```
    <antcall target="gpg_enc">
       <param name="param.gpg.op" value="--sign"/>
       <param name="param.input.file" value="${env.ENV_INPUT_FILE}"/>
      <param name="param.output.file" value="${env.ENV_OUTPUT_FILE}"/>
       <param name="param.sender.keyid" value="0x05D08C89"/>
       <param name="param.passphrase" value="phu.vuong"/>
       <param name="param.receiver.keyid" value="0xBA045DB0"/>
    </antcall>
</target>
<target name="cmm.gpg.sign.encrypt" description="Sign and encrypt the file
with GnuPG calling from CMM">
    <antcall target="gpg_enc">
       <param name="param.gpg.op" value="--sign --encrypt"/>
       <param name="param.input.file" value="${env.ENV_INPUT_FILE}"/>
       <param name="param.output.file" value="${env.ENV_OUTPUT_FILE}"/>
       <param name="param.sender.keyid" value="0x05D08C89"/>
       <param name="param.passphrase" value="phu.vuong"/>
       <param name="param.receiver.keyid" value="0xBA045DB0"/>
    </antcall>
</target>
◦
```

- GnuPG decryption

```
◦
<target name="gpg_dec" description="Generic GnuPG fucntion to verify or
decrypt the data file">
   <antcall target="showEnvironments">
       <param name="param_env_pattern" value="ENV_"/>
    </antcall>
    <!-- required input parameters -->
    <fail unless="param.gpg.op"/>
    <fail unless="param.input.file"/>
    <fail unless="param.output.file"/>
    <fail unless="param.passphrase"/>
    <echo message="GnuPG Home = ${env.GPG_HOME}"/>
    <echo message="param.gpg.op = ${param.gpg.op}"/>
    <echo message="param.input.file = ${param.input.file}"/>
    <echo message="param.output.file = ${param.output.file}"/>
    <echo message=""/>
    <property name="__gpg_cmd_args" value="--batch --no-tty --yes
   --no-secmem-warning -- homedir ${env.GPG_HOME} --passphrase-fd 0 --output
   ${param.output.file} ${param.gpg.op} ${param.input.file}"/>
   <echo message="gpg ${__gpg_cmd_args}"/>
    <exec executable="gpg" inputstring="${param.passphrase}">
       <arg line="${__gpg_cmd_args}"/>
    </exec>
</target>
◦
```

- GnuPG decryption (from the command line)

```
◦
<target name="gpg.verify" description="Verify the signature of the data file">
    <antcall target="gpg_dec">
       <param name="param.gpg.op" value="--verify"/>
    </antcall>
</target>
<target name="gpg.decrypt" description="Decrypt the file with GnuPG">
    <antcall target="gpg_dec">
       <param name="param.gpg.op" value="--decrypt"/>
    </antcall>
</target>
```

...

- GnuPG decryption (from CMM)

```
◦
<target name="gpg.decrypt.k1" description="Decrypt the file with GnuPG using
Test Key 1">
    <antcall target="gpg_dec">
       <param name="param.gpg.op" value="--decrypt"/>
       <param name="param.input.file" value="${env.ENV_INPUT_FILE}"/>
       <param name="param.output.file" value="${env.ENV_OUTPUT_FILE}"/>
       <param name="param.passphrase" value="phu.vuong"/>
    </antcall>
</target>
<target name="gpg.decrypt.k2" description="Decrypt the file with GnuPG using
Test Key 2">
    <antcall target="gpg_dec">
       <param name="param.gpg.op" value="--decrypt"/>
       <param name="param.input.file" value="${env.ENV_INPUT_FILE}"/>
       <param name="param.output.file" value="${env.ENV_OUTPUT_FILE}"/>
       <param name="param.passphrase" value="phu.vuong"/>
    </antcall>
</target>
 ...
```

- Zip compression and decompression

```
◦
<!-- Data Security for Zip Compression -->
<target name="zipFiles" description="Zip up all files.">
    <antcall target="showEnvironments">
       <param name="param_env_pattern" value="ENV_"/>
    </antcall>
    <zip destfile="${env.ENV_OUTPUT_FILE}">
       <fileset dir="${env.ENV_INPUT_DIR}"
      includes="${env.ENV_INPUT_FILENAME}"/>
    </zip>
</target>
<!-- Data Security for unZip Decompression -->
<target name="unzipFiles" description="UnZip all files.">
    <antcall target="showEnvironments">
       <param name="param_env_pattern" value="ENV_"/>
    </antcall>
    <unzip src="${env.ENV_INPUT_FILE}" dest="${env.ENV_OUTPUT_DIR}"/>
</target>
 ...
```

3. Create signer parameters for the security mechanisms:

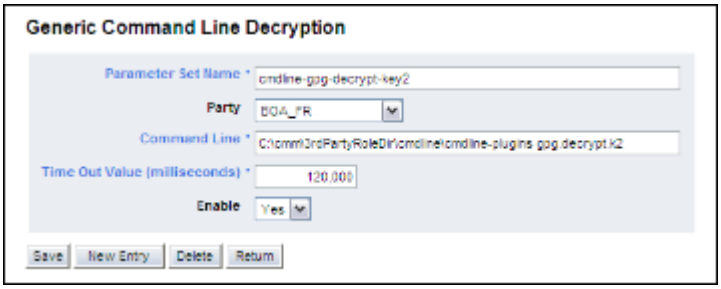
- GnuPG encryption

- GnuPG signing

– GnuPG encryption and signing

– GnuPG decryption (key 1)

– GnuPG decryption (key 2)



– Zip compression

– Zip decompression

For instructions on creating singer parameters, see 3.2.2 Managing signer parameters on page 52.

6.2.3.4 Testing the security mechanisms

To test the security mechanisms:

1. Place the following sample files (provided by Wallstreet) in `C:\temp\gpg`:
   - `BT_BAI_template.txt`
   - `BT_BAI_template2.txt`
   - `AP_FUH_TEST.txt.`
2. Open a command prompt.
3. Enter the following commands to test encryption of the `BT_BAI_template.txt` file:

```
$> set TEST_FILE_NAME=BT_BAI_template
$> set ANT_PROPERTIES=-Dparam.input.file=C:\temp\gpg\%TEST_FILE_NAME%.txt
-Dparam.output.file=C:\temp\gpg\%TEST_FILE_NAME%_enc.txt
-Dparam.sender.keyid=0x05D08C89 -Dparam.passphrase=phu.vuong
-Dparam.receiver.keyid=0xBA045DB0
$> cmdline-plugins gpg.encrypt
$> set ANT_PROPERTIES=-Dparam.input.file=C:\temp\gpg\%TEST_FILE_NAME%_enc.txt
-Dparam.output.file=C:\temp\gpg\%TEST_FILE_NAME%_enc_sign.txt
-Dparam.sender.keyid=0x05D08C89 -Dparam.passphrase=phu.vuong
-Dparam.receiver.keyid=0xBA045DB0
$> cmdline-plugins gpg.sign
$> set ANT_PROPERTIES=-Dparam.input.file=C:\temp\gpg\%TEST_FILE_NAME%.txt
-Dparam.output.file=C:\temp\gpg\%TEST_FILE_NAME%_enc_sign2.txt
-Dparam.sender.keyid=0x05D08C89 -Dparam.passphrase=phu.vuong
-Dparam.receiver.keyid=0xBA045DB0
$> cmdline-plugins gpg.sign.encrypt
$> mkdir C:\temp\gpg\zip $> del /f /s /q C:\temp\gpg\zip
$> copy C:\temp\gpg\%TEST_FILE_NAME%_enc_sign.txt
C:\temp\gpg\zip\%TEST_FILE_NAME%_enc_sign_zip.txt
$> del /f /q C:\temp\gpg\%TEST_FILE_NAME%_enc_sign_zip.txt
$> zip -j C:\temp\gpg\%TEST_FILE_NAME%_enc_sign_zip.txt
C:\temp\gpg\zip\%TEST_FILE_NAME%_enc_sign_zip.txt
```

   Ensure the file being zipped has the same name as the zipped file. Otherwise, when the file is unzipped, CMM will not be able to find it during the import process.
4. Enter the following commands to test decryption of the `BT_BAI_template.txt` file:

```
$> mkdir C:\temp\gpg\out
$> set ANT_PROPERTIES=-Dparam.input.file=C:\temp\gpg\%TEST_FILE_NAME%_enc.txt
-Dparam.output.file=C:\temp\gpg\out\%TEST_FILE_NAME%_enc.out
-Dparam.passphrase=phu.vuong
$> cmdline-plugins gpg.decrypt
$> set
ANT_PROPERTIES=-Dparam.input.file=C:\temp\gpg\%TEST_FILE_NAME%_enc_sign.txt
```



```
-Dparam.output.file=C:\temp\gpg\out\%TEST_FILE_NAME%_enc_sign.out
-Dparam.passphrase=phu.vuong
$> cmdline-plugins gpg.verify $> unzip -o
C:\temp\gpg\%TEST_FILE_NAME%_enc_sign_zip.txt -d C:\temp\gpg\unzip
$> set
ANT_PROPERTIES=-Dparam.input.file=C:\temp\gpg\unzip\%TEST_FILE_NAME%_enc_sign_zi
p.txt -Dparam.output.file=C:\temp\gpg\unzip\%TEST_FILE_NAME%_enc_sign_zip.txt.txt
-Dparam.passphrase=phu.vuong
$> cmdline-plugins gpg.decrypt
$> set
ANT_PROPERTIES=-Dparam.input.file=C:\temp\gpg\unzip\%TEST_FILE_NAME%_enc_sign_zi
p.txt.txt
-Dparam.output.file=C:\temp\gpg\unzip\%TEST_FILE_NAME%_enc_sign_zip.txt.txt.txt
-Dparam.passphrase=phu.vuong
$> cmdline-plugins gpg.decrypt
```

5. Repeat steps 3 to 4 for the `BT_BAI_template2.txt` file.
6. Repeat steps 3 to 4 for the `AP_FUH_TEST.txt` file.

6.2.4 Configuring the format processors

The fourth step in creating the example implementation of the command line interface is to set up and connect format processors.

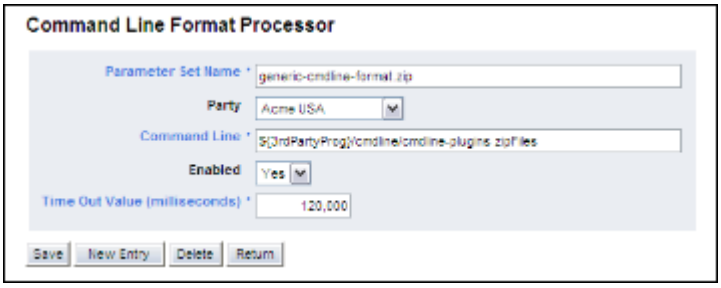
6.2.4.1 Setting up and connecting format processors

To set up and connect format processors:

1. Open the `cmdline-plugins.xml` file (located in …\3rdPartyProgDir\cmdline\).
2. Confirm the following section is available in the file and configure it as appropriate:

```
◦
<!-- Data Security for Zip Compression -->
<target name="zipFiles" description="Zip up all files.">
    <antcall target="showEnvironments">
       <param name="param_env_pattern" value="ENV_"/>
    </antcall>
    <zip destfile="${env.ENV_OUTPUT_FILE}">
       <fileset dir="${env.ENV_INPUT_DIR}"
      includes="${env.ENV_INPUT_FILENAME}"/>
    </zip>
</target>
<!-- Data Security for unZip Decompression -->
<target name="unzipFiles" description="UnZip all files.">
    <antcall target="showEnvironments">
       <param name="param_env_pattern" value="ENV_"/>
    </antcall>
    <unzip src="${env.ENV_INPUT_FILE}" dest="${env.ENV_OUTPUT_DIR}"/>
</target>
 …
```

3. Create format processor parameters for the format processors:
   - Zip compression



– Zip decompression

For instructions on creating format processor parameters, see 3.3 Managing format processors on page 55.

6.2.5 Importing bank transaction files using the command line interface

After configuring the transport mechanism and security mechanisms, you can create and then test an interface to import bank transaction files.

6.2.5.1 Creating the bank transaction import interface

To create the bank transaction import interface:

1. In the `fileimportexportformats.xml` file, ensure format ID 1's `modeltypecode` attribute is set to "2":

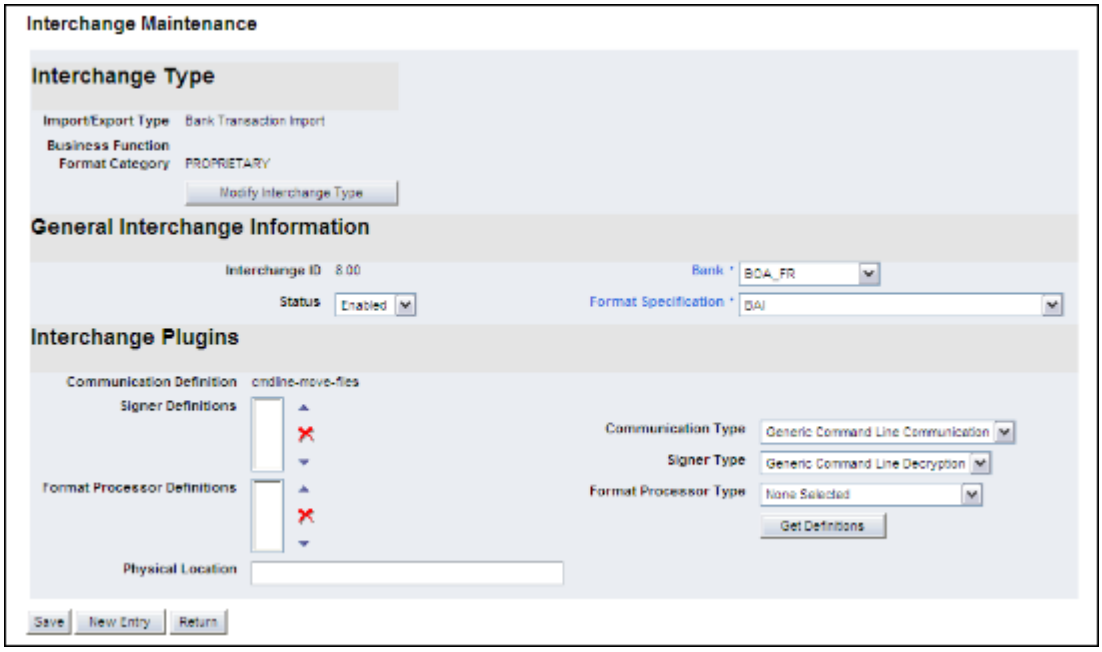
   ◦

   ```
   <row_data fileformatuniqueid="1" importexporttypeid="1" fileformatcode="BAI"
   fileformatname="BAI" fileformatdesc="Bank Administration Institute"
   formatcategoryid="4" modeltypecode="2"/>
   ```

   ◦

   For instructions on extracting and editing the `fileimportexportformats.xml` file, see 2.1 Configuring the fileimportexportformats.xml file on page 37.

2. Create the following interchange:



For instructions on creating interchanges, see 3.4 Managing interchanges on page 56.

6.2.5.2 Testing the bank transaction import interface with unencrypted files

To test the bank transaction import interface with unencrypted files:

1. Place the following sample files (provided by Wallstreet) in `RemoteFetchFolder`:
   - `BT_BAI_template.txt`
   - `BT_BAI_template2.txt`.

   If you have previously imported these files, you need to undo the imports before continuing.
2. In the Import Bank Transaction Files function, import the files.
3. In the Review Job Log function, review the jog log to ensure the import was successful.
4. Confirm that the files were moved from `RemoteFetchFolder` to `RemoteFetchFolder2`.

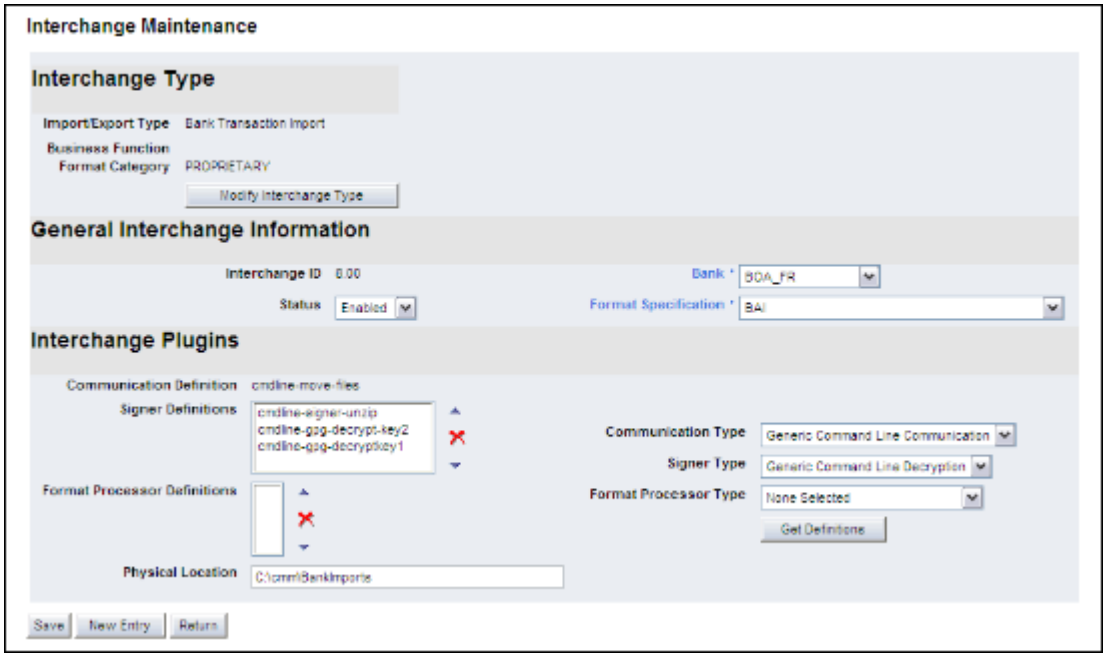
For instruction on importing bank transaction files and reviewing the jog log, see the *WebSuite User Guide* and the *WebSuite System Administration Guide*.

6.2.5.3 Testing the bank transaction import interface with encrypted files

To test the bank transaction import interface with encrypted files:

1. Place the following sample files (provided by Wallstreet) in `RemoteFetchFolder`:
   - `BT_BAI_template_enc_sign_zip.txt`
   - `BT_BAI_template2_enc_sign_zip.txt`.

   If you have previously imported these files, you need to undo the imports before continuing.
2. Edit the previously created interchange:



For instructions on editing interchanges, see 3.4 Managing interchanges on page 56.

3. In the Import Bank Transaction Files function, import the files.
4. In the Review Job Log function, review the jog log to ensure the import was successful.
5. Confirm that the files were moved from `RemoteFetchFolder` to `RemoteFetchFolder2`.

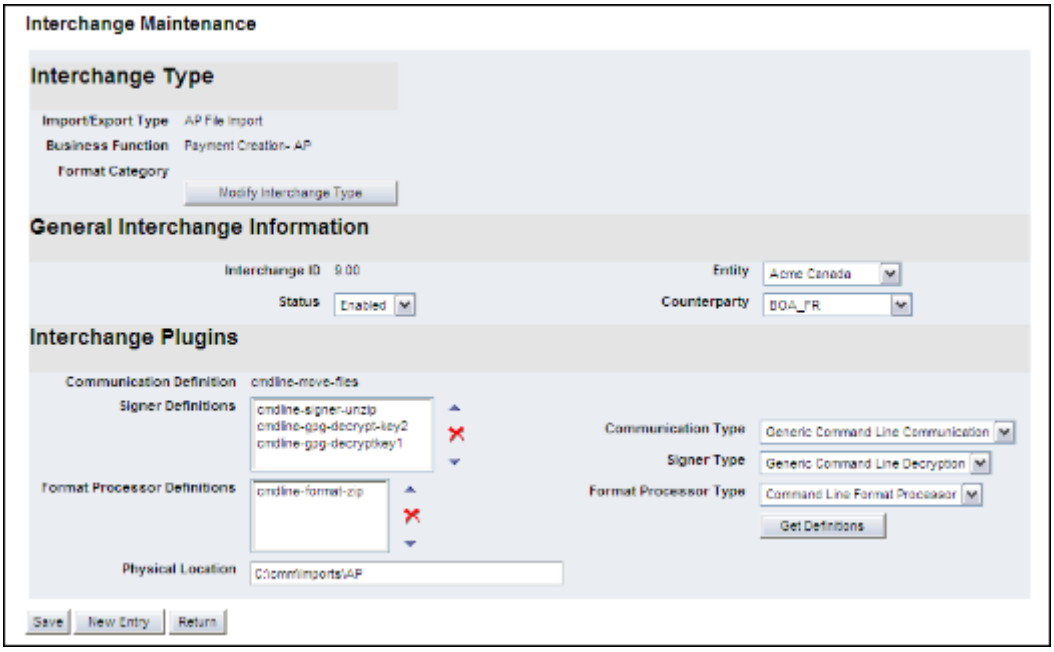
For instruction on importing bank transaction files and reviewing the jog log, see the *WebSuite User Guide* and the *WebSuite System Administration Guide*.

6.2.6 Importing accounts payable files using the command line interface

After configuring the transport mechanism, security mechanisms, and format processors you can create and then test an interface to import accounts payable files.

6.2.6.1 Creating the accounts payable import interface

Create the following interchange:



For instructions on creating interchanges, see 3.4 Managing interchanges on page 56.

6.2.6.2 Testing the accounts payable import interface

To test the accounts payable import interface:

1. Place the following sample file (provided by Wallstreet) in `RemoteFetchFolder`:
   - `AP_FUH_TEST_enc_sign_zip.txt`.

   If you have previously imported this file, you need to undo the import before continuing.
2. In the Import Transaction Files function, import the file.
3. In the Review Job Log function, review the jog log to ensure the import was successful.
4. Confirm that the file was moved from `RemoteFetchFolder` to `RemoteFetchFolder2`.

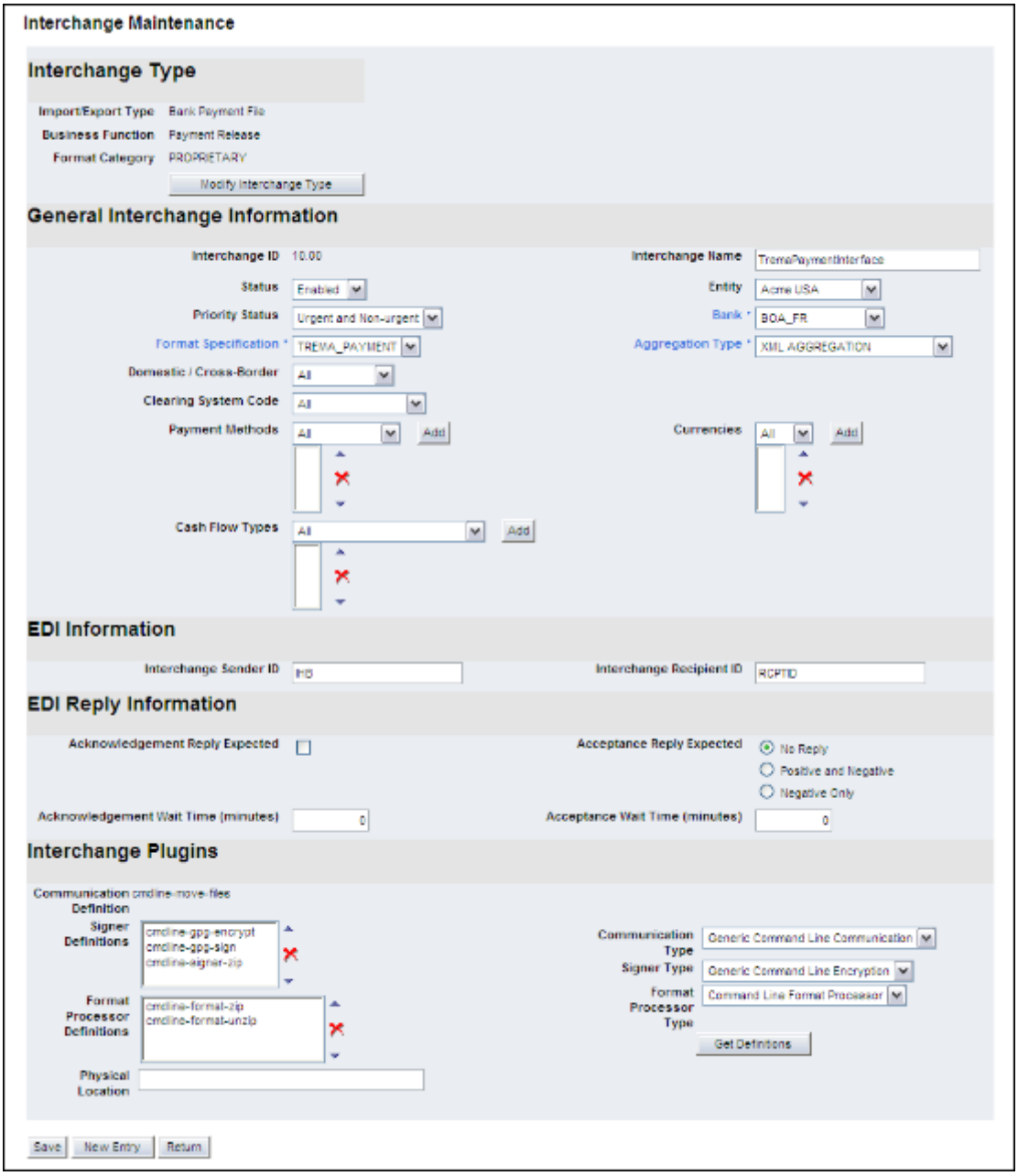
For instruction on importing transaction files and reviewing the jog log, see the *WebSuite User Guide* and the *WebSuite System Administration Guide*.

6.2.7 Exporting payment files using the command line interface

After configuring the transport mechanism, security mechanisms, and format processors you can create and then test an interface to export payment files.

6.2.7.1 Creating the payment export interface

Create the following interchange:



For instructions on creating interchanges, see 3.4 Managing interchanges on page 56.

6.2.7.2 Testing the payment export interface

To test the payment export interface:

1. In the Release Payments function, release the transactions you imported from the accounts payable file.
2. In the Review Job Log function, review the jog log to ensure the export was successful.

For instruction on exporting payments and reviewing the jog log, see the *WebSuite User Guide* and the *WebSuite System Administration Guide*.

Chapter 7 Using XML-template-based formats

Using XML templates, you can create and implement custom formats for the following import and export processes:

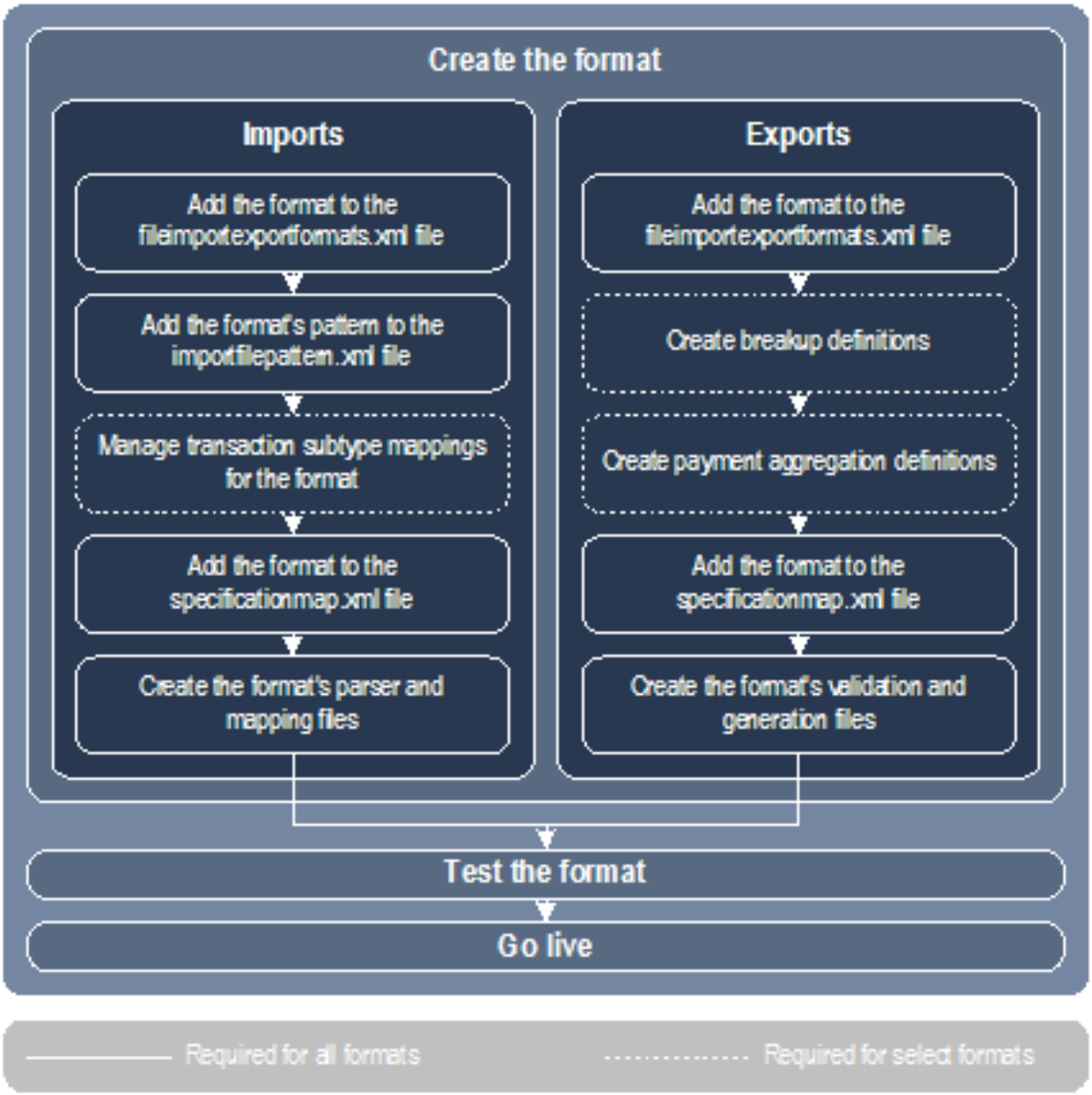
Imports	Exports
• Forecasts • Payments (AP) • Receipts (AR) • Direct debits • Bank statements • Bank messages.	• Payments • Direct debits • Letters of credit • Preadvices • Credit advices • Bank statements • General ledger postings.

Note: Unlike the web services and command line interfaces, XML templates only apply to the format components of interfaces—not the connection point, transport mechanism, and security mechanism components.

There are several benefits to creating and implementing custom formats with XML templates rather than using standard formats:

- Your organization has control over its custom formats.
- Custom formats reside outside of the CMM `DefaultData` folder (see 1.5 Opening configuration files on page 32) and, therefore, are not overwritten when your organization upgrades to a new CMM release or patch.
- You can modify custom formats without having to request and wait for changes by Wallstreet.
- You can quickly and easily create variations of formats for new countries or banks with specific needs.

Creating and implementing a custom format using XML templates involves three stages:



7.1 Creating custom formats

Creating a custom format involves different steps depending on whether the format is for imports or exports.

Note: This chapter assumes you are completing these steps in a test environment. In a later section, 7.3 Going live on page 132, you will transfer the files you created while completing these steps to a production environment.

7.1.1 Creating custom import formats

Creating a custom import format involves five steps:

1. Add the format to the `fileimportexportformats.xml` file.
2. Add the format's pattern to the `importfilepattern.xml` file.
3. Manage transaction subtype mappings for the format (if required).
4. Add the format to the `specificationmap.xml` file.
5. Create the format's parser and mapping files.

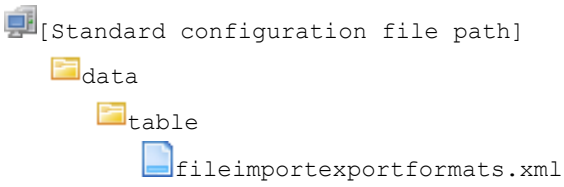
7.1.1.1 Adding formats to the fileimportexportformats.xml file

In 2.1 Configuring the fileimportexportformats.xml file on page 37, you edited the `fileimportexportformats.xml` file and removed unneeded formats from it to ensure optimal performance.

In this step, you will add a format to the `fileimportexportformats.xml` file.

To add formats to the `fileimportexportformats.xml` file:

1. Open the following configuration file:

   For instructions on opening configuration files, see "Opening configuration files" on page32.

2. In the `table_data` element, enter a `row_data` child element for the first format you want to add.

The following table defines the attributes to include in the `row_data` child element:

Attribute	Value
`fileformatuniqueid`	The format's unique ID. Note: Wallstreet recommends you use unique IDs of 1,000 or higher for custom formats.
`importexporttypeid`	The format's import/export type: • `1` for bank transaction import • `2` for accounts payable file import • `3` for foreign exchange rate import • `4` for external general ledger file export • `5` for payment file export • `6` for accounts receivable file import • `7` for counterparty import • `8` for bank account import • `9` for counterparty bank import • `10` for bank message import • `11` for remittance advice export • `12` for interest rate import • `13` for deal import • `17` for deal confirmation export • `19` for bank account confirmation message export • `22` for negative payment confirmation message export • 23 for export of bank payment file - VALIDATE • `24` for entity import • `25` for receipt file export • `26` for positive payment confirmation message export • `27` for direct debit file import • `28` for direct debit file export • `29` for bank transaction export • `30` for positive pay file export • `31` for requisition message export • `32` for transaction export for posting • `33` for bank balance export • `37` for letter of credit file export • 38 for forecast import • `41` for credit advice file export.
`fileformatcode`	The format's code. Note: This code will be mapped to the format's corresponding `specificationmap.xml` file and will be referenced in the `importfilepattern.xml` file.
`fileformatname`	The format's name. Note: Be descriptive, as the value you enter in this attribute displays in **Format Supplications** lists in the Interchanges function.
`fileformatdesc`	A detailed description of the format. Note: This attribute is optional.

Attribute	Value
formatcategoryid	The format's category: • 1 for EDIFACT • 2 for X12 • 3 for SWIFT • 4 for proprietary.
modeltypecode	2. Note: Entering 2 in this attribute indicates the format is XML template based.

3. Repeat step 2 for each format you want to add.
4. Save and close the file.

7.1.1.2 Adding format patterns to the importfilepattern.xml file

The importfilepattern.xml file you edited in 2.2 Configuring the importfilepattern.xml file on page 38 defines the format patterns that allow CMM to correctly identify and parse import files.

As part of creating a custom import format, you must add the format's pattern to the importfilepattern.xml file.

To add format patterns to the importfilepattern.xml file:

1. Open the following configuration file:

    [Standard configuration file path]

    interfaces

    importfilepattern.xml

   For instructions on opening configuration files, see "Opening configuration files" on page32.

2. In the import_file_pattern element, enter a pattern child element for the first format pattern you want to add.

The following table defines the attributes to include in the `pattern` child element:

Attribute	Value
`format_code`	The format pattern's code. Note: The value you enter in this attribute must exactly match the value you entered in the appropriate `fileformatcode` attribute in the `fileimportexportformats.xml` file.
`pattern`	A regular expression that specifies the format pattern. The following is an example for a FINSTA format: `(?s)UNB\+.*?UNH\+.*?\+FINSTA.*?:9` Note: Ensure your regular expressions patterns are unique to identify all formats you plan to import into CMM.
`format_sub_type`	The format pattern's subtype. Note: This attribute is optional.
`format_type`	The format pattern's type. Note: This attribute is optional.
`business_type`	`ACKNOWLEDGEMENT` or `ACCEPTANCE`. Note: This attribute is only required for bank message import format patterns.

3. Repeat step 2 for each format pattern you want to add.
4. Save and close the file.

7.1.1.3 Creating transaction subtype mappings for formats

For select formats (currently bank transaction import and bank message import formats), you need to create transaction subtype mappings so that CMM properly processes a file's records after importing the file.

For more information on creating transaction subtype mappings, see 4.1 Managing transaction subtype mappings on page 61.

7.1.1.4 Adding formats to the appropriate specification map files

The `specificationmap.xml` file defines the import/export type codes used in the `importexporttypeid` attributes of the `fileimportexportformats.xml` file.

Note: The `specificationmap.xml` file is included in the CMM `DefaultData` folder. See 1.5 Opening configuration files on page 32.

The `specificationmap.xml` file references six other XML configuration files:

- `apimportspectmap.xml`
- `arimportspectmap.xml`
- `balanceandtransactionimportspectmap.xml`
- `bankmessageimportspectmap.xml`
- `ddimportspectmap.xml`
- `forecastimportspectmap.xml`.

Each of these files defines the formats for it respective import/export type as shown in the following example:

```
<?xml version="1.0"?>
<spec_map>
```

```
    <condition_test_element condition_type="switch">
       <attribute_id value="format_specification"/>
       <result_handler status_code="BAI">
          <include filename="interfaces.imports.demo.bai.baispec.xml"/>
       </result_handler>
       <result_handler status_code="MEESFLEX">
          <include filename="interfaces.imports.demo.meesflex.spec.xml"/>
       </result_handler>
       <result_handler status_code="SWIFT">
          <include filename="interfaces.imports.swift.standard.bank_statement.
         swiftspec.xml"/>
       </result_handler>
       <result_handler status_code="TREMA_BANK_STATEMENT">
          <include filename="interfaces.imports.trema_bank_statement.
         trema_statement_spec.xml"/>
       </result_handler>
       <result_handler status_code="default">
          <abort message="There is no matching spec for format
         ${format_specification}, check your spec map xml files"/>
       </result_handler>
    </condition_test_element>
</spec_map>
```

If you attempt to import a file in a format not defined in these files, the following error message displays:

```
There is no matching spec for format ${format_specification}, check your spec map xml
files.
```

To prevent this error from displaying for your custom formats, you must add the formats to the appropriate specification map files:

1. Open one of the following configuration files:

   
   [Standard configuration file path]

   
   interfaces

   
   imports

   
   apimportspectmap.xml

   
   arimportspectmap.xml

   
   balanceandtransactionimportspectmap.xml

   
   bankmessageimportspectmap.xml

   
   ddimportspectmap.xml

   
   forecastimportspectmap.xml

   For instructions on opening configuration files, see “Opening configuration files” on page32.

2. In the `condition_test_element` element, enter a `result_handler` child element for the first format you want to add. The following is an example:

```
<?xml version="1.0"?>
<spec_map>
    <condition_test_element condition_type="switch">
       ...
       <result_handler status_code="ACME_BANK_STATEMENT">
         <include
         filename="interfaces.imports.acme.acme_bank_statement_spec.xml"/>
      </result_handler>
       ...
    </condition_test_element>
```

```
</spec_map>
```

3. Repeat step 2 for each format you want to add.
4. Save and close the file.

7.1.1.5 Creating format parser and mapping files

The third parties from which you are importing information can provide you with specifications that define their format and business rule requirements. You need to map these requirements to your XML template-based format.

Note: If an existing format in CMM closely matches a third party's format and business rule requirements, consider using it as a template rather than creating a new format.

CMM includes a set of attribute and handlers that you can use to define your formats. Using the attributes and handlers, you can take advantage of all of the information in an import file and ensure the file is parsed correctly. For more information on attributes and handlers, see Appendix B Attributes on page 311 and Appendix C Handlers on page 369.

There may be differences in how you want to approach the types of files that can be imported into CMM. For example, you may want to treat statement and message files from banks differently from files generated internally, or you may want to create separate formats for accounts payable files depending on their sources.

Depending on the complexity of the information you are importing, you may decide to use a tree:

Type	Example	Procedure
Simple	A fixed-length-based format	When you read the data, you know what field you are reading and you can directly set the field read from the file to the business object (for example, an AP data record), or you can save the value in context first and then manipulate the value if required and then set the manipulated context variable to the business object.
Complex	EDIFACT and BAI	When you read the data, you do not always know what attribute of the business object to which you should map, or it may be difficult to manage the mapping logic. Creating a tree allows you to put those values read from the file to a visualized tree structure. As a result, you can print out the tree if you want to look at it before mapping. After the tree is established, you can traverse it with ease, and map the proper node of the tree to the correct business object.

You can use a tree for simple information. However, Wallstreet does not recommend using a tree if you can manage without it to avoid unnecessary steps and memory consumption. You may decide to use trees initially but later move away from them as you become familiar with creating formats.

Because most banks use standard formats but can implement them in slightly different ways, you can adapt a format after creating it for subsequent banks, countries, or both. You have the option of creating multiple versions of a format or making the processing conditional within a single format. Should the rule for the variation be dependent upon something fixed, such as a bank or country, it is best to create a separate format that can be attached to a specific interchange rather than making the original format extremely complex so it applies to many bank variations.

Wallstreet can provide you with several example of standard and custom formats.

7.1.2 Creating custom export formats

Creating a custom export format involves five steps:

1. Add the format to the `fileimportexportformats.xml` file.
2. Create breakup definitions.
3. Create payment aggregation definitions.
4. Add the format to the `specificationmap.xml` file.
5. Create the format's validation and generation files.

7.1.2.1 Adding formats to the fileimportexportformats.xml file

For information on adding formats to the `fileimportexportformats.xml` file, see 7.1.1 Creating custom import formats on page 121.

7.1.2.2 Creating breakup definitions

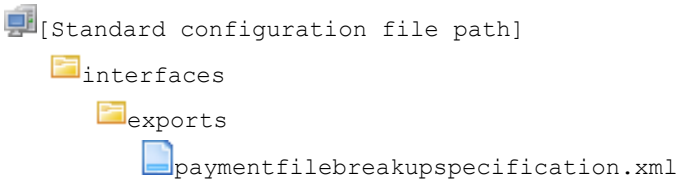
Breakup definitions allow you to generate multiple files, batches, or both for the same format in a single release. You can break up files according to the specifications of the receiving system or user-defined business drivers.

You can group transactions using multiple grouping attributes. Grouping attributes are set in such a way that the transactions must match all specified attributes to be placed in the same file. For example, all payments must be paid from the same bank account and have the same value date. Any payment which does not match these two criteria generates a separate file or batch.

You can also group transactions using a maximum group size. This can be used alone or in conjunction with defined group attributes. This limits the number of transactions that can be included in a single file or batch even though they share the same specified group attributes.

To create a breakup definition:

1. Open the following configuration file:

   For instructions on opening configuration files, see "Opening configuration files" on page32.

2. Change the `build_payment_groups` attribute for each format you want to modify. The following is an example:

```
<result_handler status_code="Datanet">
   <build_txn_groups group_id="default_id">
      <group_by attribute_id="payor_bank_account_id"/>
      <group_by attribute_id="value_date"/>
   </build_txn_groups>
</result_handler>
```

   For more information on the `build_txn_groups` handler, see C.4 build_txn_groups on page 373.

3. Save and close the file.

7.1.2.3 Creating payment aggregation definitions

When creating an interchange, you must define its aggregation type by selecting the appropriate value in the **Aggregation Type** list. If you are using a custom format, you must set this list to `XML AGGREGATION`. (If you select `COMMUNITY AGGREGATION`, `MULTI CREDIT AGGREGATION`, `NETTING AGGREGATION`, or `SINGLE CREDIT AGGREGATION`, CMM will attempt to aggregate data using a legacy Java-based method. If you select, `NO AGGREGATION`, CMM does not attempt to aggregate data.)

You do not have to make any changes to CMM to use default aggregation. However, you can create new payment aggregation definitions and apply them to specific formats if required. However, if you create a new payment aggregation definition for a format, the new definition will continue to display as `XML AGGREGATION` in the **Aggregation Type** list of the Interchanges function.

Note: If a format does not have a payment aggregation definition, no aggregation is completed when you export files using that format.

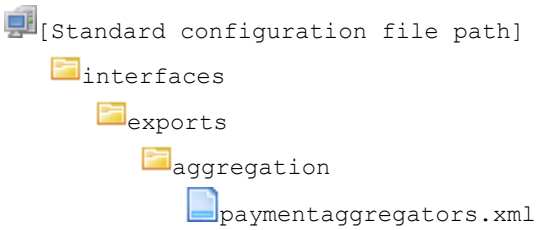
To create a payment aggregation definition:

1. Using a text editor, create an aggregation definition file. The following is an example that groups payments by paying bank account, value date, currency, delivery channel, domestic/cross-border status, transaction type, and batch ID:

```
<?xml version="1.0" ?>
<txn_aggregation type="AP Batch">
    <aggregation_key>
       <attribute id="payor_bank_account_id"/>
       <attribute id="value_date"/>
       <attribute id="payment_currency"/>
       <attribute id="payment_primary_delivery_channel"/>
       <attribute id="domestic_crossborder_status"/>
       <attribute id="transaction_type_code"/>
       <attribute id="source_batch_id"/>
    </aggregation_key>
</txn_aggregation>
```

   For more information on the `txn_aggregation` handler, see C.48 txn_aggregation on page 413.

2. Save the file with an appropriate name and the `.xml` extension in …\InstallationData\interfaces\exports\aggregation\.
3. Close the file.
4. Open the following configuration file:



   For instructions on opening configuration files, see "Opening configuration files" on page32.

5. Change the `filename` attribute for each format for which you want to use the payment aggregation definition. The following is an example:

```
<result_handler status_code="CRG_PAYMUL">
    <include filename="interfaces.exports.aggregation.acmeaggregator.xml"/>
</result_handler>
```

6. Save and close the file.

7.1.2.4 Adding formats to the specificationmap.xml file

For information on adding formats to the `specificationmap.xml` file, see 7.1.1 Creating custom import formats on page 121.

7.1.2.5 Creating format validation and generation files

The third parties to which you are exporting information can provide you with specifications that define their format and business rule requirements. You need to map these requirements to your XML template-based format.

Note: If an existing format in CMM closely matches a third party's format and business rule requirements, consider using it as a base template rather than creating an entirely new format.

CMM includes a set of attributes and handlers that you can use to define your formats. Using the attributes and handlers, you can execute complex logic to ensure the final format is correct for the third party. For more information on attributes and handlers, see Appendix B Attributes on page 311 and Appendix C Handlers on page 369.

Because most banks use standard formats but can implement them in slightly different ways, you can adapt a format after creating it for subsequent banks, countries, or both. You have the option of creating multiple versions of a format or making the processing conditional within a single format. Should the rule for the variation be dependent upon something fixed, such as a bank or country, it is best to create a separate format that can be attached to a specific interchange rather than making the original format extremely complex so it applies to many bank variations.

Wallstreet can provide you with several examples of standard and custom formats.

7.2 Testing custom formats

After completing the steps in 7.1 Creating custom formats on page 120 but before going live, you can test your new custom formats.

7.2.1 Testing custom formats

To test custom formats:

1. Create one or more interchanges for the new formats.

   For information on creating interchanges, see 3.4 Managing interchanges on page 56.

2. Do the following:
   - If you are testing import formats, ensure properly formatted test files are available in the correct locations and the required static data are available in CMM.
   - If you are testing export formats, ensure test data are available in CMM to export.

3. Use the appropriate functions to import or export data:

Type	Function
Imports	
Forecasts	Import Forecasts
Payments (AP)	Import Transaction Files
Receipts (AR)	Import AR Files
Direct debits	Import Transaction Files
Bank statements	Import Bank Transaction Files
Bank messages	Import Bank Message Files
Exports	
Payments	Release Payments
Direct debits	Release Receipts
Letters of credit	Release Receipts

Type	Function
Preadvices	Release Receipts
Credit advices	N/A
Bank statements	Export Bank Statements
General ledger postings	Export G/L Postings

For information on using these functions, see their context-sensitive help.

4. Review the job log to ensure the imports and exports were successful:
   - If all imports and exports were successful, continue to step 5.
   - If an import or export was not successful, correct any problems with the format and repeat steps 1 to 4 until the import or export is successful.

   For information on the job log, see the *WebSuite System Administration Guide*.
5. Do the following:
   - If you are testing import formats, check the resulting records in the database against the import file.
   - If you are testing export formats, check the resulting export files against the records in the database and, if required, submit the files to the appropriate banks or other external systems to ensure they can process the files.

7.2.2 Troubleshooting

If you are encountering errors while testing custom formats, you need to troubleshoot.

To troubleshoot import formats:

- Review template parsing (if the format creates a tree during parsing):
  a. Select **Admin - Utilities - Analysis - Message Log**.
  b. In the Message Control Data Maintenance - Selection page, click **0Appserver.TemplateParsing**.
  c. In the Message Levels section of the Message Control Data Maintenance page, select the **Verbose** checkbox.
  d. In the Output Destinations section of the Message Control Data Maintenance page, select the CMM **Log** checkbox.
  e. Click **Save**.

     The next time you import or export data, the logging is enabled.
  f. Review the template parsing log, which is available on the CMM server at …\VirtualDirectory\logs\ml.TemplateParsing.YYYY_MM_DD_HH_MM_SS.txt (where `YYYY_MM_DD_HH_MM_SS` is the day and time on which the log was generated).

     After you have reviewed template parsing, return to the Message Log function to turn off the logging.
- Review echo messages:
  a. Add the `echo` handler to the configuration files you want to troubleshoot. The following is an example:

     ```
     <echo message="in standardrecord 1" verbose="true"/>
     ```

For more information on the `echo` handler, see C.14 echo on page 386.

b. Select **Admin - Utilities - Analysis - Message Log**.

c. In the Message Control Data Maintenance - Selection page, click **0AppserverXMLEchoMessage**.

d. In the Message Levels section of the Message Control Data Maintenance page, select the appropriate message levels' checkboxes. (The message levels you selected in this step should be same as the ones you specified in the echo handler in step a.)

e. In the Output Destinations section of the Message Control Data Maintenance page, select the appropriate output destinations' checkboxes.

f. Click **Save**.

The next time you import or export data, the logging is enabled.

g. Select **Admin - Utilities - Analysis - Log Viewer**.

h. In the Log Viewer page, drill down on the `ml.user.alterna.YYYY_MM_DD_HH_MM.txt` log file (where `YYYY_MM_DD_HH_MM` is the day and time on which the log was generated).

- Review the import's record in the job log for an indication of the import's status.

  For more information on reviewing the job log, see the *WebSuite System Administration Guide*.

- If you encounter exceptions such as file errors or unknown errors, review the appropriate log file in the Log Viewer function. (You may need to send this file to Wallstreet for further diagnose.)

  For more information on using the Log Viewer function, see the *WebSuite System Administration Guide*.

To troubleshoot export formats:

- Review echo messages:

  a. Add the `echo` handler to the configuration files you want to troubleshoot. The following is an example:

  ```
  <echo message="in standardrecord 1" verbose="true"/>
  ```

  For more information on the `echo` handler, see C.14 echo on page 386.

  b. Select **Admin - Utilities - Analysis - Message Log**.

  c. In the Message Control Data Maintenance - Selection page, click **0AppserverXMLEchoMessage**.

  d. In the Message Levels section of the Message Control Data Maintenance page, select the appropriate message levels' checkboxes. (The message levels you selected in this step should be same as the ones you specified in the echo handler in step a.)

  e. In the Output Destinations section of the Message Control Data Maintenance page, select the appropriate output destinations' checkboxes.

  f. Click **Save**.

  The next time you import or export data, the logging is enabled.

  g. Select **Admin - Utilities - Analysis - Log Viewer**.

  h. In the Log Viewer page, drill down on the `ml.user.alterna.YYYY_MM_DD_HH_MM.txt` log file (where `YYYY_MM_DD_HH_MM` is the day and time on which the log was generated).

- Review the export's record in the job log for an indication of the export's status.

  For more information on reviewing the job log, see the *WebSuite System Administration Guide*.

- If you encounter exceptions such as file errors or unknown errors, review the appropriate log file in the Log Viewer function. (You may need to send this file to Wallstreet for further diagnose.)

  For more information on using the Log Viewer function, see the *WebSuite System Administration Guide*.

7.3 Going live

Once you have completed testing and are satisfied with the custom formats, you can move them to a production environment.

Warning: Test all possible scenarios before going live.

To do this:

1. Copy the affected configuration files from the test environment to the production environment.

   If one of the files you are copying over is `fileimportexportformats.xml`, you need to restart the production environment's application server as well to ensure the changes in the `fileimportexportformats.xml` file are reflected in the database.

2. Repeat the steps in 7.2 Testing custom formats on page 129 in the production environment to create and test interchanges in that environment.

3. Complete additional testing to satisfy your organization's and the bank's (or other external system's) requirements for production interfaces. Specifically, complete nominal value testing, where very low value payments are created from the affected account(s) to another corporate entity, and sent to the bank as an actual payment file. Once those payments have been successfully executed by the bank and all appropriate message files received back, the interface is ready to go live.

Note: Most banks do not accept payments from new interfaces until they have certified the interfaces for format and content.

Appendix A Standard formats

Wallstreet offers two standard format types:

- XML (`.xml`)
- Tab- or comma-delimited flat file (`.txt`, `.tab`, or `.csv`).

Note: Wallstreet recommends you use the XML formats over the flat file formats because the XML formats are newer and offer more advanced features.

In addition to the Wallstreet standard formats, CMM supports industry standard formats from organizations such as EDIFACT and SWIFT.

In this appendix, Wallstreet standard format are prefixed with "Wallstreet" and industry standard formats are prefixed by the names of their organizations (for example, "EDIFACT" and "SWIFT").

Note: You can create custom formats in addition to these standard ones. For more information, see Chapter 7 Using XML-template-based formats on page 119.

A.1 Summary of standard banking formats

The table below summarizes the supported banking formats.

Type	Wallstreet formats	EDIFACT formats	SWIFT formats	Other formats
Forecast	• XML forecast • Flat file forecast	N/A	N/A	N/A
Accounts payable	• XML transaction • Flat file accounts payable	• BANSTA [1] • PAYEXT	N/A	• HTML [1] • RA01 [1]
Accounts receivable	• XML transaction • Flat file accounts receivable	N/A	N/A	N/A
Direct debit	• XML transaction • Flat file direct debit	N/A	N/A	N/A
Bank message	• XML bank message • XML transaction acknowledgement	• BANSTA [1] • CONTRL [1]	• ACK/NACK [1] • ISO 20022 XML payment status	• ANSI X12 824 [1] • ANSI X12 997 [1] • CFONB/AFB REJ240 [1]
Bank statement	• XML bank statement	N/A	N/A	N/A

Type	Wallstreet formats	EDIFACT formats	SWIFT formats	Other formats
Bank transaction and balance	• XML bank transaction and balance	• FINSTA [1]	• MT940 • MT940 BCS • MT942 [1] • MT950 [1]	• BAI [1] • CFONB/AFB RDC120 [1]
Deal	• Flat file deal	N/A	N/A	N/A
Payment	N/A	• PAYEXT [1] • PAYMUL [1]	• MT100 [1] • MT101 • MT103 [1] • MT202 [1] • ISO 20022 XML credit transfer	• ANSI X12 820 [1] • BACS [1] • CFONB/AFB VIR106 [1]
Direct debit	N/A	• DIRDEB [1]	• SWIFT MT104 • ISO 20022 XML directdebit transfer	• CFONB/AFB PRE160 [1]
Letter of credit	N/A	N/A	N/A	• CFONB/AFB LCR240 [1]
Pre-advice	N/A	N/A	• SWIFT MT210	N/A
Transaction message	• XML payment message • XML receipt message	• BANSTA [1]	N/A	• HTML [1]
Deal	• XML general ledger posting	N/A	N/A	N/A
Free format	N/A	N/A	• MT199 • MT999	N/A

A.2 Forecast import formats

CMM supports the following standard formats for forecast imports:

Type	Wallstreet formats	EDIFACT formats	SWIFT formats	Other formats
Forecast	• XML forecast • Flat file forecast	N/A	N/A	N/A

A.2.1 Wallstreet XML forecast

The following is an example Wallstreet XML forecast file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<transactions document_reference_id="107" format_code="ATTRS_TXN"
xmlns="http://www.trema.com/externalinterface/XMLschema">
    <transaction sequence="1" type="forecast">
       <attribute name="payment_method" value="EFT"/>
       <attribute name="currency" value="USD"/>
       <attribute name="amount" value="50000.00"/>
       <attribute name="value_date" value="20-Oct-2006"/>
```

```
        <attribute name="description" value="Invoice 67003"/>
        <attribute name="party_id" value="AcmeUS"/>
        <attribute name="cpty_id" value="SmithCo"/>
    </transaction>
    <transaction sequence="2" type="forecast">
        <attribute name="payment_method" value="EFT"/>
        <attribute name="currency" value="USD"/>
        <attribute name="amount" value="60000.00"/>
        <attribute name="value_date" value="21-Oct-2006"/>
        <attribute name="description" value="Invoice 67004"/>
        <attribute name="party_id" value="AcmeUS"/>
        <attribute name="cpty_id" value="SmithCo"/>
    </transaction>
</transactions>
```

This file contains three basic elements:

Element	Description
`transactions`	The file's identifier, format code, and schema.
`transaction`	An individual transaction in the file.
`attribute`	An individual attribute of a transaction.

Note: The elements must be nested as displayed in the above table, and the file must validate against the Wallstreet XML transaction schema. For more information on the Wallstreet XML transaction schema, see A.11 Wallstreet XML schemas on page 292.

A.2.1.1 transactions element

A Wallstreet XML forecast file contains only one `transactions` element.

The `transactions` element defines the file's identifier, format code, and schema as specified in the element's three attributes:

Attribute	Acceptable values
`document_reference_ID`	[A unique identifier for the file]
`format_code`	`ATTRS_TXN`
`xmlns`	`http://www.trema.com/externalinterface/XMLschema`

A.2.1.2 transaction elements

A Wallstreet XML forecast file can contain multiple transactions with each transaction represented by a `transaction` element.

Each `transaction` element defines its transaction's sequence number and type as specified in the element's two attributes:

Attribute	Acceptable values
`sequence`	[A unique sequence number for the transaction (In a typical file, the first transaction is numbered `1`; the second, `2`; the third, `3`; and so on.)]
`type`	`forecast`

A.2.1.3 attribute elements

Each `transaction` element contains a set of `attribute` elements. These elements define the transactions' attributes and the values of those attributes.

Each `attribute` element has two attributes:

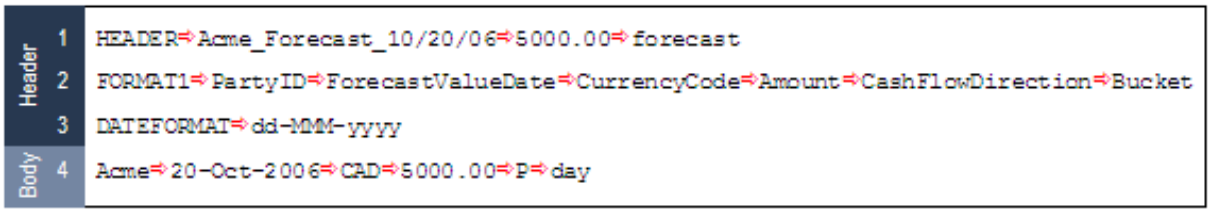
Attribute	Acceptable values
`name`	[The attribute's name as specified in B.1 Import attributes on page 312 (The value you provide in this attribute must exactly match the value in the "Name" column in B.1 Import attributes on page 312.)]
`value`	[The attribute's value (The value you provide in this attribute must comply with the type, size, and other requirements specified in B.1 Import attributes on page 312.)]

As noted in B.1 Import attributes on page 312, some attributes are required and must be provided with every transaction while others are not required.

You can specify the attributes in any order in the Wallstreet XML forecast files. In addition, you do not need to specify the same set of attributes in every transaction in a file. For example, one transaction can include the `description` attribute while all others in the file do not.

A.2.2 Wallstreet flat file forecast

The following is an example forecast flat file (with tabs as the delimiters):



This file contains four lines grouped into two sections:

Line	Name	Description
Header		
1	File information	General information on the file.
2	File layout	The columns included in the file (in the order in which they display in line 4).
3	Date format	The date format used in the file. Note: This line is optional.
Body		
4	Forecast details	The individual forecasts. Note: Include one line 4 for each forecast in the file.

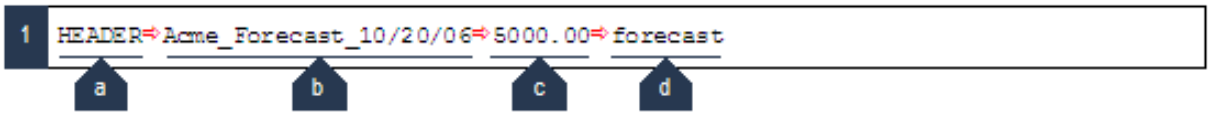
A.2.2.1 Wallstreet flat file forecast header

The header section of a forecast flat file consists of three lines:

- File information (line 1)
- File layout (line 2)
- Date format (line 3).

A.2.2.1.1 File information (line 1)

Line 1 contains general information on the file:



The following table defines the components of this line:

No.	Component	Description	Value	Constraints	Req.	Source
1a	File information indicator	An indicator that the line contains general file information.	HEADER	• None	Yes	User
1b	File ID	A unique identifier for the file. Note: CMMchecks the value of this component before loading the file to prevent accidentally reimporting the same file twice.	Text (50 characters maximum)	• None	Yes	User
1c	Control total	The sum of all forecast amounts in the file. This is a positive number, based on summing the absolute values of the amounts from each record. Note: If you supply a value (which must be positive), CMM does not sum the amounts of all forecasts in the file to confirm completeness before importing.	Positive floating point number	• None	No	User/ System
1d	Type indicator	The file type indicator.	AP	• None	Yes	User

A.2.2.1.2 File layout (line 2)

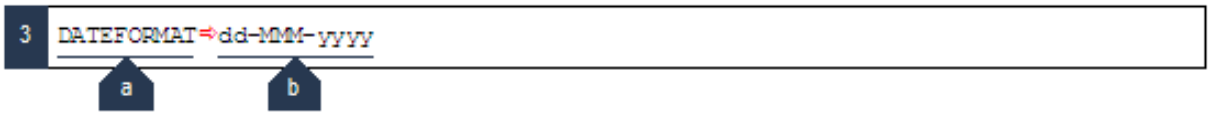
Line 2 defines columns included in the file:

The following table defines the components of this line:

No.	Component	Description	Value	Constraints	Req.	Source
2a	File layout indicator	An indicator that the line contains file layout information.	FORMAT1	• None	Yes	User
2b	Columns	A list (tab- or comma-delimited) of all of the columns that appear in the file. This list identifies: • The columns you are providing in the import file • The order of the columns. The valid columns are documented in A.2.2.2 Wallstreet flat file forecast body on page 138.	Text	• None	Yes	User

A.2.2.1.3 Date format (line 3)

Line 3 defines the date format used in the file:



The following table defines the components of this line:

No.	Component	Description	Value	Constraints	Req.	Source
3a	Date format indicator	An indicator that the line contains date format information.	`DATEFORMAT`	• None	Yes	User
3b	Data format	The date format used when parsing the file. The value of this component indicates the following: • The order of the date component fields (year, month, day, and so on) • Any delimiters or separators between the fields. Note: If you do not specify a value, CMM uses the default format (`dd-MMM-yyyy`).	Text	• None	Yes	User

A.2.2.2 Wallstreet flat file forecast body

The body section of a forecast flat file consists of one line:

- Forecast details (line 4).

A.2.2.2.1 Forecast details (line 4)

Line 4 contains individual forecasts. (Include one line 4 for each forecast in the file.)

The column values in line 4 must display in the same order as the columns in line 2 and must be delimited with tabs or commas.

The following are columns supported by the Wallstreet standard forecast flat file format:

Column	Description	Value	Constraints	Req.	Source
Basic					
`CurrencyCode`	The forecast's currency.	Currency ID	• None	Yes	User
`ForecastAmountInput`	The forecast's amount (in the forecast's currency).	Floating point number	• None	Yes	User
`ForecastItemTypeCode`	The forecast's type.	One of the following values: • `General` • `Big Ticket Item`	• None	No	User

Column	Description	Value	Constraints	Req.	Source
Bucket	The forecast's time bucket, which indicates whether the forecast is short-term or long-term.	One of the following values: • day • month	• None	Yes	User
CashFlowDirection	A value that indicates whether the forecast is a payment or a receipt.	One of the following values: • P if the forecast is a payment • R if the forecast is a receipt	• None	Yes	User
InstrumentTypeID	The forecast's cash flow type. By assigning cash flow types to forecasts and other activity in CMM, you can categorize this activity for a variety of purposes. In particular, you can run several reports in the module that display activity by cash flow type. Note: In CMM, a cash flow type is an instrument type with the **Status** attribute set to Enabled.	Instrument type ID	• None	No	User
PaymentMethodID	The forecast's payment method. A forecast's payment method defines how the forecast is to be paid (for example, by EFT or by check) if it becomes a releasable transaction.	Payment method ID	• None	No	User

Column	Description	Value	Constraints	Req.	Source
ForecastValueDate	The forecast's starting value date. Note: Medium- and long-term forecasts can span a range of value dates (for example, a medium-term forecast can span a month), while short-term forecasts usually only span one value date.	Date	• None	Yes	User
InvoiceDate	The forecast's invoice date.	Date	• None	No	User
WorkflowAction	The action CMM should take upon importing the forecast. Note: If you do not specify a value in this column, CMM creates a new record for the forecast upon importing it.	One of the following values: • edit • cancel	• None	No	User
Description	A relevant description of or other information pertaining to the forecast.	Text (2,000 characters maximum)	• None	No	User
Party					
PartyID	The forecast's party.	Entity ID	• None	Yes	User
Party bank					
BankID	The bank where the forecast's party bank account is held.	In-house bank ID External bank ID	• Must be provided if the party bank account number is also provided	No	User
Party bank account					

Column	Description	Value	Constraints	Req.	Source
BankAccountNumber	The forecast's party bank account. It is not usually necessary to define a party bank account for a forecast. However, this attribute is provided for situations in which you need to define a party bank account for a short-term forecast (or "cash position"). Note: Do not include format characters (_, -, and so on) in this column.	Entity bank account number	• None	No	System
BankAccountCurrencyCode	The curency of the forecast's party bank account.	Currency ID	• Must be provided if the party bank account number is also provided	No	User
Counterparty					
CptyID	The forecast's counterparty. It is not usually necessary to define a counterparty for a forecast. However, this attribute is provided for situations in which you need to define a counterparty for a short-term forecast (or "cash position").	Entity ID Counterparty ID	• None		
Counterparty bank					
CounterpartyBankAccountBankID	The ID of the bank where the forecast's counterparty bank account is held.	In-house bank ID External bank ID	• Must be provided if the counterparty bank account number is also provided	No	User
Counterparty bank account					

Column	Description	Value	Constraints	Req.	Source
CounterpartyBankAccountNumber	The bank account number of the forecast's counterparty bank account. Note: Do not include format characters (_, -, and so on) in this column.	Entity bank account number Counterparty bank account number	• None	No	User
CounterpartyBankAccountCurrencyCode	The ISO currency code of the forecast's counterparty bank account.	Currency ID	• Must be provided if the counterparty bank account number is also provided	No	User
Other					
BusinessSegmentID	The forecast's business segment. A business segment is a business-based division of your organization used for reporting purposes.	Business segment ID	• None	No	User
RegionID	The forecast's region. A region is a geographic-based division of your organization for reporting purposes.	Region ID	• None	No	User
SourceReferenceTextID	The forecast's source reference text ID.	Text (50 characters maximum)	• None	No	User
StringAttributeValue_0 StringAttributeValue_1 StringAttributeValue_2 StringAttributeValue_3 StringAttributeValue_4 StringAttributeValue_5 StringAttributeValue_6 StringAttributeValue_7 StringAttributeValue_8 StringAttributeValue_9	The forecast's custom attributes. Note: You or another user in your organization can define custom attributes (or "parameters") for forecasts in the Parameter Editor function.	Text (50 characters maximum per field)	• None	No	User

A.3 Transaction import formats

CMM supports the following standard formats for transaction imports:

Type	Wallstreet formats	EDIFACT formats	SWIFT formats	Other formats
Accounts payable	• XML transaction • Flat file accounts payable	• BANSTA [1] • PAYEXT	N/A	• HTML [1] • RA01 [1]
Accounts receivable	• XML transaction • Flat file accounts receivable	N/A	N/A	N/A
Direct debit	• XML transaction • Flat file direct debit	N/A	N/A	N/A
Table notes: 1. For information on formats not documented in this appendix, contact Wallstreet.				

A.3.1 Wallstreet XML transaction

The following is an example Wallstreet XML transaction file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<transactions document_reference_id="107" format_code="ATTRS_TXN"
xmlns="http://www.trema.com/externalinterface/XMLschema">
    <transaction sequence="1" type="aprecord">
       <attribute name="payment_method" value="EFT"/>
       <attribute name="currency" value="USD"/>
       <attribute name="amount" value="50000.00"/>
       <attribute name="txn_date" value="20-Oct-2006"/>
       <attribute name="description" value="Invoice 67003"/>
       <attribute name="customer_reference_id" value="123456"/>
       <attribute name="party_id" value="AcmeUS"/>
       <attribute name="cpty_id" value="SmithCo"/>
    </transaction>
    <transaction sequence="2" type="aprecord">
       <attribute name="payment_method" value="EFT"/>
       <attribute name="currency" value="USD"/>
       <attribute name="amount" value="25000.00"/>
       <attribute name="txn_date" value="20-Oct-2006"/>
       <attribute name="description" value="Invoice 67004"/>
       <attribute name="customer_reference_id" value="123457"/>
       <attribute name="party_id" value="AcmeUS"/>
       <attribute name="cpty_id" value="WilliamsCo"/>
       <attribute composite="true" name="remittance_details">
          <component_transaction type="remittancedetail">
             <attribute name="remittance_date" value="20-Oct-2006"/>
             <attribute name="remittance_detail_message_type" value="100"/>
             <attribute name="remittance_retail_reference_type_code" value=""/>
             <attribute name="remittance_invoice_number" value="invoice_1"/>
             <attribute name="remittance_detail_reference" value=""/>
             <attribute name="remittance_currency_code" value="USD"/>
             <attribute name="document_amount" value="25000.00"/>
             <attribute name="document_amount_type" value="204"/>
          </component_transaction>
          <component_transaction type="remittancedetail">
             <attribute name="remittance_date" value="20-Oct-2006"/>
             <attribute name="remittance_detail_message_type" value="100"/>
```

```
            <attribute name="remittance_retail_reference_type_code" value=""/>
            <attribute name="remittance_invoice_number" value="invoice_1"/>
            <attribute name="remittance_detail_reference" value=""/>
            <attribute name="remittance_currency_code" value="USD"/>
            <attribute name="document_amount" value="25000.00"/>
            <attribute name="document_amount_type" value="203"/>
          </component_transaction>
        </attribute>
    </transaction>
</transactions>
```

This file contains four basic elements:

Element	Description
`transactions`	The file's identifier, format code, and schema.
`transaction`	An individual transaction in the file.
`attribute`	An individual attribute of a transaction.
`component_transaction`	An individual remittance detail in a transaction. Note: This element is only applicable to the accounts payable and direct debit formats.
`attribute`	An individual attribute of a remittance detail.

Note: The elements must be nested as displayed in the above table, and the file must validate against the Wallstreet XML transaction schema. For more information on the Wallstreet XML transaction schema, see A.11 Wallstreet XML schemas on page 292.

A.3.1.1 transactions element

A Wallstreet XML transaction file contains only one `transactions` element.

The `transactions` element defines the file's identifier, format code, and schema as specified in the element's three attributes:

Attribute	Acceptable values
`document_reference_ID`	[A unique identifier for the file]
`format_code`	`ATTRS_TXN`
`xmlns`	`http://www.trema.com/externalinterface/XMLschema`

A.3.1.2 transaction elements

A Wallstreet XML transaction file can contain multiple transactions with each transaction represented by a `transaction` element.

Each `transaction` element defines its transaction's sequence number and type as specified in the element's two attributes:

Attribute	Acceptable values
`sequence`	[A unique sequence number for the transaction (In a typical file, the first transaction is numbered `1`; the second, `2`; the third, `3`; and so on.)]

`type`	`aprecord`
	`arrecord`
	`ddrecord`

A.3.1.3 attribute elements

Each `transaction` element contains a set of `attribute` elements. These elements define the transactions' attributes and the values of those attributes.

There are two types of `attribute` elements:

Type	Available in
`attribute` elements that do not contain `component_transaction` elements	• Accounts payable • Accounts receivable • Direct debit
`attribute` elements that contain `component_transaction` elements	• Accounts payable • Direct debit

Each `attribute` element of the first type has two attributes:

Attribute	Acceptable values
`name`	[The attribute's name as specified in B.1 Import attributes on page 312 (The value you provide in this attribute must exactly match the value in the "Name" column in B.1 Import attributes on page 312.)]
`value`	[The attribute's value (The value you provide in this attribute must comply with the type, size, and other requirements specified in B.1 Import attributes on page 312.)]

Each `attribute` element of the second type has two attributes:

Attribute	Acceptable values
`composite`	`true`
`name`	`remittance_detail`

As noted in B.1 Import attributes on page 312, some attributes are required and must be provided with every transaction while others are not required.

You can specify the attributes in any order in the Wallstreet XML transaction files. In addition, you do not need to specify the same set of attributes in every transaction in a file. For example, one transaction can include the `description` attribute while all others in the file do not.

A.3.1.4 component_transaction elements

In a Wallstreet XML accounts payable or direct debit file, each transaction can contain multiple remittance details with each remittance detail represented by a `component_transaction` element.

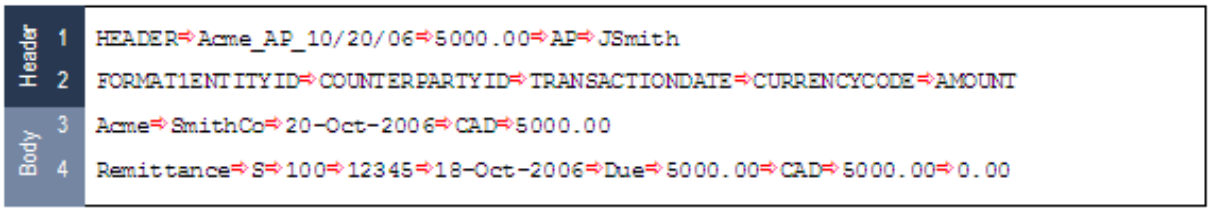
Note: `component_transaction` elements must be nested inside an appropriate `attribute` element of their parent transaction.

Each `component_transaction` element defines its remittance detail's type as specified in the element's attribute:

Attribute	Acceptable values
`type`	`remittancedetail`

A.3.2 Wallstreet flat file accounts payable

The following is an example accounts payable flat file (with tabs as the delimiters):



This file contains four lines grouped into two sections:

Line	Name	Description
Header		
1	File information	General information on the file.
2	File layout	The columns included in the file (in the order in which they display in line 3).
Body		
3	Transaction details	The individual transactions. Note: Include one line 3 for each transaction in the file.
4	Remittance details	The individual remittance details. Note: Include one line 4 for each remittance detail in the file.

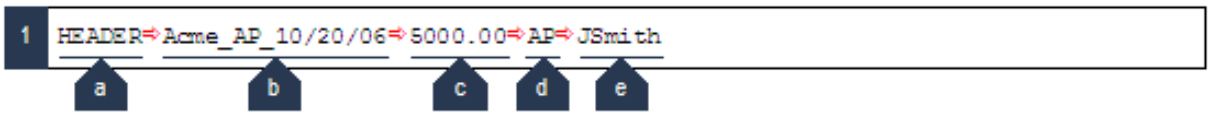
A.3.2.1 Wallstreet flat file accounts payable header

The header section of an accounts payable flat file consists of two lines:

- File information (line 1)
- File layout (line 2).

A.3.2.1.1 File information (line 1)

Line 1 contains general information on the file:



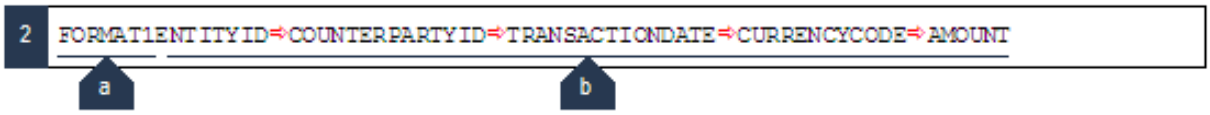
The following table defines the components of this line:

No.	Component	Description	Value	Constraints	Req.	Source
1a	File information indicator	An indicator that the line contains general file information.	HEADER	• None	Yes	User
1b	File ID	A unique identifier for the file. Note: CMMchecks the value of this component before loading the file to prevent accidentally reimporting the same file twice.	Text (50 characters maximum)	• None	Yes	User

No.	Component	Description	Value	Constraints	Req.	Source
1c	Control total	The sum of all transaction amounts in the file. This is a positive number, based on summing the absolute values of the amounts from each record. Note: If you supply a value (which must be positive), CMM does not sum the amounts of all transactions in the file to confirm completeness before importing.	Positive floating point number	• None	No	User/ System
1d	Type indicator	The file type indicator.	AP	• None	Yes	User
1e	User ID	Your user ID.	Text	• None	No	User

A.3.2.1.2 File layout (line 2)

Line 2 defines columns included in the file:



The following table defines the components of this line:

No.	Component	Description	Value	Constraints	Req.	Source
2a	File layout indicator	An indicator that the line contains file layout information.	FORMAT1	• None	Yes	User
2b	Columns	A list (tab- or comma-delimited) of all of the columns that appear in the file. This list identifies: • The columns you are providing in the import file • The order of the columns. The valid columns are documented in A.3.2.2 Wallstreet flat file accounts payable body on page 147.	Text	• None	Yes	User

Note: Do not place a tab, comma, space, or other character between the file layout indicator (component 2a) and the columns (component 2b).

A.3.2.2 Wallstreet flat file accounts payable body

The body section of an accounts payable flat file consists of two lines:

- Transaction details (line 3)
- Remittance details (line 4).

A.3.2.2.1 Transaction details (line 3)

Line 3 contains the individual transactions. (Include one line 3 for each transaction in the file.)

The column values in line 3 must display in the same order as the columns in line 2 and must be delimited with tabs or commas.

The following are columns supported by the Wallstreet standard accounts payable flat file format:

Column	Description	Value	Constraints	Req.
Basic				
CURRENCYCODE	The transaction's currency. (String of three alphabetical characters in capital letters.)	Currency ID	Must be a valid ISO currency code	Yes
AMOUNT	The transaction's absolute value (in the transaction's currency). Maximum size is 1E15.	Floating point number	None	Yes
PAYMENTSTATUSCODE	A value that indicates whether the transaction is actual activity or forecasted activity. Note: CMM 7.1 introduced forecasts as a replacement for forecasted transactions.	One of the following: • Actual if the transaction is actual activity • Forecast if the transaction is forecasted activity Note: If you do not specify a value in this column, system sets it to A.	None	Yes
PRIORITYID	The transaction's priority.	One of the following values: • 1=NON-URGENT if the transaction is not urgent • 2=URGENT if the transaction is urgent Note: If you do not specify a value in this column, the system sets it to 1 (non-urgent)	None	No
CASHFLOWTYPE	The transaction's cash flow type. By assigning cash flow types to transactions and other activity in CMM, you can categorize this activity for a variety of purposes. In particular, you can run several reports in the module that display activity by cash flow type. Note: In CMM, a cash flow type is an instrument type with the **Status** attribute set to Enabled.	Instrument type ID Note: If you do not specify a value in this column, CMM sets it to COMMP (Commercial Payment) regardless of the cash flow type default value set in User Options.	Must be a valid cash flow type ID. Must be included in the selected report mapping	• No

Column	Description	Value	Constraints	Req.
PAYMENTMETHOD	The transaction's payment method. A transaction's payment method defines how the transaction is to be paid when it is released (for example, by EFT or by check).	Payment method ID Default value is EFT regarless of the cash flow type default value set in User Options. Note that, based on the chosen value, counterparty bank account details might be required. (Check for example does not require any counterparty account details)	None	• No
TRANSACTIONDATE	The transaction's transaction date.	Date Note: If you do not specify a value in this column, CMM sets it to the current system date.	None	No
VALUEDATE	The transaction's value date.	Date Note: If you do not specify a value in this column, CMM sets it to the same value as TRANSACTIONDATE	None	No
CREATIONDATE	The transaction's creation date.	Date Note: If you do not specify a value in this column, CMM sets it to the same value as TRANSACTIONDATE	None	No
DESCRIPTION/REFERENCE	A relevant description of or other information pertaining to the transaction.	Text (2,000 characters maximum)	None	No
Party				
ENTITYID	The transaction's party.	Entity ID (25 characters maximum)	Must be a valid entity ID or mapped in "Entity Codes Mapping Table Maintenance"	Yes

Column	Description	Value	Constraints	Req.
SUPPLYPAYINGACCOUNT	A value that indicates whether you are supplying party bank account details.	One of the following values: • Y if you are supplying party bank account details • N if you are not supplying party bank account details Note: N is the default if you do not specify a value or if you do not provide this column.	None	No
Party bank				
ENTITYBANKBRANCHID [1]	The branch ID of the transaction's party bank (as assigned by an agency).	Text (15 characters maximum)	None	No
Party bank account				
ENTITYBANKACCOUNTNUMBER [1]	The transaction's party bank account. Note: If you do not provide a party bank account, CMM can select one using routing rules.	Entity bank account primary number See Entity bank account checks on page 156.	Must be held by the transaction's party	No
BANKACCOUNTENTITYID [1]	The owner of the transaction's party bank account.	Entity ID See Entity bank account checks on page 156.	When provided, it has to be equal or mapped with ENTITYID.	No
ENTITYBANKACCOUNTCURRENCYCODE [1]	The currency of the transaction's party bank account.	Currency ID See Entity bank account checks on page 156.	None	No
Counterparty				
COUNTERPARTYID	The transaction's counterparty.	Counterparty ID (25 characters maximum)	Must be a valid party ID.	If you select a counterparty and a counterparty type in the criteria, but the counterparty type is not assigned to the counterparty, the report contains no results

Column	Description	Value	Constraints	Req.
SUPPLYCPTYDETAILS	A value that indicates whether you are supplying counterparty details.	One of the following values: • Y if you are supplying counterparty details • N if you are not supplying counterparty details Note: N is the default if you do not specify a value or if you do not provide this column.	None	No
COUNTERPARTYNAME [2]	The name of the transaction's counterparty.	Text (35 characters maximum)	None	No
COUNTERPARTYADDRESS1 COUNTERPARTYADDRESS2 COUNTERPARTYADDRESS3 COUNTERPARTYADDRESS4 [2]	The address of the transaction's counterparty.	Text (40 characters maximum per field)	None	No
COUNTERPARTYCITY [2]	The city of the transaction's counterparty.	Text (50 characters maximum)	None	No
COUNTERPARTYSTATE [2]	The state of the transaction's counterparty.	State ID (20 characters maximum)	None	No
COUNTERPARTYCOUNTRYCODE [2]	The country of the transaction's counterparty.	Country ID (two-character ISO country code)	None	No
COUNTERPARTYZIP [2]	The postal or ZIP code of the transaction's counterparty.	Text (10 characters maximum)	None	No
Counterparty bank				
COUNTERPARTYBANKCODE [2]	The SWIFT or ABA code of the transaction's counterparty bank. Considered a branch Sort Code if BANKCODETYPE is A or ABA, and then checked against country code rules (See Country Code Maintenance).	Text (20 characters maximum)	None	No
COUNTERPARTYBANKCODETYPE [2]	The SWIFT or ABA code type of transaction's counterparty bank.	One of the following: • S if the code type is SWIFT • A if the code type is ABA	None	No

Column	Description	Value	Constraints	Req.
COUNTERPARTYBANKBRANCHID [2]	The branch ID of the transaction's counterparty bank (as assigned by an agency). Considered a branch Sort Code if BANKCODETYPE is empty and then checked against country code rules(See Country Code Maintenance).	Text (15 characters maximum)	None	No
COUNTERPARTYBANKNAME	The name of the transaction's counterparty bank.	Text (35 characters maximum)	None	No
COUNTERPARTYBANKADDRESS COUNTERPARTYBANKADDRESS2	The address of the transaction's counterparty bank.	Text (40 characters maximum per field)	None	No
COUNTERPARTYBANKCITY	The city of the transaction's counterparty bank.	Text (50 characters maximum)	None	No
COUNTERPARTYBANKSTATE	The state of the transaction's counterparty bank.	State ID (20 characters maximum)	None	No
COUNTERPARTYBANKZIP	The postal or ZIP code of the transaction's counterparty bank.	Text (10 characters maximum)	None	No
Counterparty bank account				
COUNTERPARTYBANKACCOUNTNUMBER [2]	The primary number (usually, BBAN) of the transaction's counterparty bank account.	Text (35 characters maximum)	Must validate against the BBAN rules of the bank account's country. See Entity bank account checks on page 156.	No
CPTYPRIMARYACCOUNTNUMBERTYPE [2]	The primary number type of the transaction's counterparty bank account.	BBAN	None See Entity bank account checks on page 156.	No
CPTYSECONDARYACCOUNTNUMBER [2]	The secondary number (usually IBAN) of the transaction's counterparty bank account.	Text (34 characters maximum)	Must validate against the IBAN rules See Entity bank account checks on page 156.	No
CPTYSECONDARYACCOUNTNUMBERTYPE [2]	The secondary number type of the transaction's counterparty bank account.	IBAN	None See Entity bank account checks on page 156.	No

Column	Description	Value	Constraints	Req.
COUNTERPARTYBANKACCOUNTCOUNTRYCODE	The country of the transaction's counterparty bank.	Country ID (ISO country code on 2 characters)	None See Entity bank account checks on page 156.	No
COUNTERPARTYBANKCCOUNTCURRENCYCODE	The currency of the transaction's counterparty bank account.	Currency ID	None See Entity bank account checks on page 156.	No
Intermediary bank				
INTERMEDIARYBANKCODE	The SWIFT or ABA code of the transaction's intermediary bank.	Text (20 characters maximum)	None	No
INTERMEDIARYBANKCODETYPE	The SWIFT or ABA code type of transaction's intermediary bank.	One of the following values: • S if the code type is SWIFT • A if the code type is ABA	None	No
INTERMEDIARYBANKNAME	The name of the transaction's intermediary bank.	Text (40 characters maximum)	None	No
INTERMEDIARYBANKADDRESS INTERMEDIARYBANKADDRESS2	The address of the transaction's intermediary bank.	Text (40 characters maximum per field)	None	No
INTERMEDIARYCITY	The city of the transaction's intermediary bank.	Text (40 characters maximum)	None	No
INTERMEDIARYSTATE	The state of the transaction's intermediary bank.	State ID (20 characters maximum)	None	No
INTERMEDIARYCOUNTRYCODE	The country of the transaction's intermediary bank.	Country ID (Two-character ISO country code)	None	No
INTERMEDIARYZIP	The postal or ZIP code of the transaction's intermediary bank.	Text (40 characters maximum)	None	No
Intermediary bank account				
INTERMEDIARYBANKACCOUNTNUMBER	The primary number (usually BBAN) of the transaction's intermediary bank account.	Text (40 characters maximum)	Must validate against the BBAN rules of the bank account's country	No
Secondary intermediary bank				
SECONDINTERMEDIARYBANKCODE	The SWIFT or ABA code of the transaction's intermediary bank.	Text (20 characters maximum)	None	No

Column	Description	Value	Constraints	Req.
SECONDINTERMEDIARYBANKCODETYPE	The SWIFT or ABA code type of transaction's intermediary bank.	One of the following values: • S if the code type is SWIFT • A if the code type is ABA	None	No
SECONDINTERMEDIARYBANKNAME	The name of the transaction's intermediary bank.	Text (40 characters maximum)	None	No
SECONDINTERMEDIARYBANKADDRESS SECONDINTERMEDIARYBANKADDRESS2	The address of the transaction's intermediary bank.	Text (40 characters maximum per field)	None	No
SECONDINTERMEDIARYCITY	The city of the transaction's intermediary bank.	Text (40 characters maximum)	None	No
SECONDINTERMEDIARYSTATE	The state of the transaction's intermediary bank.	State ID (20 characters maximum)	None	No
SECONDINTERMEDIARYCOUNTRYCODE	The country of the transaction's intermediary bank.	Country ID (Two-character ISO country code)	None	No
SECONDINTERMEDIARYZIP	The postal or ZIP code of the transaction's intermediary bank.	Text (40 characters maximum)	None	No
Second intermediary bank account				
SECONDINTERMEDIARYBANKACCOUNTNUMBER	The primary number (usually BBAN) of the transaction's intermediary bank account.	Text (40 characters maximum)	Must validate against the BBAN rules of the bank account's country	No
Aggregation				
AGGREGATEDETAILCODE	A value that indicates whether the transaction is an aggregate or detail.	One of the following values: • A if the transaction is an aggregate • D if the transaction is a detail	None	No
AGGREGATEREFERENCEID	If the transaction is an aggregate, the transaction's unique reference number. If the transaction is a detail, the reference number of the aggregate transaction in which the detail transaction is contained.	Text (50 characters maximum)	None	No

Column	Description	Value	Constraints	Req.
Tax				
TAXPAYMENTTYPECODE	A code identifying the type of tax being paid.	Text (255 characters maximum)	None	No
TAXPAYMENTSUBTYPECODE	A code identifying the subtype of tax being paid.	Text (255 characters maximum)	None	No
TAXPAYERID	A number assigned to a purchaser by a taxing jurisdiction. Note: This ID is also known as a tax exemption number or certificate number.	Text (255 characters maximum)	None	No
TAXPAYERVERIFICATIONID	An ID the receiver can use to confirm the taxpayer's identity.	Text (255 characters maximum)	None	No
Regulatory reporting				
REGULATORYCODE	The transaction's regulatory code.	Regulatory code	None	No
REGULATORYQUALIFIER	The qualifier of the transaction's regulatory code.	Text (50 characters maximum)	None	No
REGULATORYAGENCY	The agency of the transaction's regulatory code.	Text (50 characters maximum)	Must be provided if the transaction's regulatory code is provided	No
REGULATORYDESCRIPTION	A description of the transaction's regulatory code.	Text (50 characters maximum)	None	No
Other				
BANKINSTRUCTIONS	Instructions or other messages to the bank.	Text (100 characters maximum)	None	No
CHEQUENUMBER	The transaction's check number (if the transaction is paid by check).	Text (35 characters maximum)	None	No
COUNTERPARTYMESSAGE	Instructions or other messages to the transaction's counterparty.	Text (255 characters maximum)	None	No
CUSTOMERBATCHREFERENCEID	The customer's batch reference ID for the transaction.	Text (30 characters maximum)	None	No
CUSTOMERREFERENCENUMBER	The transaction's customer reference ID.	Text (50 characters maximum)	None	No

Column	Description	Value	Constraints	Req.
CUSTOMERTRANSACTIONREFERENCEID	The customer's reference ID for the transaction. Note: This ID does not have to be unique.	Text (30 characters maximum)	None	No
FINANCIALCHARGEALLOCATION	The financial allocation charge of the transaction.	One of the following values: • 1 if the financial charges are being shared equally between the sender and receiver • 2 if the financial charges are being paid entirely by the sender • 3 if the financial charges are being paid entirely by the receiver	None	No
HANDLINGINSTRUCTIONCODE	The ID of the transaction's handling instructions.	Text (5 characters maximum)	None	No
PERSONALIDENTIFICATIONNUMBER	The personal identification number of the transaction.	Text (255 characters maximum)	None	No
RETRIEVALLOCATIONID	The ID of the transaction's pickup location. Note: This attribute is used in eastern Europe.	Text (255 characters maximum)	None	No
TRANSACTIONTYPECODE	The ID of the transaction's type. Note: This is country and bank specific.	Text (10 characters maximum)	None	No
Table notes: Only supply a value in this column if you have set SUPPLYPAYINGACCOUNT to Y. Only supply a value in this column if you have set SUPPLYCPTYDETAILS to Y.				

Entity bank account checks

The system searches for a valid Bank Account using:

- ENTITYID
- ENTITYBANKACCOUNTNUMBER
- ENTITYBANKACCOUNTCURRENCYCODE
- ENTITYBANKBRANCHID

If the currency of the bank account (ENTITYBANKACCOUNTCURRENCYCODE) is not provided, CMM uses the transaction currency (CURRENCYCODE) instead for the search.

This can be disabled by setting the configuration parameter 'Default bank account currency to transaction currency during routing' to False.

If Bank Sort Code (ENTITYBANKBRANCHID) is not provided, the system ignores it in the search.

If it is provided, this code could be ignored by setting the configuration parameter 'Use Bank Branch ID (sort code) during routing' to False.

The search is:

- Done first with the bank account ENTITYBANKACCOUNTNUMBER as BBAN (Primary Account Number),
- Then, should the first search be unsuccessful, a second search is done with the bank account number (ENTITYBANKACCOUNTNUMBER) considered as IBAN (Secondary Account Number).

The search is unsuccessful if none or multiple account(s) are found.

Counterparty bank account checks

During Import (before routing)

If SUPPLYCPTYDETAILS is Y, the system validates the format of primary and secondary counterparty bank account numbers (COUNTERPARTYBANKACCOUNTNUMBER, CPTYSECONDARYACCOUNTNUMBER):

- Both account numbers cannot be identical.
- If both accounts are provided, the primary account must be a BBAN number and the secondary account has to be an IBAN (use CPTYPRIMARYACCOUNTNUMBERTYPE and CPTYSECONDARYACCOUNTNUMBERTYPE to specify the types).
- If the primary account only is provided, the number can be in IBAN or BBAN format. If the format is not specified (COUNTERPARTYBANKACCOUNTNUMBER) or is specified as BBAN (CPTYPRIMARYACCOUNTNUMBERTYPE), the systen still tries to convert it (runs IBAN country specific checks through it) to IBAN before accepting it as BBAN. If the bank account number complies with the country's specific checks, it is considered as IBAN.
- If the secondary account number only is provided (CPTYSECONDARYACCOUNTNUMBER), it must be in IBAN format (set CPTYSECONDARYACCOUNTNUMBERTYPE accordingly).

In all cases, both IBAN and BBAN account formats are validated against country rules using the country code if provided (COUNTERPARTYBANKACCOUNTCOUNTRYCODE).

If the configuration parameter "AllowTransactionsOnClosedCounterpartyAccounts" is set to False, the system checks that the specified account is Open and disallows closed account selection.

The system then tries to find the counterparty bank account using:

- COUNTERPARTYID
  Total specific: If Counterparty (COUNTERPARTYID) is not specified, the system tries to deduce it from the counterparty account numbers (COUNTERPARTYBANKACCOUNTNUMBER and CPTYSECONDARYACCOUNTNUMBER) and the counterparty bank account currency.
- COUNTERPARTYBANKCODE
- COUNTERPARTYBANKACCOUNTNUMBER
- COUNTERPARTYBANKACCOUNTCURRENCYCODE

If Counterparty Bank (COUNTERPARTYBANKCODE) is not provided, it is ignored.

If Counterparty Bank Account Currency Code (COUNTERPARTYBANKACCOUNTCURRENCYCODE) is not provided, the transaction currency is used instead.

If the search is not successful, the process is not interrupted as the Routing process will try to complete the payment using the configurable routing rules. An unsuccessful search means that either the account was not found using the search criteria, or multiple accounts were found.

During routing

To search for a valid counterparty account, the system uses:

- COUNTERPARTYID
- COUNTERPARTYBANKACCOUNTNUMBER
- CPTYPRIMARYACCOUNTNUMBERTYPE
- COUNTERPARTYBANKACCOUNTCURRENCYCODE

If the counterparty bank account bank currency (COUNTERPARTYBANKACCOUNTCURRENCYCODE) is not provided, the system uses the transaction currency instead. This can be disabled by setting the configuration parameter "Default bank account currency to transaction currency during routing" to False.

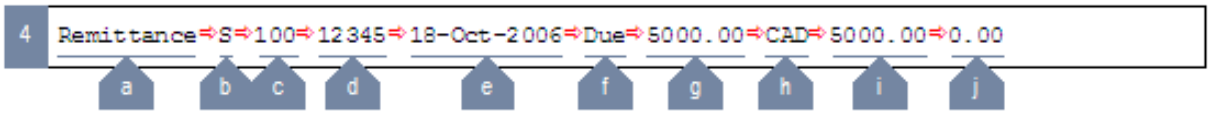
Only Open accounts are used.

Other Checks:

- If payment method (PAYMENTMETHOD) is ITC, Paying Account has to be internal,
- If payment method (PAYMENTMETHOD) is DD and Transaction Type Code (TRANSACTIONTYPECODE) is "Direct Debit", paying and payee country codes must be the same.

A.3.2.2.2 Remittance details (line 4)

Line 4 contains the individual remittance details:



(Include one line 4 for each remittance detail in the file.)

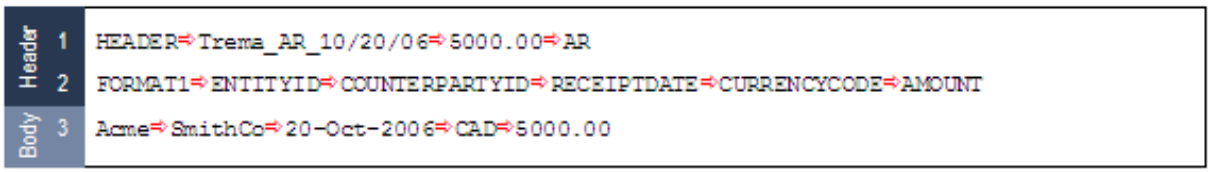
The following table defines the components of this line:

No.	Component	Description	Value	Constraints	Req.	Srce
4a	Remittance indicator	An indicator that the line contains file layout information.	`Remittance`	• None	Yes	
4b	Remittance handling code	A value that indicates how remittance details are handled in the file.	One of the following values: • `S` if the remittance details are handled in a single line • `M` if the remittance details are handled in multiple lines Note: If you enter `M`, provide one remittance line detail for the invoice amount and one for the actual amount paid.	• None	Yes	
4c	Document type code	The remittance detail's type.	One of the following values: • `100` if the remittance detail is an invoice • `101` if the remittance detail is a credit • `103` if the remittance detail is a debit	• None	Yes	

No.	Component	Description	Value	Constraints	Req.	Srce
4d	Document message number	The remittance detail's invoice, credit note, or debit note.	Text (35 characters maximum)	• None	Yes	
4e	Document date	The remittance detail's date.	Date	• None	Yes	
4f	Document amount type	The type of the remittance detail's amount.	One of the following values: • `Due` if the remittance handling code is set to `S` • `201` if the remittance detail's amount is paid • `202` if the remittance detail's amount is remitted • `203` if the remittance detail's amount is discontinued • `204` if the remittance detail's amount is payable	• None	No	
4g	Document Amount	The remittance detail's amount.	Floating point number	• None	Yes	
4h	Document currency code	The remittance detail's currency.	Currency ID	• None	No	
4i	Amount paid	The actual amount paid.	Floating point number	• Can only be provided if the remittance handling code is `Y`.	No	
4j	Amount Discounted	The amount discounted.	Floating point number	• Can only be provided if the remittance handling code is `Y`.	No	

A.3.3 Wallstreet flat file accounts receivable

The following is an example accounts receivable flat file (with tabs as the delimiters):



This file contains three lines grouped into two sections:

Line	Name	Description
Header		
1	File information	General information on the file.

Line	Name	Description
2	File layout	The columns included in the file (in the order in which they display in line 3).
Body		
3	Transaction details	The individual transactions. Note: Include one line 3 for each transaction in the file.

A.3.3.1 Wallstreet flat file accounts receivable

The header section of an accounts receivable flat file consists of two lines:

- File information (line 1)
- File layout (line 2).

A.3.3.1.1 File information (line 1)

Line 1 contains general information on the file:

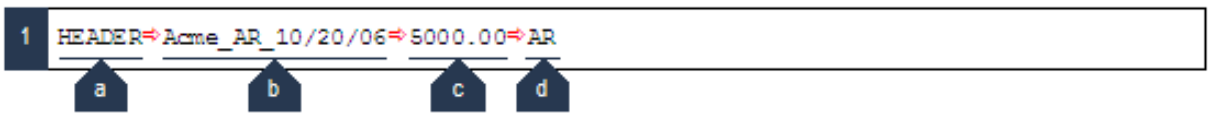
The following table defines the components of this line:

No.	Component	Description	Value	Constraints	Req.	Source
1a	File information indicator	An indicator that the line contains general file information.	`HEADER`	• None	Yes	User
1b	File ID	A unique identifier for the file. Note: CMMchecks the value of this component before loading the file to prevent accidentally reimporting the same file twice.	Text (50 characters maximum)	• None	Yes	User
1c	Control total	The sum of all transaction amounts in the file. Note: If you supply a value, CMM does not sum the amounts of all transactions in the file to confirm completeness before importing.	Floating point number	• None	No	User
1d	Type indicator	The file type indicator.	`AR`	• None	Yes	User

Note: Line 1 is not required for accounts receivable flat files.

A.3.3.1.2 File layout (line 2)

Line 2 defines columns included in the file:



The following table defines the components of this line:

No.	Component	Description	Value	Constraints	Req.	Source
2a	File layout indicator	An indicator that the line contains file layout information.	FORMAT1	• None	Yes	User
2b	Columns	A list (tab- or comma-delimited) of all of the columns that appear in the file. This list identifies: • The columns you are providing in the import file • The order of the columns. The valid columns are documented in A.3.3.2 Wallstreet flat file accounts receivable body on page 161.	Text	• None	Yes	User

A.3.3.2 Wallstreet flat file accounts receivable body

The body section of an accounts receivable flat file consists of one line:

- Transaction details (line 3).

A.3.3.2.1 Transaction details (line 3)

Line 3 contains the individual transactions. (Include one line 3 for each transaction in the file.)

The column values in line 3 must display in the same order as the columns in line 2 and must be delimited with tabs or commas.

The following are columns supported by the Wallstreet standard accounts receivable flat file format:

Column	Description	Value	Constraints	Req.	Srce
Basic					
CURRENCYCODE	The transaction's currency.	Currency ID	• None	Yes	User
AMOUNT	The transaction's absolute value (in the transaction's currency).	Floating point number	• None	Yes	User
RECEIPTSTATUSCODE	A value that indicates whether the transaction is actual activity or forecasted activity. Note: CMM 7.1 introduced forecasts as a replacement for forecasted transactions.	One of the following: • A if the transaction is actual activity • F if the transaction is forecasted activity Note: A is the default if you do not specify a value.	• None	No	User

Column	Description	Value	Constraints	Req.	Srce
CASHFLOWTYPE	The transaction's cash flow type. By assigning cash flow types to transactions and other activity in CMM, you can categorize this activity for a variety of purposes. In particular, you can run several reports in the module that display activity by cash flow type. Note: In CMM, a cash flow type is an instrument type with the **Status** attribute set to Enabled.	Instrument type ID Note: COMMR (Commercial Receipt) is the default if you do not specify a value.	• None	No	User
RECEIPTMETHOD	The transaction's payment method. A transaction's payment method defines how the transaction is to be paid when it is released (for example, by EFT or by check).	Payment method ID	• None	Yes	User
TRANSACTIONDATE	The transaction's transaction date.	Date Note: The current date is the default if you do not specify a value.	• None	No	User
VALUEDATE	The transaction's value date.	Date Note: The transaction date is the default if you do not specify a value.	• None	No	User
DESCRIPTION/REFERENCE	A relevant description of or other information pertaining to the transaction.	Text (2,000 characters maximum)	• None	No	User
Party					
ENTITYID	The transaction's party.	Entity ID	• None	Yes	User

Column	Description	Value	Constraints	Req.	Srce
SUPPLYRECEIVINGACCOUNT	A value that indicates whether you are supplying party bank account details.	One of the following values: • Y if you are supplying party bank account details • N if you are not supplying party bank account details Note: N is the default if you do not specify a value.	• None	No	User
Party bank					
ENTITYBANKBRANCHID [1]	The branch ID of the transaction's party bank (as assigned by an agency).	Text (15 characters maximum)	• None	No	User
Party bank account					
ENTITYBANKACCOUNTNUMBER [1]	The transaction's party bank account. Note: If you do not provide a party bank account, CMM can select one using routing rules.	Entity bank account primary number	• Must be held by the transaction's party	No	User
BANKACCOUNTENTITYID [1]	The owner of the transaction's party bank account.	Entity ID	• None	No	User
ENTITYBANKACCOUNTCURRENCYCODE [1]	The currency of the transaction's party bank account.	Currency ID	• None	No	User
Counterparty					
COUNTERPARTYID	The transaction's counterparty.	Entity ID Counterparty ID	• None	No	User
Other					
CUSTOMERREFERENCENUMBER	The transaction's customer reference ID.	Text (50 characters maximum)	• None	No	User

Column	Description	Value	Constraints	Req.	Srce
DELETEEXISTINGREFERENCE	A value that indicates whether CMM should delete existing receipts for the current invoice.	One of the following values: • Y if CMM should deleting existing receipts • N if CMM should not delete existing receipts Note: N is the default if you do not specify a value.	• None	No	User
HANDLINGINSTRUCTIONCODE	The ID of the transaction's handling instructions.	Text (5 characters maximum)	• None	No	User
INVOICENUMBER	The transaction's invoice number.	Text (20 characters maximum)	• Must be provided if existing references are deleted.	No	User
Table notes: 1. Only supply a value in this column if you have set SUPPLYPAYINGACCOUNT to Y.					

A.3.4 Wallstreet flat file direct debit

The following is an example direct debit flat file (with tabs as the delimiters):

```
Header 1  HEADER⇨Acme_DD_10/20/06⇨5000.00⇨DD⇨JSmith
       2  FORMAT1ENTITYID⇨COUNTERPARTYID⇨TRANSACTIONDATE⇨CURRENCYCODE⇨AMOUNT
Body   3  Acme⇨SmithCo⇨20-Oct-2006⇨CAD⇨5000.00
       4  Remittance⇨S⇨100⇨12345⇨18-Oct-2006⇨Due⇨5000.00⇨CAD⇨5000.00⇨0.00
```

This file contains four lines grouped into two sections:

Line	Name	Description
Header		
1	File information	General information on the file.
2	File layout	The columns included in the file (in the order in which they display in line 3).
Body		
3	Transaction details	The individual transactions. Note: Include one line 3 for each transaction in the file.
4	Remittance details	The individual remittance details. Note: Include one line 4 for each remittance detail in the file.

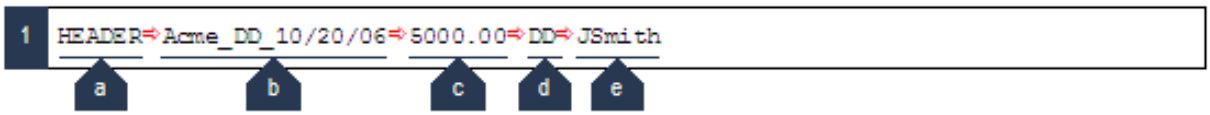
A.3.4.1 Wallstreet flat file direct debit header

The header section of a direct debit flat file consists of two lines:

- File information (line 1)
- File layout (line 2).

A.3.4.1.1 File information (line 1)

Line 1 contains general information on the file:

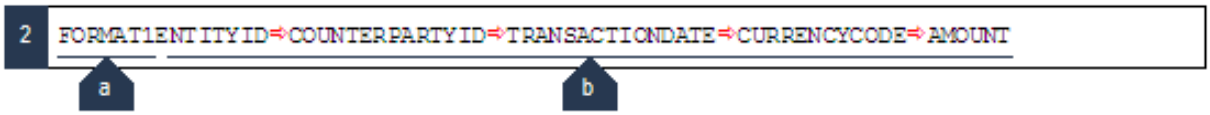


The following table defines the components of this line:

No.	Component	Description	Value	Constraints	Req.	Source
1a	File information indicator	An indicator that the line contains general file information.	HEADER	• None	Yes	User
1b	File ID	A unique identifier for the file. Note: CMMchecks the value of this component before loading the file to prevent accidentally reimporting the same file twice.	Text (50 characters maximum)	• None	Yes	User
1c	Control total	The sum of all transaction amounts in the file. Note: If you supply a value, CMM does not sum the amounts of all transactions in the file to confirm completeness before importing.	Floating point number	• None	No	User
1d	Type indicator	The file type indicator.	DD	• None	Yes	User
1e	User ID	Your user ID.	Text	• None	No	User

A.3.4.1.2 File layout (line 2)

Line 2 defines columns included in the file:



The following table defines the components of this line:

No.	Component	Description	Value	Constraints	Req.	Source
2a	File layout indicator	An indicator that the line contains file layout information.	FORMAT1	• None	Yes	User

No.	Component	Description	Value	Constraints	Req.	Source
2b	Columns	A list (tab- or comma-delimited) of all of the columns that appear in the file. This list identifies: • The columns you are providing in the import file • The order of the columns. The valid columns are documented in A.3.4.2 Wallstreet flat file direct debit body on page 166.	Text	• None	Yes	User

Note: Do not place a tab, comma, space, or other character between the file layout indicator (component 2a) and the columns (component 2b).

A.3.4.2 Wallstreet flat file direct debit body

The body section of a direct debit flat file consists of two lines:

- Transaction details (line 3)
- Remittance details (line 4).

A.3.4.2.1 Transaction details (line 3)

Line 3 contains the individual transactions. (Include one line 3 for each transaction in the file.)

The column values in line 3 must display in the same order as the columns in line 2 and must be delimited with tabs or commas.

The following are columns supported by the Wallstreet standard direct debit flat file format:

Column	Description	Value	Constraints	Req.	Srce
Basic					
CURRENCYCODE	The transaction's currency.	Currency ID	• None	Yes	User
AMOUNT	The transaction's absolute value (in the transaction's currency).	Floating point number	• None	Yes	User
PAYMENTSTATUSCODE	A value that indicates whether the transaction is actual activity or forecasted activity. Note: CMM 7.1 introduced forecasts as a replacement for forecasted transactions.	One of the following: • A if the transaction is actual activity • F if the transaction is forecasted activity Note: A is the default if you do not specify a value.	• None	No	User

Column	Description	Value	Constraints	Req.	Srce
CASHFLOWTYPE	The transaction's cash flow type. By assigning cash flow types to transactions and other activity in CMM, you can categorize this activity for a variety of purposes. In particular, you can run several reports in the module that display activity by cash flow type. Note: In CMM, a cash flow type is an instrument type with the **Status** attribute set to Enabled.	Instrument type ID Note: COMMP (Commercial Payment) is the default if you do not specify a value.	• None	No	User
PAYMENTMETHOD	The transaction's payment method. A transaction's payment method defines how the transaction is to be paid when it is released (for example, by EFT or by check).	Payment method ID	• None		User
TRANSACTIONDATE	The transaction's transaction date.	Date Note: The current date is the default if you do not specify a value.	• None	No	User
VALUEDATE	The transaction's value date.	Date Note: The transaction date is the default if you do not specify a value.	• None	No	User
CREATIONDATE	The transaction's creation date.	Date Note: The transaction date is the default if you do not specify a value.	• None	No	User
DESCRIPTION/REFERENCE	A relevant description of or other information pertaining to the transaction.	Text (2,000 characters maximum)	• None	No	User
Entity					
ENTITYID	The transaction's party.	Entity ID	• None	Yes	User

Column	Description	Value	Constraints	Req.	Srce
SUPPLYPAYINGACCOUNT	A value that indicates whether you are supplying party bank account details.	One of the following values: • Y if you are supplying party bank account details • N if you are not supplying party bank account details Note: N is the default if you do not specify a value.	• None	No	User
Entity bank					
ENTITYBANKBRANCHID [1]	The branch ID of the transaction's party bank (as assigned by an agency).	Text (15 characters maximum)	• None	No	User
Entity bank account					
ENTITYBANKACCOUNTNUMBER [1]	The transaction's party bank account. Note: If you do not provide a party bank account, CMM can select one using routing rules.	Entity bank account primary number	• Must be held by the transaction's party	No	User
BANKACCOUNTENTITYID [1]	The owner of the transaction's party bank account.	Entity ID	• None	No	User
ENTITYBANKACCOUNTCURRENCYCODE [1]	The currency of the transaction's party bank account.	Currency ID	• None	No	User
Counterparty					
COUNTERPARTYID	The transaction's counterparty.	Entity ID Counterparty ID	• None	No	User
SUPPLYCPTYDETAILS	A value that indicates whether you are supplying counterparty details.	One of the following values: • Y if you are supplying counterparty details • N if you are not supplying counterparty details Note: N is the default if you do not specify a value.	• None	No	User

Column	Description	Value	Constraints	Req.	Srce
COUNTERPARTYNAME [2]	The name of the transaction's counterparty.	Text (35 characters maximum)	• None	No	User
COUNTERPARTYADDRESS1 COUNTERPARTYADDRESS2 COUNTERPARTYADDRESS3 COUNTERPARTYADDRESS4 [2]	The address of the transaction's counterparty.	Text (35 characters maximum per field)	• None	No	User
COUNTERPARTYCITY [2]	The city of the transaction's counterparty.	Text (35 characters maximum)	• None	No	User
COUNTERPARTYSTATE [2]	The state of the transaction's counterparty.	State ID	• None	No	User
COUNTERPARTYCOUNTRYCODE [2]	The country of the transaction's counterparty.	Country ID	• None	No	User
COUNTERPARTYZIP [2]	The postal or ZIP code of the transaction's counterparty.	Text (10 characters maximum)	• None	No	User
Counterparty bank					
COUNTERPARTYBANKCODE [2]	The SWIFT or ABA code of the transaction's counterparty bank.	Text (20 characters maximum)	• None	No	User
COUNTERPARTYBANKCODETYPE [2]	The SWIFT or ABA code type of transaction's counterparty bank.	One of the following: • S if the code type is SWIFT • A if the code type is ABA	• None	No	User
COUNTERPARTYBANKBRANCHID [2]	The branch ID of the transaction's counterparty bank (as assigned by an agency).	Text (15 characters maximum)	• None	No	User
COUNTERPARTYBANKNAME	The name of the transaction's counterparty bank.	Text (35 characters maximum)	• None	No	User
COUNTERPARTYBANKADDRESS COUNTERPARTYBANKADDRESS2	The address of the transaction's counterparty bank.	Text (35 characters maximum per field)	• None	No	User
COUNTERPARTYBANKCITY	The city of the transaction's counterparty bank.	Text (35 characters maximum)	• None	No	User

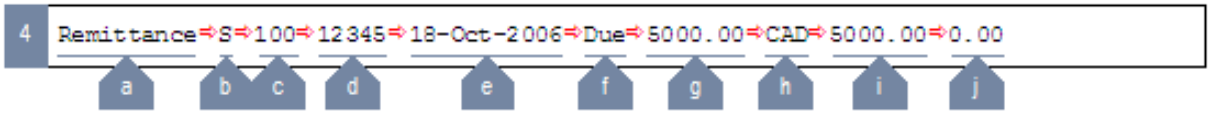
Column	Description	Value	Constraints	Req.	Srce
COUNTERPARTYBANKSTATE	The state of the transaction's counterparty bank.	State ID	• None	No	User
COUNTERPARTYBANKZIP	The postal or ZIP code of the transaction's counterparty bank.	Text (10 characters maximum)	• None	No	User
Counterparty bank account					
COUNTERPARTYBANKACCOUNTNUMBER [2]	The primary number (usually, BBAN) of the transaction's counterparty bank account.	Text (60 characters maximum)	• Must validate against the BBAN rules of the bank account 's country	No	User
CPTYPRIMARYACCOUNTNUMBERTYPE [2]	The primary number type of the transaction's counterparty bank account.	BBAN	• None	No	User
CPTYSECONDARYACCOUNTNUMBER [2]	The secondary number (usually IBAN) of the transaction's counterparty bank account.	Text (60 characters maximum)	• Must validate against the IBAN rules	No	User
CPTYSECONDARYACCOUNTNUMBERTYPE [2]	The secondary number type of the transaction's counterparty bank account.	IBAN	• None	No	User
COUNTERPARTYBANKACCOUNTCOUNTRYCODE	The country of the transaction's counterparty bank.	Country ID	• None	No	User
COUNTERPARTYBANKCCOUNTCURRENCYCODE	The currency of the transaction's counterparty bank account.	Currency ID	• None	No	User
Intermediary bank					
Intermediary bank account					
INTERMEDIARYBANKACCOUNTNUMBER	The primary number (usually BBAN) of the transaction's intermediary bank account.	Text (60 characters maximum)	• Must validate against the BBAN rules of the bank account 's country	No	User

Column	Description	Value	Constraints	Req.	Srce
Aggregation					
AGGREGATEDETAILCODE	A value that indicates whether the transaction is an aggregate or detail.	One of the following values: • A if the transaction is an aggregate • D if the transaction is a detail	• None	No	User
AGGREGATEREFERENCEID	If the transaction is an aggregate, the transaction's unique reference number. If the transaction is a detail, the reference number of the aggregate transaction in which the detail transaction is contained.	Text (50 characters maximum)	• None	No	User
Regulatory reporting					
REGULATORYCODE	The transaction's regulatory code.	Regulatory code	• None	No	User
REGULATORYQUALIFIER	The qualifier of the transaction's regulatory code.	Text (50 characters maximum)	• None	No	User
REGULATORYAGENCY	The agency of the transaction's regulatory code.	Text (50 characters maximum)	• Must be provided if the transaction's regulatory code is provided	No	User
REGULATORYDESCRIPTION	A description of the transaction's regulatory code.	Text (50 characters maximum)	• None	No	User
Other					
BANKINSTRUCTIONS	Instructions or other messages to the bank.	Text (100 characters maximum)	• None	No	User
BANKPREAUTHORIZATIONID	The transaction's pre-authorization ID.	Text (255 characters maximum)	• None	No	User
BANKPREAUTHORIZATIONSTATUS	The transaction's pre-authorization status.	One of the following: • AUTHORIZED • UNAUTHORIZED	• None	No	User
COUNTERPARTYMESSAGE	Instructions or other messages to the transaction's counterparty.	Text (100 characters maximum)	• None	No	User

Column	Description	Value	Constraints	Req.	Srce
CUSTOMERBATCHREFERENCEID	The customer's batch reference ID for the transaction.	Text (30 characters maximum)	• None	No	User
CUSTOMERREFERENCENUMBER	The transaction's customer reference ID.	Text (50 characters maximum)	• None	No	User
CUSTOMERTRANSACTIONREFERENCEID	The customer's reference ID for the transaction. Note: This ID does not have to be unique.	Text (30 characters maximum)	• None	No	User
HANDLINGINSTRUCTIONCODE	The ID of the transaction's handling instructions.	Text (5 characters maximum)	• None	No	User
TRANSACTIONTYPECODE	The ID of the transaction's type. Note: This is country and bank specific.	Text (10 characters maximum)	• None	No	User
Table notes: 1. Only supply a value in this column if you have set SUPPLYPAYINGACCOUNT to Y. 2. Only supply a value in this column if you have set SUPPLYCPTYDETAILS to Y.					

A.3.4.2.2 Remittance details (line 4)

Line 4 contains the individual remittance details:



(Include one line 4 for each remittance detail in the file.)

The following table defines the components of this line:

No.	Component	Description	Value	Constraints	Req.	Source
4a	Remittance indicator	An indicator that the line contains file layout information.	Remittance	• None	Yes	User

No.	Component	Description	Value	Constraints	Req.	Source
4b	Remittance handling code	A value that indicates how remittance details are handled in the file.	One of the following values: • `S` if the remittance details are handled in a single line • `M` if the remittance details are handled in multiple lines Note: If you enter `M`, provide one remittance line detail for the invoice amount and one for the actual amount paid.	• None	Yes	User
4c	Document type code	The remittance detail's type.	One of the following values: • `100` if the remittance detail is an invoice • `101` if the remittance detail is a credit • `103` if the remittance detail is a debit	• None	Yes	User
4d	Document message number	The remittance detail's invoice, credit note, or debit note.	Text (35 characters maximum)	• None	Yes	User
4e	Document date	The remittance detail's date.	Date	• None	No	User
4f	Document amount type	The type of the remittance detail's amount.	One of the following values: • `Due` if the remittance handling code is set to `S` • `201` if the remittance detail's amount is paid • `202` if the remittance detail's amount is remitted • `203` if the remittance detail's amount is discontinued • `204` if the remittance detail's amount is payable	• None	No	User
4g	Document Amount	The remittance detail's amount.	Floating point number	• None	No	User
4h	Document currency code	The remittance detail's currency.	Currency ID	• None	No	User

No.	Component	Description	Value	Constraints	Req.	Source
4i	Amount paid	The actual amount paid.	Floating point number	• Can only be provided if the remittance handling code is Y.	No	User
4j	Amount Discounted	The amount discounted.	Floating point number	• Can only be provided if the remittance handling code is Y.	No	User

A.3.5 EDIFACT PAYEXT

This section documents the EDIFACT D97A extended payment order (PAYEXT) message file format that CMM currently supports for accounts payable file imports.

The following table presents the components of the PAYEXT message file format:

Level	Name	Description	Required	Repetitions
1	UNB	Interchange header	Yes	1
1	UNH	Message header	Yes	1
1	BGM	Beginning of message	Yes	1
1	BUS	Business function	No	0 to 1
1	PAI	Payment instructions	No	0 to 1
1	DTM	Date/time/period	Yes	1 to 4
1	PAYEXT01	Segment group 1	No	0 to 1
2	RFF	Reference	No	0 to 1
2	DTM	Date/time/period	No	0 to 1
1	PAYEXT02	Segment group 2	Yes	1
2	MOA	Monetary amount	Yes	1
1	PAYEXT03	Segment group 3	No	0 to 1
2	FII	Financial institution information	Yes	1 to 2
2	CTA	Contact information	No	0 to 1
2	COM	Communication contact	No	0 to 1
1	PAYEXT04	Segment group 4	No	0 to 6
2	NAD	Name and address	Yes	1
2	CTA	Contact information	No	0 to 1
2	COM	Communication contact	No	0 to 1
1	PAYEXT05	Segment group 5	No	0 to 4
2	INP	Parties to instruction	Yes	1
2	FTX	Free text	No	0 to 1
2	DTM	Date/time/period	No	0 to 2
1	PAYEXT07	Segment group 7	No	0 to 1

Level	Name	Description	Required	Repetitions
2	PRC	Process identification	Yes	1
2	FTX	Fee text	No	0 to 5
2	PAYEXT08	Segment group 8	No	0 to 9,999
3	DOC	Document/message details	Yes	1
3	MOA	Monetary amount	Yes	0 to 5
3	DTM	Date/time/period	Yes	0 to 5
3	RFF	Reference	Yes	0 to 5
1	PAYEXT15	Segment group 15	No	0 to 5
2	AUT	Authentication result	Yes	1
2	DTM	Date/time/period	No	0 to 1
1	UNT	Message trailer	Yes	1
1	UNZ	Interchange trailer	Yes	1

Note: CMM does not support segment groups 1 and 5.

For each element, this section documents the following:

Criterion	Description
Segment	The element's parent segment.
Sep.	The character that separates the element from neighbor elements.
Element	The element's identifier.
Description	A brief description of the element.
Req.	A flag indicating if the element is required or not.
Used	A flag indicating if the element is used by CMM or not.
Format	The element's format. For more information, see A.12.1 EDIFACT Level A character set on page 302.

A.3.5.1 EDIFACT PAYEXT start of interchange

The start of interchange specifies the start of the interchange. This group is required and can be repeated once. It contains the following segment:

ID	Name	Required	Repetitions
UNB	Interchange header	Yes	1

A.3.5.1.1 UNB segment

The UNB segment starts, identifies, and specifies an interchange.

The following table presents the elements of this segment:

Segment	Sep.	Element	Description	Req.	Used	Format
UNB			Segment identifier	Yes	Yes	

Segment	Sep.	Element	Description	Req.	Used	Format
		S001	Syntax identifier	Yes	Yes	
	+	0001	Syntax identifier [1]	Yes	Yes	a4
	:	0002	Syntax version number [2]	Yes	Yes	n1
		S002	Interchange sender	Yes	Yes	
	+	0004	Sender identifier [3]	Yes	Yes	an..35
	:	0007	Sender identifier code qualifier	No	Yes	an..4
		S003	Interchange recipient	Yes	No	
	+	0010	Recipient identifier	Yes	No	an..35
	:	0007	Recipient identifier code qualifier	Yes	No	an..4
		S004	Date/time of preparation	Yes	Yes	
	+	0017	Date of preparation [4]	Yes	Yes	n6
	:	0019	Time of preparation [5]	Yes	Yes	n4
	+	0020	Interchange control reference [6]	Yes	Yes	an..14
		S005	Recipient’s reference/password	No	No	
	+	0022	Recipient’s reference/password	Yes	No	an..14
	:	0025	Recipient’s reference/password qualifier	No	No	an2
	+	0026	Application reference [7]	No	Yes	an..14
	+	0029	Processing priority code	No	No	a1
	+	0031	Acknowledgement request	No	No	n1
	+	0032	Communications agreement identifier	No	No	an..35
	+	0035	Test indicator	No	Yes	n1
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element contains a fixed value of UNOB, indicating that the interchange uses the EDIFACT Level A character set. For more information, see A.12.1 EDIFACT Level A character set on page 302.
2. This element contains a fixed value of 2.
3. This element contains the sender’s EDI organization identifier code as specified in the interchange agreement. For CMM, this element must contain a valid entity ID from the CMM database or an ID that is mapped to a valid entity ID from the CMM database.
4. This element contains the date on which the interchange was prepared in YYMMDD format.
5. This element contains the time on which the interchange was prepared in HHMM format.
6. This element contains a unique reference for the interchange, assigned by the sender. This is a sequential number that is incremented by one for each interchange sent. In a customer payment, this element must always be identical to element 0020 in the UNZ segment.
7. This element contains the message type. The value of this element is usually PAYEXT, which is identical to element 0065 in the UNH segment.

The following is an example of this segment:

```
UNB+UNOB:2+ENTITY1+CITIGB2L+970115:0920+INTERCHANGEREF++PAYEXT’
```

A.3.5.2 EDIFACT PAYEXT start of message

The start of message specifies the start of the message. This group is required and can be repeated once. It contains the following segment:

ID	Name	Required	Repetitions
UNH	Message header	Yes	1
BGM	Beginning of message	Yes	1
BUS	Business function	No	1
PAI	Payment instructions	No	1
DTM	Date/time/period	Yes	4

The UNH and BGM segments contain data related to the whole message. The BUS, PAI, and DTM segments contain date related to the payment means, charge allocation, and message creation date respectively.

A.3.5.2.1 UNH segment

The UNH segment is a service segment that starts and uniquely identifies a message.

The following table presents the elements of this segment:

Segment	Sep.	Element	Description	Req.	Used	Format
UNH			Segment identifier	Yes	Yes	
	+	0062	Message reference number [1]	Yes	Yes	an..14
		S009	Message identifier	Yes	Yes	
	+	0065	Message type [2]	Yes	Yes	an..6
	:	0052	Message version number [3]	Yes	Yes	an..3
	:	0054	Message release number [4]	Yes	Yes	an..3
	:	0051	Controlling agency [5]	Yes	Yes	an..2
	:	0057	Association-assigned code	No	No	an..6
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element contains a unique reference for the message, assigned by the sender. The value for element 0062 in this segment must be the same as value for element 0062 in the UNT segment.
2. This element contains the EDIFACT message type defined within the UNH and UNT segments. In an extended payment order message, the message type must always be PAYEXT.
3. This element identifies the EDIFACT version number. The version of an extended payment order message must always be D.
4. This element contains the EDIFACT release number. The release number of an extended payment order message must always be 97A.
5. Because PAYEXT messages are defined by UN/EDIFACT, this element contains the value UN.

The following is an example of this segment:

```
UNH+1+PAYEXT:D:97A:UN:PH0011’
```

A.3.5.2.2 BGM segment

The BGM segment is used by the sender to uniquely identify the message by using its code type, number, and—when necessary—function.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
BGM			Segment identifier	Yes	Yes	
		C002	Document/message name	No	Yes	
	+	1001	Document/message name (coded) [1]	No	Yes	an..3
	+	1004	Document/message number [2]	No	Yes	an..35
	+	1225	Message function (coded)	No	No	an..3
	+	4343	Response type (coded)	No	No	an..3
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the type of document contained within the message, in coded format. The valid value is `451` for extended payment order.
2. This element contains a reference to the transaction assigned by the sender. For CMM, this element is used as the customer reference number and must be unique across the whole organization.

The following is an example of this segment:

```
BGM+451+PO176521A’
```

A.3.5.2.3 BUS segment

The BUS segment is used to:

- Identify the payment as belonging to a particular business function (for example, GDS for "Goods and Services")
- Indicate if it is domestic or international
- Indicate if it is intercompany and can be routed through in-house bank accounts.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
BUS			Segment identifier	Yes	Yes	
	+	C521	Business function [1]	No	Yes	
	:	4027	Business function qualifier [2]	Yes	Yes	an..3
	:	4025	Business function (coded) [3]	Yes	Yes	an..3
	:	1131	Code list qualifier	No	No	an..3
	:	3055	Code list responsible agency (coded)	No	No	an..3
	:	4022	Business description	No	No	an..70
	+	3279	Geographic environment (coded) [4]	No	Yes	an..3
	+	4487	Type of financial transaction (coded)	No	Yes	an..3
		C551	Bank operation	No	No	

Segment	Sep.	Item	Description	Req.	Used	Format
	+	4383	Bank operation (coded)	No	No	an..3
	+	4463	Intercompany payment (coded) [5]	Yes	Yes	an..3
	’		End of segment marker	Yes	Yes	

Table notes:

1. This composite element contains the regulatory code information. Elements used to make up the regulatory code information include 4027, 4025, 1131, 3055, and 4022.
2. This element contains the type of business function.
3. This element contains a code describing the specific business function.
4. This element indicates whether the payment is domestic or international. Acceptable values: `DO` for domestic and `IN` for international. Through CMM’s routing functionality, an international payment can be routed through a domestic account.
5. This element indicates whether the payment is intercompany and should be routed through in-house bank accounts or external and should be routed through external bank accounts. If the payment is intercompany, both payor and payee must be valid entities in CMM and must have in-house bank accounts. Acceptable values: `1` for intercompany and any value other than 1 for external.

The following is an example of this segment:

```
BUS+1:GDS+DO+++1’
```

A.3.5.2.4 PAI segment

The PAI segment specifies the method of payment and the payment channel to be used for the payment. This segment is not required and can be included once per file.

Note: The absence of this segment means that the payment is to be made via EFT (42).

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
PAI			Segment identifier	Yes	Yes	
		C534	Payment instruction details	Yes	Yes	
	+	4439	Payment conditions (coded)	No	No	an..3
	:	4431	Payment guarantee (coded)	No	No	an..3
	:	4461	Payment means (coded) [1]	No	Yes	an..3
	:	1131	Code list qualifier	No	No	an..3
	:	3055	Code list resp. agency (coded)	No	No	an..3
	:	4435	Payment channel (coded) [2]	No	Yes	an..3
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the means by which payment or reimbursement is to be made. Acceptable values: `20` for check (US and Canada only) and `42` for payment to bank account (wire transfer). If the payment method is set to `42`, the payment method in CMM is set to EFT. At payment release, CMM determines if the payment method is a wire transfer or must go through a clearing system if it is domestic.
2. This element contains a fixed value of `6`.

The following is an example of this segment:

```
PAI+::42:::6'
```

A.3.5.2.5 DTM segment

The DTM segment can be repeated twice to include the value date (203) and transaction date (137) in CMM.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
DTM			Segment identifier	Yes	Yes	
		C507	Date/time/period	Yes	Yes	
	+	2005	Date/time/period qualifier [1]	Yes	Yes	an..3
	:	2380	Date/time/period [2]	No	Yes	an..35
	:	2379	Date/time/period format qualifier [3]	No	Yes	an..3
	'		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the date type. Acceptable values: `137` for message creation date (transaction date in CMM) and `203` for value date (value date in CMM).
2. This element contains the message creation date or value date and must be in the YYYYMMDD format.
3. This element contains the date format and must always be `102` (for the YYYYMMDD format).

The following is an example of this segment:

```
DTM+137:20000728:102'
```

A.3.5.3 EDIFACT PAYEXT segment group 1

Segment group 1 identifies the ACH transaction code. This group is required and can be repeated once. It contains the following segments:

ID	Name	Required	Repetitions
RFF	Reference (ACH transaction code)	Yes	1
DTM	Date/time period	Yes	1

Note: This group is not supported by CMM.

A.3.5.3.1 RFF segment

The RFF segment identifies the ACH transaction code.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
RFF			Segment identifier	Yes	Yes	
		C506	Reference	Yes	Yes	
	+	1153	Reference qualifier [1]	Yes	Yes	an..3
	:	1154	Reference number [2]	No	Yes	an..35
	'		End of segment marker	Yes	Yes	

Segment	Sep.	Item	Description	Req.	Used	Format
Table notes: 1. This element contains the type of reference in coded form. The valid value is RT for payee's financial institution transit routing number. 2. This element contains the ACH transaction code (usually four characters) as defined in the country rules. Its use is dictated by the requirements of individual countries.						

The following is an example of this segment:

```
RFF+RT:0123'
```

A.3.5.3.2 DTM segment

The DTM segment can be repeated once to include the value date (203) in CMM.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
DTM			Segment identifier	Yes	Yes	
		C507	Date/time/period	Yes	Yes	
	+	2005	Date/time/period qualifier [1]	Yes	Yes	an..3
	:	2380	Date/time/period [2]	No	Yes	an..35
	:	2379	Date/time/period format qualifier [3]	No	Yes	an..3
	'		End of segment marker	Yes	Yes	
Table notes: 1. This element contains the date type. Acceptable value: 203 for order execution date. 2. This element contains the message creation date or value date and must be in the YYYYMMDD format. 3. This element contains the date format and must always be 102 (for the YYYYMMDD format).						

The following is an example of this segment:

```
DTM+137:20000728:102'
```

A.3.5.4 EDIFACT PAYEXT segment group 2

Segment group 2 identifies the monetary amount of the payment. This group is required and can be repeated once. It contains the following segments:

ID	Name	Required	Repetitions
MOA	Monetary amount	Yes	1

A.3.5.4.1 MOA segment

The MOA segment identifies the original amount requested by the ordering customer to be paid to the beneficiary's (or "payee's") account.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
MOA			Segment identifier	Yes	Yes	
		C516	Monetary amount	Yes	Yes	

Segment	Sep.	Item	Description	Req.	Used	Format
	+	5025	Monetary amount type qualifier [1]	Yes	Yes	an..3
	:	5004	Monetary amount	No	Yes	n..18
	:	6345	Currency (coded) [2]	Yes	Yes	an..3
	:	6343	Currency qualifier [3]	No	Yes	an..3
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the monetary amount type. Acceptable value: `9` for amount due/amount payable.
2. This element must contain a valid ISO 4217 currency code.
3. This element contains the currency qualifier: `11` for payment currency. CMM is only concerned with the monetary amount (element 5004) and the currency and always assumes the currency is the payment currency.

The following is an example of this segment:

```
MOA+9:1000.00:EUR’
```

A.3.5.5 EDIFACT PAYEXT segment group 3

Segment group 3 identifies the financial institutions (including their contact people or departments and their communication numbers) and accounts involved in the transaction. This group is required and can be repeated twice. It contains the following segments:

ID	Name	Required	Repetitions
FII	Financial institution information	Yes	1
CTA	Contact information	No	1
COM	Communication contact	No	1

A.3.5.5.1 Entity bank FII segment

The FII segment identifies a specific account and related financial institution involved in the extended payment order. The first occurrence of this segment identifies the entity’s (ordering party’s) account information.

Note: If routing is enabled in CMM, this section is ignored and the payment is routed through the most cost effective bank account.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
FII			Segment identifier	Yes	Yes	
	+	3035	Party qualifier [1]	Yes	Yes	
		C078	Account identification	No	Yes	an..35
	+	3194	Account holder number [2]	No	Yes	an..35
	:	3192	Account holder name [3]	No	Yes	an..35
	:	3192	Account holder name [3]	No	Yes	an..35
	:	6345	Currency (coded) [4]	No	Yes	an..3

Segment	Sep.	Item	Description	Req.	Used	Format
		C088	Institution identification [5]	No	Yes	an..35
	+	3433	Institution name identifier [6]	No	Yes	an..11
	:	1131	Code list qualifier [7]	No	Yes	an..3
	:	3055	Code list rsp. agency (coded)	No	No	an..3
	:	3434	Institution branch number	No	No	an..17
	:	1131	Code list qualifier	No	No	an..3
	:	3055	Code list rsp. agency (coded)	No	No	an..3
	:	3432	Institution name [8]	No	Yes	an..70
	:	3436	Institution branch place [9]	No	Yes	an..70
	+	3207	Country (coded) [10]	No	Yes	an..3
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the function of the party identified. Acceptable value: OR for ordered bank.
2. This element contains the number of the account that is to be debited to effect the payment.
3. This element contains the name of the account holder.
4. This element must contain a valid ISO 4217 currency code.
5. This composite element contains the ordered bank.
6. This element must contain a valid ISO bank identifier code.
7. This element must contain 25 (bank identification).
8. This element contains the bank name.
9. This element contains the bank city.
10. This element contains the bank country.

The following is an example of this segment:

```
FII+OR+999888333:::EUR+3000400813:25:108::::BANQUE NATIONALE DE PARIS:PARIS+FR’
```

A.3.5.5.2 Target bank FII segment

The FII segment identifies a specific account and related financial institution involved in the extended payment order. The second occurrence identifies the beneficiary’s bank and account. The information from this segment (or the lack of) can sometimes result in the payment not being processed by the sending bank. Therefore, it is recommended to send as much information as possible in this segment.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
FII			Segment identifier	Yes	Yes	
	+	3035	Party qualifier [1]	Yes	Yes	
		C078	Account identification	No	Yes	an..35
	+	3194	Account holder number [2]	No	Yes	an..35
	:	3192	Account holder name [3]	No	Yes	an..35

Segment	Sep.	Item	Description	Req.	Used	Format
	:	3192	Account holder name [3]	No	Yes	an..35
	:	6345	Currency (coded) [4]	No	Yes	an..3
		C088	Institution identification [5]	No	Yes	an..35
	+	3433	Institution name identifier [6]	No	Yes	an..11
	:	1131	Code list qualifier [7]	No	Yes	an..3
	:	3055	Code list rsp. agency (coded) [8]	No	No	an..3
	:	3434	Institution branch number [9]	No	No	an..17
	:	1131	Code list qualifier	No	No	an..3
	:	3055	Code list rsp. agency (coded)	No	No	an..3
	:	3432	Institution name [10]	No	Yes	an..70
	:	3436	Institution branch place [11]	No	Yes	an..70
	+	3207	Country (coded) [12]	No	Yes	an..3
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the function of the party identified. Acceptable value: `BF` for beneficiary bank.
2. This element contains the number of the account that is to be debited to effect the payment.
3. This element contains the name of the account holder.
4. This element must contain a valid ISO 4217 currency code.
5. This composite element contains the beneficiary’s bank. Certain countries and banks require different beneficiary information for processing payments, and this validation exists within CMM. For example, there are certain countries (for example, Italy) that require the institution name (element 3436) for the payment to be processed. If certain information, such as the institution name, exists in the AP file generation system that is being used, Wallstreet recommends that this information be provided in the file for CMM purposes.
6. This element must contain a valid eight- or eleven-digit SWIFT code.
7. This element must contain `25` (bank identification).
8. This element must contain `5` (BIC).
9. If an eight-digit SWIFT code is provided in element 3433, this element is required for processing the payment.
10. If an eight-digit SWIFT code is provided in element 3433, this element is required for processing the payment. In addition, certain countries require this information be provided.
11. If an eight-digit SWIFT code is provided in element 3433, this element is required for processing the payment. In addition, certain countries require this information be provided.
12. This element must contain a valid ISO two-character alphabetic country code. This element is required as it determines routing that takes place in CMM.

The following is an example of this segment:

```
FII+OR+999888333:::EUR+3000400813:25:108::::BANQUE NATIONALE DE PARIS:PARIS+FR’
```

A.3.5.5.3 CTA segment

The CTA segment identifies the contact person at either of the financial institutions.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
CTA			Segment identifier	Yes	Yes	
	+	3139	Contact function (coded) [1]	No	Yes	an..3
	+	C056	Department or employee details	No	Yes	
	:	3413	Department or employee identifier [2]	No	Yes	an..17
	:	3412	Department or employee [3]	No	Yes	an..35
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element must contain `IC` (information contact).
2. This element contains the internal identification code.
3. This element contains the department or person within an organizational entity.

The following is an example of this segment:

```
CTA+IC+:T Smith’
```

A.3.5.5.4 COM segment

The COM segment provides the communication details necessary for sending printed information to the financial institutions. This segment is not required and can be included once per file.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
COM			Segment identifier	Yes	Yes	
		C076	Communication contact	Yes	Yes	
	+	3148	Communication number [1]	Yes	Yes	an..512
	:	3155	Communication channel qualifier [2]	Yes	Yes	an..3
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the communication number.
2. This element contains the communication channel. Acceptable values: `MA` for mail, `TM` for telemail, `FX` for facsimile, and `EM` for e-mail.

A.3.5.6 EDIFACT PAYEXT segment group 4

Segment group 4 identifies the parties involved in the extended payment order by function as well as by identifier or by name and address. This group is not required and can be repeated six times. It contains the following segments:

ID	Name	Required	Repetitions
NAD	Payee name and address	Yes	1
CTA	Payee contact information	No	1

ID	Name	Required	Repetitions
COM	Payee communication contact	No	1
NAD	Payor name and address	Yes	1
CTA	Payor contact information	No	1
COM	Payor communication contact	No	1

A.3.5.6.1 Payee NAD segment

The NAD segment identifies a specific party involved in the extended payment order. The first occurrence of this segment identifies the payee.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
NAD			Segment identifier	Yes	Yes	
	+	3035	Party qualifier [1]	Yes	Yes	an..3
		C082	Party identification details	No	Yes	
	+	3039	Party identification [2]	Yes	Yes	an..35
	:	1131	Code list qualifier	No	No	an..3
	:	3055	Code list resp. agency (coded)	No	No	an..3
		C058	Name and address [3]	No	Yes	
	+	3124	Name and address line	Yes	Yes	an..35
	:	3124	Name and address line	No	Yes	an..35
	:	3124	Name and address line	No	Yes	an..35
	:	3124	Name and address line	No	Yes	an..35
	:	3124	Name and address line	No	Yes	an..35
		C080	Party name [4]	No	Yes	
	+	3036	Party name	Yes	Yes	an..35
	:	3036	Party name	No	Yes	an..35
	:	3036	Party name	No	Yes	an..35
	:	3036	Party name	No	Yes	an..35
	:	3036	Party name	No	Yes	an..35
		C059	Street [5]	No	Yes	
	+	3042	Street and number/PO box	Yes	Yes	an..35
	:	3042	Street and number/PO box	No	Yes	an..35
	:	3042	Street and number/PO box	No	Yes	an..35
	:	3042	Street and number/PO box	No	Yes	an..35
	+	3164	City name	No	Yes	an..35
	+	3229	Country subentity identifier	No	Yes	an..9
	+	3251	Postal code	No	Yes	an..9
	+	3207	Country (coded) [6]	No	Yes	an..3

Segment	Sep.	Item	Description	Req.	Used	Format
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the function of the party identified. Acceptable value: `PE` for payee.
2. If the payment is intercompany, this element must contain an entity ID stored in CMM.
3. For proper handling in CMM, use composite elements C080 and C059 rather than C058.
4. If the payment method is check, this composite element is required.
5. If the payment method is check, this composite element is required.
6. This element must contain a valid ISO country code.

The following is an example of this segment:

```
NAD+PE+88889999:160:92++PAYEE+123 MAIN STREET+NEW YORK+NY+82100+US’
```

A.3.5.6.2 Payee CTA segment

The payee CTA segment identifies the contact person for the payee.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
CTA			Segment identifier	Yes	Yes	
	+	3139	Contact function (coded) [1]	No	Yes	an..3
	+	C056	Department or employee details	No	Yes	
	:	3413	Department or employee identifier [2]	No	Yes	an..17
	:	3412	Department or employee [3]	No	Yes	an..35
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element must contain `IC` (information contact).
2. This element contains the internal identification code.
3. This element contains the department or person within an organizational entity.

The following is an example of this segment:

```
CTA+IC+:T Smith’
```

A.3.5.6.3 Payee COM segment

The payee COM segment provides the communication details necessary for sending printed information to the payee.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
COM			Segment identifier	Yes	Yes	
		C076	Communication contact	Yes	Yes	
	+	3148	Communication number [1]	Yes	Yes	an..512

Segment	Sep.	Item	Description	Req.	Used	Format
	:	3155	Communication channel qualifier [2]	Yes	Yes	an..3
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the communication number.
2. This element contains the communication channel. Acceptable values: `MA` for mail, `TM` for telemail, `FX` for facsimile, and `EM` for e-mail.

A.3.5.6.4 Payor NAD segment

The NAD segment identifies a specific party involved in the extended payment order. The first occurrence of this segment identifies the payor.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
NAD			Segment identifier	Yes	Yes	
	+	3035	Party qualifier [1]	Yes	Yes	an..3
		C082	Party identification details	No	Yes	
	+	3039	Party identification [2]	Yes	Yes	an..35
	:	1131	Code list qualifier	No	No	an..3
	:	3055	Code list resp. agency (coded)	No	No	an..3
		C058	Name and address [3]	No	Yes	
	+	3124	Name and address line	Yes	Yes	an..35
	:	3124	Name and address line	No	Yes	an..35
	:	3124	Name and address line	No	Yes	an..35
	:	3124	Name and address line	No	Yes	an..35
	:	3124	Name and address line	No	Yes	an..35
		C080	Party name [4]	No	Yes	
	+	3036	Party name	Yes	Yes	an..35
	:	3036	Party name	No	Yes	an..35
	:	3036	Party name	No	Yes	an..35
	:	3036	Party name	No	Yes	an..35
	:	3036	Party name	No	Yes	an..35
		C059	Street [5]	No	Yes	
	+	3042	Street and number/PO box	Yes	Yes	an..35
	:	3042	Street and number/PO box	No	Yes	an..35
	:	3042	Street and number/PO box	No	Yes	an..35
	:	3042	Street and number/PO box	No	Yes	an..35
	+	3164	City name	No	Yes	an..35

Segment	Sep.	Item	Description	Req.	Used	Format
	+	3229	Country subentity identifier	No	Yes	an..9
	+	3251	Postal code	No	Yes	an..9
	+	3207	Country (coded) [6]	No	Yes	an..3
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the function of the party identified. Acceptable value: `PR` for payor.
2. If the payment is intercompany, this element must contain an entity ID stored in CMM.
3. For proper handling in CMM, use composite elements C080 and C059 rather than C058.
4. If the payment method is check, this composite element is required.
5. If the payment method is check, this composite element is required.
6. This element must contain a valid ISO country code.

The following is an example of this segment:

```
NAD+PR+99998888:160:92++PAYEE+123 MAIN STREET+CHICAGO+IL+58018+US’
```

A.3.5.6.5 Payor CTA segment

The payor CTA segment identifies the contact person for the payor.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
CTA			Segment identifier	Yes	Yes	
	+	3139	Contact function (coded) [1]	No	Yes	an..3
	+	C056	Department or employee details	No	Yes	
	:	3413	Department or employee identifier [2]	No	Yes	an..17
	:	3412	Department or employee [3]	No	Yes	an..35
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element must contain `IC` (information contact).
2. This element contains the internal identification code.
3. This element contains the department or person within an organizational entity.

The following is an example of this segment:

```
CTA+IC+:T Smith’
```

A.3.5.6.6 Payor COM segment

The payor COM segment provides the communication details necessary for sending printed information to the payor.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
COM			Segment identifier	Yes	Yes	
		C076	Communication contact	Yes	Yes	
	+	3148	Communication number [1]	Yes	Yes	an..512
	:	3155	Communication channel qualifier [2]	Yes	Yes	an..3
	'		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the communication number.
2. This element contains the communication channel. Acceptable values: `MA` for mail, `TM` for telemail, `FX` for facsimile, and `EM` for e-mail.

A.3.5.7 EDIFACT PAYEXT segment group 5

Segment group 5 contains instructions and related information from the originator of the payment to the parties identified in segment groups 3 and 4. This group is not required and can be repeated four times. It contains the following segments:

ID	Name	Required	Repetitions
INP	Parties to instruction	Yes	1
FTX	Free text	No	1
DTM			

Note: This group is not supported by CMM.

A.3.5.7.1 INP segment

The INP segment specifies parties to an instruction and, where relevant, the instruction.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
INP			Segment identifier	Yes	No	
		C849	Parties to instruction	No	No	
	+	3301	Party enacting instruction identifier [1]	Yes	No	an..17
	:	3285	Recipient of instruction identifier	No	No	an..17
		C522	Instruction	No	No	
	+	4403	Instruction qualifier	Yes	No	an..3
	:	4401	Instruction (coded)	No	No	an..3
	'		End of segment marker	Yes	No	

Table notes:

1. This element contains the party to which the instruction is addressed. Acceptable values: `REC` for receiving bank and `ACC` for target bank.

The following is an example of this segment:

```
INP+REC'
```

A.3.5.7.2 FTX segment

The FTX segment contains bank-to-bank information in free-text format.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
FTX			Segment identifier	Yes	No	
	+	4451	Text subject qualifier [1]	Yes	No	an..3
	+	4453	Text function (coded)	No	No	an..3
		C107	Text reference	No	No	
	+	4441	Free text (coded)	Yes	No	an..3
	:	1131	Code list qualifier	No	No	an..3
	:	3055	Code list resp. agency (coded)	No	No	an..3
		C108	Text literal	No	No	
	+	4440	Free text [2]	Yes	No	an..70
	:	4440	Free text [2]	No	No	an..70
	:	4440	Free text [2]	No	No	an..70
	:	4440	Free text [2]	No	No	an..70
	:	4440	Free text [2]	No	No	an..70
	+	3453	Language (coded)	No	No	an..3
	'		End of segment marker	Yes	No	

Table notes:

1. This element contains the type of information that will be carried in the FTX segment. Acceptable value: `AAG` for party instructions.
2. This element contains bank-to-bank instructions. A maximum of five free-text elements can be accepted.

The following is an example of this segment:

```
FTX+AAG++BANK-TO-BANK INSTRUCTIONS:BANK-TO-BANK INSTRUCTIONS'
```

A.3.5.8 EDIFACT PAYEXT segment group 7

Segment group 7 provides information relating to the purpose of the payment from the ordering customer to the beneficiary. In this implementation, this group is used for remittance advice details in either structured or unstructured form. This group is not required and can be repeated once. It contains the following segments:

ID	Name	Required	Repetitions
PRC	Process identification	Yes	1

ID	Name	Required	Repetitions
FTX	Free text	No	5

Note: Segment group 7 also contains segment group 8. For more information on segment group 8, see A.3.5.9 EDIFACT PAYEXT segment group 8 on page 193.

A.3.5.8.1 PRC segment

This segment identifies the type of process on the beneficiary's side of the payment.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
PRC			Segment identifier	Yes	Yes	
		C242	Process type and description	Yes	Yes	
	+	7187	Process type identification [1]	Yes	Yes	an..17
	:	1131	Code list qualifier	No	Yes	an..3
	:	3055	Code list resp. agency (coded)	No	Yes	an..3
	'		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the type of process required. Acceptable values: `11` for processing of unstructured information and `8` for processing of structured information. CMM only supports structured remittance advice information.

The following is an example of this segment:

```
PRC+8'
```

A.3.5.8.2 FTX segment

The FTX segment specifies free-form or coded text information relevant to the message. It allows the ordering customer to provide remittance advice information in a free-text format.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
FTX			Segment identifier	Yes	No	
	+	4451	Text subject qualifier [1]	Yes	No	an..3
	+	4453	Text function (coded)	No	No	an..3
		C107	Text reference	No	No	
	+	4441	Free text (coded)	Yes	No	an..3
	:	1131	Code list qualifier	No	No	an..3
	:	3055	Code list resp. agency (coded)	No	No	an..3
		C108	Text literal	No	No	
	+	4440	Free text [2]	Yes	No	an..70
	:	4440	Free text [2]	No	No	an..70

Segment	Sep.	Item	Description	Req.	Used	Format
	:	4440	Free text [2]	No	No	an..70
	:	4440	Free text [2]	No	No	an..70
	:	4440	Free text [2]	No	No	an..70
	+	3453	Language (coded)	No	No	an..3
	’		End of segment marker	Yes	No	

Table notes:

1. This element contains the type of information that will be carried in the FTX segment. Acceptable value: `PMD` for party detail/remittance information.
2. These elements contain free text and can be used in the following ways: as headings (the first element contains `HEADER`, and the second element contains the headings for the remittance advice details); as 20-character remittance detail lines (the first element contains `DETAIL 120`, the second element contains the first 70 characters of the first detail line, the third element contains the remaining 50 characters of the first detail line, the fourth element contains the first 70 characters of the second detail line if any, and the fifth line contains the remaining 50 characters of the second detail line if any); or as 70-character remittance detail lines (all five elements contain remittance detail information).

A.3.5.9 EDIFACT PAYEXT segment group 8

Segment group 8 provides details of all documents (for example, invoices) to which the extended payment order refers. This group includes information on the monetary amounts for each document. This group is not required and can be repeated 9,999 times. It contains the following segments:

ID	Name	Required	Repetitions
DOC	Document/message details	Yes	1
MOA	Monetary amount	No	5
DTM	Date/time/period	No	5
RFF	Reference	No	1

A.3.5.9.1 DOC segment

The DOC segment identifies the documents, meaningful to the beneficiary, to which the extended payment order relates.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
DOC			Segment identifier	Yes	Yes	
		C002	Document/message name	Yes	Yes	
	+	1001	Document/message name (coded) [1]	No	Yes	an..3
	:	1131	Code list qualifier	No	No	an..3
	:	3055	Code list resp. agency (coded)	No	No	an..3
	:	1000	Document/message name	No	No	an..35
		C503	Document/message details	No	Yes	
	+	1004	Document/message number [2]	No	Yes	an..35

Segment	Sep.	Item	Description	Req.	Used	Format
	:	1373	Document/message status (coded)	No	No	an..3
	:	1366	Document/message source	No	No	an..35
	:	3453	Language (coded)	No	No	an..3
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the type of related document in coded form. Acceptable values: `380` for commercial invoice, `381` for credit note, and `383` for debit note.
2. This element contains the invoice/credit note number.

The following is an example of this segment:

```
DOC+380+123456’
```

A.3.5.9.2 MOA segment

The MOA segment specifies the monetary amount of each referenced document and the relevant currency, if necessary.

Note: There may be one or two occurrences of this segment. The first occurrence must always contain the amount due/payable and the optional second occurrence can contain the amount remitted. The currency of the latter, if given, must be the same as that of the former.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
MOA			Segment identifier	Yes	Yes	
		C516	Monetary amount	Yes	Yes	
	+	5025	Monetary amount type qualifier [1]	Yes	Yes	an..3
	:	5004	Monetary amount	No	Yes	n..18
	:	6345	Currency (coded) [2]	No	Yes	an..3
	:	6343	Currency qualifier	No	No	an..3
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the type of amount. Acceptable values: `9` for amount due/amount payable, `12` for amount remitted, and `11` for amount paid.
2. This element must contain the ISO 4217 currency code.

The following is an example of this segment:

```
MOA+9:100:USD’
```

A.3.5.9.3 DTM segment

The DTM segment specifies the date of the reference document and indicates any other relevant dates applicable.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
DTM			Segment identifier	Yes	Yes	
		C507	Date/time/period	Yes	Yes	
	+	2005	Date/time/period qualifier [1]	Yes	Yes	an..3
	:	2380	Date/time/period [2]	No	Yes	an..35
	:	2379	Date/time/period format qualifier [3]	No	Yes	an..3
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the date type. Acceptable value: `137` for document/message date/time.
2. This element contains the document message date/time and must be in the YYYYMMDD format.
3. This element contains the date format and must always be `102` (for the YYYYMMDD format).

The following is an example of this segment:

```
DTM+171:19971201:102’
```

A.3.5.9.4 RFF segment

The RFF segment identifies a transaction from the ordering customer to the beneficiary and/or from the order customer to the ordered bank.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
RFF			Segment identifier	Yes	Yes	
		C506	Reference	Yes	Yes	
	+	1153	Reference qualifier [1]	Yes	Yes	an..3
	:	1154	Reference number	No	Yes	an..35
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the type of reference in coded form. Acceptable value: `ACE` for related document number.

The following is an example of this segment:

```
RFF+RA:0123’
```

A.3.5.10 EDIFACT PAYEXT segment group 15

Segment group 15 specifies the details of any authentication (validation) procedure applied to the PAYEXT message. This group is not required and can be repeated once. It contains the following segments:

ID	Name	Required	Repetitions
AUT	Authentication result	Yes	1
DTM	Date/time/period	No	1

A.3.5.10.1 AUT segment

The AUT segment specifies the details of any authentication (validation) procedures applied to the extended payment order. This segment is composed of the validation result optionally followed by the validation key identification.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
AUT			Segment identifier	Yes	Yes	
	+	9280	Validation result	Yes	Yes	an..35
	+	9282	Validation key identifier	Yes	Yes	an..35
	’		End of segment marker	Yes	Yes	

The following is an example of this segment:

```
AUT+9876532+1515’
```

A.3.5.10.2 DTM segment

The DTM segment identifies the date of authentication.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
DTM			Segment identifier	Yes	Yes	
		C507	Date/time/period	Yes	Yes	
	+	2005	Date/time/period qualifier [1]	Yes	Yes	an..3
	:	2380	Date/time/period [2]	No	Yes	an..35
	:	2379	Date/time/period format qualifier [3]	No	Yes	an..3
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the date type. Acceptable value: `218` for authentication/validation date/time.
2. This element contains the authentication/validation date/time and must be in the YYYYMMDD format.
3. This element contains the date format and must always be `102` (for the YYYYMMDD format).

The following is an example of this segment:

```
DTM+218:19980115:102’
```

A.3.5.11 EDIFACT PAYEXT end of message

The end of message specifies the end of the message. This group is required and can be repeated once. It contains the following segment:

ID	Name	Required	Repetitions
UNT	Message trailer	Yes	1

A.3.5.11.1 UNT segment

The UNT segment specifies the end of the message, providing the total number of segments and control reference number of the message, as quoted in the UNH segment.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
UNT			Segment identifier	Yes	Yes	
	+	0074	Number of segments in message [1]	Yes	Yes	n..6
	+	0062	Message reference number [2]	Yes	Yes	an..14
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the total number of segments, starting with the UNH segment and ending with the UNT segment, that have been used in the message. This information provides the receiver with the necessary data to ensure that the entire message has been received.
2. This element contains a message by a unique number within a range of messages (for example, the first message is 1; the second, 2; the third, 3; and so on). The value for element 0062 in this segment must be the same as the value in element 0062 in the corresponding UNH segment.

The following is an example of this segment:

```
UNT+78+1’
```

A.3.5.12 EDIFACT PAYEXT end of interchange

The end of interchange specifies the end of the interchange. This group is required and can be repeated once. It contains the following segment:

ID	Name	Required	Repetitions
UNZ	Interchange trailer	Yes	1

A.3.5.12.1 UNZ segment

The UNZ segment ends and checks the completeness of an interchange.

The following table presents the elements of this segment:

Segment	Sep.	Item	Description	Req.	Used	Format
UNZ			Segment identifier	Yes	Yes	
	+	0036	Interchange control count [1]	Yes	Yes	n..6
	+	0020	Interchange control reference [2]	Yes	Yes	an..14
	’		End of segment marker	Yes	Yes	

Table notes:

1. This element contains the number of messages within the interchange, where the start of each message is denoted by a UNH segment and the end of each message is denoted by a UNT segment.
2. This element contains a unique reference for the message, assigned by the sender. This reference must contain the same value as element 0020 in the UNB segment.

The following is an example of this segment:

```
UNZ+22+INTERCHANGEREF'
```

A.4 Bank message import formats

CMM supports the following standard formats for bank message imports:

Type	Wallstreet formats	EDIFACT formats	SWIFT formats	Other formats
Bank message	• XML bank message • XML transaction acknowledgement	• BANSTA [1] • CONTRL [1]	• ACK/NACK [1] • ISO 20022 XML payment status	• ANSI X12 824 [1] • ANSI X12 997 [1] • CFONB/AFB REJ240 [1]

Table notes:

1. For information on formats not documented in this appendix, contact Wallstreet.

A.4.1 Wallstreet XML bank message

The following is an example Wallstreet XML bank message file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<transactions document_reference_id="107" format_code="ATTRS_TXN"
xmlns="http://www.trema.com/externalinterface/XMLschema">
    <transaction sequence="1" type="bankmessage">
       <attribute name="customer_reference_id" value="123456"/>
       <attribute name="bank_reference_id" value="123456"/>
       <attribute name="message_text_desc" value="Description"/>
    </transaction>
    <transaction sequence="2" type="bankmessage">
       <attribute name="customer_reference_id" value="123457"/>
       <attribute name="bank_reference_id" value="123457"/>
       <attribute name="message_text_desc" value="Description"/>
    </transaction>
</transactions>
```

This file contains three basic elements:

Element	Description
`transactions`	The file's identifier, format code, and schema.
`transaction`	An individual transaction in the file.
`attribute`	An individual attribute of a transaction.

Note: The elements must be nested as displayed in the above table, and the file must validate against the Wallstreet XML transaction schema. For more information on the Wallstreet XML transaction schema, see A.11 Wallstreet XML schemas on page 292.

A.4.1.1 transactions element

A Wallstreet XML bank message file contains only one `transactions` element.

The `transactions` element defines the file's identifier, format code, and schema as specified in the element's three attributes:

Attribute	Acceptable values
`document_reference_ID`	[A unique identifier for the file]
`format_code`	`ATTRS_TXN`
`xmlns`	`http://www.trema.com/externalinterface/XMLschema`

A.4.1.2 transaction elements

A Wallstreet XML bank message file can contain multiple transactions with each transaction represented by a `transaction` element.

Each `transaction` element defines its transaction's sequence number and type as specified in the element's two attributes:

Attribute	Acceptable values
`sequence`	[A unique sequence number for the transaction (In a typical file, the first transaction is numbered `1`; the second, `2`; the third, `3`; and so on.)]
`type`	`bankmessage`

A.4.1.3 attribute elements

Each `transaction` element contains a set of `attribute` elements. These elements define the transactions' attributes and the values of those attributes.

Each `attribute` element has two attributes:

Attribute	Acceptable values
`name`	[The attribute's name as specified in B.1 Import attributes on page 312 (The value you provide in this attribute must exactly match the value in the "Name" column in B.1 Import attributes on page 312.)]
`value`	[The attribute's value (The value you provide in this attribute must comply with the type, size, and other requirements specified in B.1 Import attributes on page 312.)]

As noted in B.1 Import attributes on page 312, some attributes are required and must be provided with every transaction while others are not required.

You can specify the attributes in any order in the Wallstreet XML bank message files. In addition, you do not need to specify the same set of attributes in every transaction in a file. For example, one transaction can include the `description` attribute while all others in the file do not.

A.4.2 Wallstreet XML transaction acknowledgement

The following is an example Wallstreet XML transaction acknowledgement file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<transactions xmlns="http://www.trema.com/externalinterface/XMLschema"
format_code="TREMA_ACK" document_reference_id = "test_1">
    <transaction sequence="1" type="acknowledgement">
       <transaction_id>26</transaction_id>
       <customer_reference_number>ACH32934298349834</customer_reference_number>
       <external_reference_number>ext_1</external_reference_number>
       <transaction_success>false</transaction_success>
       <rejection_reason>invalid beneficiary bank account number</rejection_reason>
    </transaction>
```

```
    <transaction sequence="2" type="acknowledgement">
       <transaction_id>27</transaction_id>
       <customer_reference_number>FT89438904384585489</customer_reference_number>
       <external_reference_number>ext_2</external_reference_number>
       <transaction_success>true</transaction_success>
    </transaction>
</transactions>
```

This file contains two basic elements:

Element	Description
transactions	The file's identifier, format code, and schema.
transaction	An individual transaction acknowledgement in the file.

Note: The elements must be nested as displayed in the above table, and the file must validate against the Wallstreet XML transaction acknowledgement schema. For more information on the Wallstreet XML transaction acknowledgment schema, see A.11 Wallstreet XML schemas on page 292.

A.4.2.1 transactions element

A Wallstreet XML transaction acknowledgement file contains only one `transactions` element.

The `transactions` element defines the file's identifier, format code, and schema as specified in the element's three attributes:

Attribute	Acceptable values
xmlns	http://www.trema.com/externalinterface/XMLschema
format_code	TREMA_ACK
document_reference_id	[A unique identifier for the file]

A.4.2.2 transaction elements

A Wallstreet XML transaction acknowledgement file can contain multiple transaction acknowledgements with each transaction acknowledgement represented by a `transaction` element.

Each `transaction` element defines its transaction acknowledgement's sequence number and type as specified in the element's two attributes:

Attribute	Acceptable values
sequence	[A unique sequence number for the transaction acknowledgement (In a typical file, the first transaction acknowledgement is numbered 1; the second, 2; the third, 3; and so on.)]
type	acknowledgement

In addition, each `transaction` element defines its transaction acknowledgement's details as specified in the element's child elements:

Element	Acceptable values
transaction_id	[The transaction's ID]
customer_reference_number	[The transaction's customer reference number]
external_reference_number	[The transaction's external reference number]

Element	Acceptable values
transaction_success	true if the transaction was accepted false if the transaction was rejected
rejection_reason	[The reason for the rejection of the transaction.] Note: This element is only required if the value of the transaction_success element is false.

A.4.3 ISO 20022 XML payment status

The ISO 20022 XML standard is an effort of financial institution, corporations, and vendors to define standard message formats for financial transactions. SWIFT is heavily involved in the standard by providing expertise and facilitating meetings.

Using CMM's XML template tool, Wallstreet has implemented a subset of the initial ISO 20022 XML standard formats—including the payment status format. You can use Wallstreet's initial work as the basis for implementing the ISO 20022 XML standard. You can also customize the formats to accommodate your banks' unique interpretations of the standard.

The ISO 20022 XML payment status message is sent by an instructing agent to the previous party in the payment chain. It is used to inform the party of the positive or negative status of an instruction (either a single transaction or a file). It is also used to report on a pending instruction.

The ISO 20022 XML payment status message consists of the following sections:

Name	Required	Number of occurrences	Contents
Group header	Yes	One	Element such as MessageIdentification and CreationDateAndTime
Original group information and status	Yes	One	Elements such as OriginalMessageIdentification, OriginalMessageNameIdentification, and GroupStatus
Transaction information and status	No	Multiple	Elements referencing the original instruction

A.4.3.1 ISO 20022 XML payment status format

The following table presents the format of ISO 20022 XML payment status messages:

Message item	XML tag	Mult/format	Data type	CMM mapping and posting
Group header	<GrpHdr>	N/A	N/A	N/A
MessageIdentification	<MsgID>	[1..1]	Text	Map to file_id
CreationDateTime	<CreDtTm>	[1..1]	Date/time	Map to file_creation_time
InitiatingParty	<InitgPty>	[0..1]	N/A	N/A
ForwardingAgent	<FwdgAgt>	[0..1]	Indicator	N/A
DebtorAgent	<DbtrAgt>	[0..1]	N/A	N/A
CreditorAgent	<CdtrAgt>	[0..1]	N/A	N/A

Message item	XML tag	Mult/format	Data type	CMM mapping and posting
InstructingAgent	<InstgAgt>	[0..1]	N/A	N/A
InstructedAgent	<InstdAgt>	[0..1]	N/A	N/A
OriginalGroupInformationAndStatus	<OrgnlGrpInfAndSts>	N/A	N/A	N/A
OriginalMessageIdentification	<OrgnlMsgId>	[1..1]	Text	Map to orig_file_export_reference_id
NetworkFileName	<NtwkFileNm>	[1..1]	Text	N/A
OriginalMessageNameIdentification	<OrgnlMsgNmID>	[1..1]	N/A	Post one of the following: • pain.001.001.02 • pain.008.001.01 • pain.007.001.01 • pain.006.001.01
OriginalCreationDateTime	<OrgnlCreDtTm>	[0..1]	Date/time	N/A
FileOriginator	<FileOrgtr>	[0..1]	Text	N/A
OriginalNumberOfTransactions	<OrgnlNbOfTxs>	[0..1]	Text	N/A
OriginalControlSum	<OrgnlCtrlSum>	[0..1]	Quantity	N/A
GroupStatus	<GrpSts>	[0..1]	Code	Map to external_message_code and message_text_desc
StatusReasonInformation	<StsRsnInf>	[0..n]	N/A	N/A
StatusOriginator	<StsOrgtr>	[0..1]	N/A	N/A
StatusReason	<StsRsn>	[0..1]	N/A	N/A
Code	<Cd>	[1..1]	Code	Map to message_text_desc
Proprietary	<Prtry>	[1..1]	Text	Map to message_text_desc

Message item	XML tag	Mult/format	Data type	CMM mapping and posting
AdditionalStatusReasonInformation	<AddtlStsRsnInf>	[0..n]	Text	N/A
NumberOfTransactionsPerStatus	<NbOfTxsPerSts>	[0..n]	N/A	N/A
DetailedNumberOfTransactions	<DtldNbOfTxs>	[1..1]	Text	N/A
DetailedStatus	<DtldSts>	[1..1]	Code	Map to message_text_desc
DetailedControlSum	<DtldCtrlSum>	[0..1]	Quantity	N/A
TransactionInformationAndStatus	<TxInfAndSts>	[0..n]	N/A	N/A
StatusIdentification	<StsId>	[0..1]	Text	N/A
OriginalPaymentInformationIdentification	<OrgnlPmtInfId>	[0..1]	Text	N/A
OriginalInstructionIdentification	<OrgnlInstrId>	[0..1]	Text	N/A
OriginalEndToEndIdentification	<OrgnlEndToEndId>	[0..1]	Text	Map to customer_reference_id
OriginalTransactionIdentification	<OrgnlTxId>	[0..1]	Text	Map to customer_reference_id
TransactionStatus	<TxSts>	[0..1]	Code	Map to external_message_code and message_text_desc
StatusReasonInformation	<StsRxnInf>	[0..n]	N/A	N/A
StatusOriginator	<StsOrgtr>	[0..1]	N/A	N/A
StatusReason	<StsRsn>	[0..1]	N/A	N/A
Code	<Cd>	[1..1]	Code	Map to message_text_desc
Proprietary	<Prtry>	[1..1]	Text	Map to message_text_desc
AdditonalStatusReasonInformation	<AddtlStsRsnInf>	[0..n]	Text	N/A

Message item	XML tag	Mult/format	Data type	CMM mapping and posting
ChargesInformation	<ChrgsInf>	[0..n]	N/A	N/A
AcceptanceDateTime	<AccptncDtTm>	[0..1]	Date/time	N/A
InstructingAgent	<InstgAgt>	[0..1]	N/A	N/A
InstructedAgent	<InstdAgt>	[0..1]	N/A	N/A
OriginalTransactionReference	<OrgnlTxRef>	[0..1]	N/A	N/A

A.4.3.2 ISO 20022 XML payment status parsing logic

This section defines parsing logic for ISO 20022 XML payment status messages.

A.4.3.2.1 OriginalGroupInformationAndStatus parsing logic

The following is the logic to parse OriginalGroupInformationAndStatus:

1. This status is related to the import/export status log; it is a file-level bank message. CMM creates one or more bank messages based on this group, and they are all linked with an export active in the import/export log. CMM sets `orig_file_export_reference_id` with the value of OriginalMessageIdentification.
2. If GroupStatus are not `RJCT` and are not empty:

   CMM**a.** parses the general GroupStatus:

   - CMM sets `external_message_code` with the value of GroupStatus.
   - CMM sets `message_text_desc` with the definition of the code of GroupStatus.
   - CMM builds a bank message record.

   CMM**b.** parses NumberOfTransactionsPerStatus.

If NumberOfTransactionsPerStatus exists:

- CMM sets `external_message_code` with DetailedStatus.
- CMM sets `message_text_desc` with the definition of the code of DetailedStatus.
- CMM builds a new bank messages record.

3. If GroupStatus are `RJCT`:

CMM**a.** parses the general GroupStatus:

- CMM sets `external_message_code` with `RJCT`.
- CMM sets `message_text_desc` with `Payment initiation or individual transaction included in the payment initiation has been rejected..`
- CMM builds a bank message record.

CMM**b.** parses StatusReasonInformation:

- CMM does one of the following:
  - CMM set `external_message_code` with `RJCT`.
  - CMM sets `message_text_desc` with the definition of the code of StatusReason.
- CMM builds a new bank messages record.

4. If `GroupStatus` are empty:

CMM**a.** parses StatusReasonInformation:

- CMM does one of the following:
  - CMM sets `external_message_code` with `RJCT`.
  - CMM sets `message_text_desc` with the definition of the code of StatusReason.
- CMM builds a new bank message record.

CMM**b.** parses NumberOfTransactionsPerStatus:

- If NumberOfTransactionsPerStatus exists:
  - CMM sets `external_message_code` with DetailedStatus.
  - CMM sets `message_text_desc` with the definition of the code of DetailedStatus.
- CMM builds a new bank message record.

A.4.3.2.2 TransactionInformationAndStatus parsing logic

The following is the logic to parse TransactionInformationAndStatus:

1. If TransactionInformationAndStatus exists:

   CMM**a.** parses OriginalEndtoEndIdentification:

   - CMM validates if OriginalEndToEndIdentification is present since it is mandatory.
   - If valid, CMM sets `customer_reference_id` with OriginalEndToEndIdentification.

   CMM**b.** parses general TransactionStatus

   - CMM sets `external_message_code` with the value of TransactionStatus
   - CMM sets `message_text_desc` with the definition of the code of TransactionStatus
   - CMM builds a bank messages record.

   CMM**c.** parses StatusReasonInformation:

   - CMM sets `external_message_code` with `RJCT`.
   - CMM sets `message_text_desc` with the definition of the code of StatusReason.
   - CMM builds a new bank message record.

A.4.3.3 ISO 20022 XML payment status valid codes

This section defines valid codes for ISO 20022 XML payment status messages.

A.4.3.3.1 GroupStatus valid codes

The following table presents valid codes for GroupStatus:

Code	Name	Definition
`ACCP`	AcceptedCustomerProfile	Preceding check of technical validation was successful. Customer profile check was also successful. This includes the assessment of the static risks.
`ACCR`	AcceptedCancellationRequest	Cancellation is accepted.
`ACSC`	AcceptedSettlementCompleted	Settlement on the debtor's account has been completed. Usage: This can be used by the first agent to report to the debtor that the transaction has been completed. Warning: This status is provided for transaction status reasons, not for financial information. It can only be used after bilateral agreement.
`ACSP`	AcceptedSettlementInProcess	All preceding checks, such as technical validation and customer profile, were successful. Dynamic risk assessment is now also successful; therefore, the payment initiation has been accepted for execution.
`ACTC`	AcceptedTechnicalValidation	Authentication and syntactical and semantical validation are successful.
`ACWC`	AcceptedWithChange	Instruction is accepted but a change will be made (for example, date, remittance not sent).
`PART`	PartiallyAccepted	A number of transactions have been accepted, whereas another number of transactions have not yet achieved "accepted" status.
`PDNG`	Pending	Payment initiation or an individual transaction included in the payment initiation is pending. Further checks and status updates will be performed.

Code	Name	Definition
RCVD	Received	Payment initiation has been received by the receiving agent.
RJCT	Rejected	Payment initiation or an individual transaction included in the payment initiation has been rejected.

A.4.3.3.2 Group StatusReason valid codes

The following table presents valid codes for group StatusReason:

Code	Name	Definition
AC01	IncorrectAccountNumber	Format of the account number specified is not correct.
AC04	ClosedAccountNumber	Account number specified has been closed on the receiver's books.
AC06	BlockedAccount	Account specified is blocked, prohibiting posting of transactions against it.
AG01	TransactionForbidden	Transaction forbidden on this type of account (formerly NoAgreement).
AG02	InvalidBankOperationCode	Bank operation code specified in the message is not valid for receiver.
AM01	ZeroAmount	Specified message amount is equal to zero.
AM02	NotAllowedAmount	Specified transaction/message amount is greater than allowed maximum.
AM03	NotAllowedCurrency	Specified message amount is in an non processable currency outside of existing agreement.
AM04	InsufficientFunds	Amount of funds available to cover specified message amount is insufficient.
AM05	Duplication	Message appears to have been duplicated.
AM06	TooLowAmount	Specified transaction amount is less than agreed minimum.
AM07	BlockedAmount	Amount specified in message has been blocked by regulatory authorities.
AM09	WrongAmount	Amount received is not the amount agreed upon or expected.
AM10	InvalidControlSum	Sum of instructed amounts does not equal the control sum.
BE01	InconsistentWithEndCustomer	Identification of end customer is not consistent with associated account number (formerly CreditorConsistency).
BE04	MissingCreditorAddress	Specification of creditor's address, which is required for payment, is missing/not correct (formerly IncorrectCreditorAddress).
BE05	UnrecognisedInitiatingParty	Party who initiated the message is not recognized by the end customer.
BE06	UnknownEndCustomer	End customer specified is not known at associated sort/national bank code or no longer exists in the books.
BE07	MissingDebtorAddress	Specification of debtor's address, which is required for payment, is missing/not correct.
DT01	InvalidDate	Invalid date (for example, wrong settlement date).

Code	Name	Definition
ED01	CorrespondentBankNotPossible	Correspondent bank not possible.
ED03	BalanceInfoRequested	Balance of payments complementary information is requested.
ED05	SettlementFailed	Settlement of the transaction has failed.
MD01	NoMandate	Mandate is canceled or invalid.
MD02	MissingMandatoryInformationInMandate	Mandate related information data required by the scheme is missing.

A.4.3.3.3 DetailedStatus valid codes

The following table presents valid codes for DetailedStatus:

Code	Name	Definition
ACCP	AcceptedCustomerProfile	Preceding check of technical validation was successful. Customer profile check was also successful. This includes the assessment of the static risks.
ACCR	AcceptedCancellationRequest	Cancellation is accepted.
ACSC	AcceptedSettlementCompleted	Settlement on the debtor's account has been completed. Usage: This can be used by the first agent to report to the debtor that the transaction has been completed. Warning: This status is provided for transaction status reasons, not for financial information. It can only be used after bilateral agreement.
ACSP	AcceptedSettlementInProcess	All preceding checks, such as technical validation and customer profile, were successful. Dynamic risk assessment is now also successful; therefore, the payment initiation has been accepted for execution.
ACTC	AcceptedTechnicalValidation	Authentication and syntactical and semantical validation are successful.
ACWC	AcceptedWithChange	Instruction is accepted but a change will be made (for example, date, remittance not sent).
PDNG	Pending	Payment initiation or an individual transaction included in the payment initiation is pending. Further checks and status updates will be performed.
RJCT	Rejected	Payment initiation or an individual transaction included in the payment initiation has been rejected.

A.4.3.3.4 TransactionStatus valid codes

The following table presents valid codes for TransactionStatus:

Code	Name	Definition
ACCP	AcceptedCustomerProfile	Preceding check of technical validation was successful. Customer profile check was also successful. This includes the assessment of the static risks.
ACCR	AcceptedCancellationRequest	Cancellation is accepted.

Code	Name	Definition
ACSC	AcceptedSettlementCompleted	Settlement on the debtor's account has been completed. Usage: This can be used by the first agent to report to the debtor that the transaction has been completed. Warning: This status is provided for transaction status reasons, not for financial information. It can only be used after bilateral agreement.
ACSP	AcceptedSettlementInProcess	All preceding checks, such as technical validation and customer profile, were successful. Dynamic risk assessment is now also successful; therefore, the payment initiation has been accepted for execution.
ACTC	AcceptedTechnicalValidation	Authentication and syntactical and semantical validation are successful.
ACWC	AcceptedWithChange	Instruction is accepted but a change will be made (for example, date, remittance not sent).
PDNG	Pending	Payment initiation or an individual transaction included in the payment initiation is pending. Further checks and status updates will be performed.
RJCT	Rejected	Payment initiation or an individual transaction included in the payment initiation has been rejected.

A.4.3.3.5 Transaction StatusReason valid codes

The following table presents valid codes for transaction StatusReason:

Code	Name	Definition
AC01	IncorrectAccountNumber	Format of the account number specified is not correct.
AC04	ClosedAccountNumber	Account number specified has been closed on the receiver's books.
AC06	BlockedAccount	Account specified is blocked, prohibiting posting of transactions against it.
AG01	TransactionForbidden	Transaction forbidden on this type of account (formerly NoAgreement).
AG02	InvalidBankOperationCode	Bank operation code specified in the message is not valid for receiver.
AM01	ZeroAmount	Specified message amount is equal to zero.
AM02	NotAllowedAmount	Specified transaction/message amount is greater than allowed maximum.
AM03	NotAllowedCurrency	Specified message amount is in an non processable currency outside of existing agreement.
AM04	InsufficientFunds	Amount of funds available to cover specified message amount is insufficient.
AM05	Duplication	Message appears to have been duplicated.

Code	Name	Definition
AM06	TooLowAmount	Specified transaction amount is less than agreed minimum.
AM07	BlockedAmount	Amount specified in message has been blocked by regulatory authorities.
AM09	WrongAmount	Amount received is not the amount agreed upon or expected.
AM10	InvalidControlSum	Sum of instructed amounts does not equal the control sum.
BE01	InconsistentWithEndCustomer	Identification of end customer is not consistent with associated account number (formerly CreditorConsistency).
BE04	MissingCreditorAddress	Specification of creditor's address, which is required for payment, is missing/not correct (formerly IncorrectCreditorAddress).
BE05	UnrecognisedInitiatingParty	Party who initiated the message is not recognized by the end customer.
BE06	UnknownEndCustomer	End customer specified is not known at associated sort/national bank code or no longer exists in the books.
BE07	MissingDebtorAddress	Specification of debtor's address, which is required for payment, is missing/not correct.
DT01	InvalidDate	Invalid date (for example, wrong settlement date).
ED01	CorrespondentBankNotPossible	Correspondent bank not possible.
ED03	BalanceInfoRequested	Balance of payments complementary information is requested.
ED05	SettlementFailed	Settlement of the transaction has failed.
MD01	NoMandate	Mandate is canceled or invalid.
MD02	MissingMandatoryInformationInMandate	Mandate related information data required by the scheme is missing.
MD03	InvalidFileFormatForOtherReasonThanGroupingIndicator	File format incomplete or invalid.
MD04	InvalidFileFormatForGroupingIndicator	File format incorrect in terms of grouping indicator.
MD06	RefundRequestByEndCustomer	Return of funds requested by end customer.
MD07	EndCustomerDeceased	End customer is deceased.
MS02	NotSpecifiedReasonCustomerGenerated	Reason has not been specified by end customer.
MS03	NotSpecifiedReasonAgentGenerated	Reason has not been specified by agent.
NARR	Narrative	Reason is provided as narrative information in the additional reason information.

Code	Name	Definition
RC01	BankIdentifierIncorrect	Bank identifier code specified in the message has an incorrect format (formerly IncorrectFormatForRoutingCode).
RF01	NotUniqueTransactionReference	Transaction reference is not unique within the message. UNIFI (ISO 20022) - Payments Standards - Initiation October 2006 Message: PaymentStatusReportV02 <pain.002.001.02> Page 252
TM01	CutOffTime	Associated message was received after agreed processing cut-off time.

A.5 Bank transaction and balance import formats

CMM supports the following standard formats for bank transaction and balance imports:

Type	Wallstreet formats	EDIFACT formats	SWIFT formats	Other formats
Bank statement	• XML bank statement	N/A	N/A	N/A
Bank transaction and balance	• XML bank transaction and balance	• FINSTA [1]	• MT940 • MT940 BCS • MT942 [1] • MT950 [1]	• BAI [1] • CFONB/AFB RDC120 [1]

Table notes:

1. For information on formats not documented in this appendix, contact Wallstreet.

A.5.1 Wallstreet XML bank statement

The following is an example Wallstreet XML bank statement file:

```
<?xml version="1.0" encoding="UTF-8"?>
<bankaccountstatementholder document_reference_id="test1" statement_type="P"
format_code="TREMA_BANK_STATEMENT"
xmlns="http://www.trema.com/externalinterface/XMLschema">
    <account_statement sequence="1">
       <account>
          <bank_swift_code/>
          <bank_branch_number>1234</bank_branch_number>
          <account_number>10101</account_number>
          <account_iban/>
          <account_currency>USD</account_currency>
       </account>
       <balance type="19">
          <date>2005-05-02</date>
          <amount>300.89</amount>
       </balance>
       <balance type="6">
          <date>2005-05-02</date>
          <amount>1300.89</amount>
```

```
        </balance>
        <transaction type="1">
           <customer_reference_number>test_123</customer_reference_number>
           <bank_reference_number>98548958</bank_reference_number>
           <book_date>2005-05-02</book_date>
           <value_date>2005-05-02</value_date>
           <amount>-50.89</amount>
           <description>test pmt</description>
        </transaction>
        <transaction type="2">
           <customer_reference_number>test_456</customer_reference_number>
           <bank_reference_number>985489586</bank_reference_number>
           <book_date>2005-05-02</book_date>
           <value_date>2005-05-02</value_date>
           <amount>150.89</amount>
           <description>test rct</description>
        </transaction>
        <transaction type="4">
           <customer_reference_number>test_789</customer_reference_number>
           <bank_reference_number>9854895866</bank_reference_number>
           <book_date>2005-05-02</book_date>
           <value_date>2005-05-02</value_date>
           <amount>-1150.89</amount>
        </transaction>
    </account_statement>
</bankaccountstatementholder>
```

This file contains four basic elements:

Element	Description
`bankaccountstatementholder`	The file's identifier, bank statement type, format code, and schema.
`account_statement`	An individual bank statement in the file.
`account`	The bank account to which the bank statement applies.
`balance`	An individual bank balance in the bank statement.
`transaction`	An individual bank transaction in the bank statement.

Note: The elements must be nested as displayed in the above table, and the file must validate against the Wallstreet XML bank statement schema. For more information on the Wallstreet XML bank statement schema, see A.11 Wallstreet XML schemas on page 292.

A.5.1.1 bankaccountstatementholder element

A Wallstreet XML bank statement file contains only one `bankaccountstatementholder` element.

The `bankaccountstatementholder` element defines the file's identifier, bank statement type, format code, and schema as specified in the element's three attributes:

Attribute	Acceptable values
`document_reference_id`	[A unique identifier for the file]
`statement_type`	`C` for an intraday bank statement `P` for a previous-day bank statement

Attribute	Acceptable values
format_code	TREMA_BANK_STATEMENT
xmlns	http://www.trema.com/externalinterface/XMLschema

A.5.1.2 account_statement elements

A Wallstreet XML bank statement file can contain multiple bank statements with each bank statement represented by an account_statement element.

Each account_statement element defines its bank statement's sequence number as specified in the element's attribute:

Attribute	Acceptable values
sequence	[A unique sequence number for the bank statement (In a typical file, the first bank statement is numbered 1; the second, 2; the third, 3; and so on.)]

A.5.1.3 account elements

Each account_statement element in a Wallstreet XML bank statement file must contain one account child element that defines the bank account to which the bank statement applies.

Each account element defines its bank statement's bank account as specified in the element's child elements:

Element	Acceptable values
bank_swift_code	[The SWIFT code of the bank in which the bank account is held]
bank_branch_number	[The branch number of the bank in which the bank account is held]
account_number	[The bank account's basic bank account number (BBAN)]
account_iban	[The bank account's international bank account number (IBAN)]
account_currency	[The bank account's currency]

A.5.1.4 balance elements

Each account_statement element in a Wallstreet XML bank statement file can contain one or more balance child element that define bank balances in the bank statement.

Each balance element defines its bank balance's type as specified in the element's attribute:

Attribute	Acceptable values
type	3 for opening value date 6 for closing value data 8 for opening transaction date 19 for closing transaction date 1001 for opening bank statement 1002 for closing bank statement

In addition, each `balance` element defines its bank balance's date and amount as specified in the element's child elements:

Element	Acceptable values
`date`	[The bank balance's date]
`amount`	[The bank balance's amount]

A.5.1.5 transaction elements

Each `account_statement` element in a Wallstreet XML bank statement file can contain one or more `transaction` child element that define bank transactions in the bank statement.

Each `transaction` element defines its bank transaction's type as specified in the element's attribute:

Attribute	Acceptable values
`type`	`COMMP` for commercial payment `COMMR` for commercial receipt `INTER_ACCT` for intercompany `AGGRP` for aggregate `RCNADJ` for reconciliation adjustment `POOL` for pool transfer `ZBA` for zero balance `BANKINT` for interest `BANKFEE` for bank fee `COMMFEE` for commitment fee `OVDRFEE` for overdraft fee `TAX` for tax

In addition, each `transaction` element defines its bank transaction's customer and bank reference numbers, book and value dates, amounts, and descriptions as specified in the element's child elements:

Element	Acceptable values
`customer_reference_number`	[The bank transaction's customer reference number]
`bank_reference_number`	[The bank transaction's bank reference number]
`book_date`	[The bank transaction's book (or "invoice") date]
`value_date`	[The bank transaction's value date]
`amount`	[The bank transaction's amount]
`description`	[A description of the bank transaction]

A.5.2 Wallstreet XML bank transaction and balance

The following is an example Wallstreet XML bank transaction and balance file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<transactions document_reference_id="107" format_code="ATTRS_TXN"
xmlns="http://www.trema.com/externalinterface/XMLschema">
    <transaction sequence="1" type="banktxn">
```

```
        <attribute name="currency_code" value="USD"/>
        <attribute name="amount" value="50000.00"/>
        <attribute name="txn_date" value="20-Oct-2006"/>
        <attribute name="description" value="Invoice 67003"/>
        <attribute name="customer_reference_id" value="123456"/>
        <attribute name="bank_acct_number" value="12345678"/>
        <attribute name="cpty_bank_account_number" value="87654321"/>
    </transaction>
    <transaction sequence="1" type="banktxn">
        <attribute name="currency_code" value="USD"/>
        <attribute name="amount" value="60000.00"/>
        <attribute name="txn_date" value="21-Oct-2006"/>
        <attribute name="description" value="Invoice 67004"/>
        <attribute name="customer_reference_id" value="123456"/>
        <attribute name="bank_acct_number" value="12345678"/>
        <attribute name="cpty_bank_account_number" value="87654321"/>
    </transaction>
</transactions>
```

This file contains three basic elements:

Element	Description
`transactions`	The file's identifier, format code, and schema.
`transaction`	An individual transaction in the file.
`attribute`	An individual attribute of a transaction.

Note: The elements must be nested as displayed in the above table, and the file must validate against the Wallstreet XML transaction schema. For more information on the Wallstreet XML transaction schema, see A.11 Wallstreet XML schemas on page 292.

A.5.2.1 transactions element

A Wallstreet XML bank transaction and balance file contains only one `transactions` element.

The `transactions` element defines the file's identifier, format code, and schema as specified in the element's three attributes:

Attribute	Acceptable values
`document_reference_ID`	[A unique identifier for the file]
`format_code`	`ATTRS_TXN`
`xmlns`	`http://www.trema.com/externalinterface/XMLschema`

A.5.2.2 transaction elements

A Wallstreet XML bank transaction and balance file can contain multiple transactions with each transaction represented by a `transaction` element.

Each `transaction` element defines its transaction's sequence number and type as specified in the element's two attributes:

Attribute	Acceptable values
`sequence`	[A unique sequence number for the transaction (In a typical file, the first transaction is numbered `1`; the second, `2`; the third, `3`; and so on.)]

Attribute	Acceptable values
`type`	`banktxn` `bankbalance`

A.5.2.3 attribute elements

Each `transaction` element contains a set of `attribute` elements. These elements define the transactions' attributes and the values of those attributes.

Each `attribute` element has two attributes:

Attribute	Acceptable values
`name`	[The attribute's name as specified in B.1 Import attributes on page 312 (The value you provide in this attribute must exactly match the value in the "Name" column in B.1 Import attributes on page 312.)]
`value`	[The attribute's value (The value you provide in this attribute must comply with the type, size, and other requirements specified in B.1 Import attributes on page 312.)]

As noted in B.1 Import attributes on page 312, some attributes are required and must be provided with every transaction while others are not required.

You can specify the attributes in any order in the Wallstreet XML bank transaction and balance files. In addition, you do not need to specify the same set of attributes in every transaction in a file. For example, one transaction can include the `description` attribute while all others in the file do not.

A.5.3 SWIFT MT940

The Society for Worldwide Interbank Financial Telecommunication (SWIFT) is owned by the financial industry to cooperatively supply secure, standardized messaging across more than 200 countries. SWIFT supports many different formats. A few of the common message formats are MT940, MT942, and MT950. They are used for current day, previous day, and summary reporting on bank accounts with the bank. Many of Wallstreet's CMM customers do business with banks that report bank account information in these formats.

CMM can import and export SWIFT message files. CMM's bank transaction import and export functionality supports both code-based and **XML**-template based formats.

The following table presents the major features of the SWIFT message file format:

Features	Description
Acknowledgements (imports)	These formats support positive and negative acknowledgements; however, they do not support SWIFT delivery notifications.
Transaction types	These formats support bank statements for internal (activity on bank accounts with the in-house bank) and external (activity on external bank accounts) transactions.
Bank statement file names	File names follow the CMM standard. In other words, CMM creates the file name as a concatenation of the date-time stamp and the bank name to ensure it is unique.
Interface component scope	The file is plain text only. There is no digital signature, encryption, or other security features required. If you want to use these feature, you can do so by adding them to the interchange (see 3.4 Managing interchanges on page 56).

A.5.3.1 SWIFT MT940 layout

SWIFT MT940 message files can consist of up to five blocks:

Block	Contents
1 to 3	Header information, such as the sender, receiver, and SWIFT category information. For more information, see A.5.3.2 SWIFT MT940 header blocks on page 217.
4	Main message information, usually beginning with tag 20. For more information, see A.5.3.3 SWIFT MT940 main message block on page 221.
5	Control information (This block is optional and is usually omitted.)

Note: Each block is started by an opening brace ({) and terminated by a closing brace (}). Block 4 is terminated by a hyphen followed by a closing brace (-}). Only the contents of block 4 are mappable using **XML**-based files.

The following is an example SWIFT MT940 message file:

```
{1:F01SWIFTBOFAB1X0000000000}{2:O9400000000000SWIFTBOFAB1X0000000000000000N}{3:}{4:
:20:cititest
:25:12345678
:28C:109/10
:60F:C040920USD0,
°
-}
```

A.5.3.2 SWIFT MT940 header blocks

SWIFT MT940 message files begin with three header blocks:

1. Fixed basic header block
2. Fixed application header block
3. User header block.

Header block 3 is usually omitted. CMM does not currently support this header block and ignore it if it is included in import files.

A.5.3.2.1 Fixed basic header block

The following table presents the contents of the fixed basic header block:

Content	Description	Required	Acceptable values	XML attribute
{1:	Fixed basic header identifier	Yes	{1:	Not available
a	Application identifier	Yes	Alphabetic (A to Z)	Not available
01	Service identifier	Yes	01 to 99	Not available
aaaaaaaaxxxx	ID of the ordering party bank	Yes	Alphabetic (A to Z) Note: If the ID of the ordering party bank is not 12 characters long, add Xs until it is 12 characters long.	interchange_ interchange_ sender_id

Content	Description	Required	Acceptable values	XML attribute
dddd	Number of data sets	Yes	0001 to 9999	Not available
dddddd}	Transaction number in the file	Yes	000001 to 999999	Not available

A.5.3.2.2 Fixed application input header block

The following table presents the contents of the fixed application input header block:

Content	Description	Required	Acceptable values	XML attribute
{2:	Fixed application header identifier	Yes	{2:	Not available
I	Input/output identifier	Yes	I	Not available
ddd	Message type	Yes	001 to 999	Not available
aaaaaaaaxxxx	Destination address	Yes	Alphabetic (A to Z) Note: If the destination address is not 12 characters long, add Xs until it is 12 characters long.	interchange_ interchange_ recipient_id
a	Message priority	Yes	S for user-to-system messages U for urgent messages. N for all user-to-user messages	Not available
d	Message monitoring	No	If the message priority is U, message monitoring must be 1 or 3 If the message priority is N, message monitoring can be 2	Not available
ddd	Obsolescence period Note: If the obsolescence period is provided, the message monitoring must also be provided.	No	If the message priority is U, the default obsolescence period is 3 (15 minutes) If the message priority is N, the default obsolescence period is 20 (100 minutes) Note: CMM assumes the default periods for the obsolescence period regardless of the values specified by the user.	Not available

The following is an example input application header:

```
{2:I202BANKDEFFXXXXu3003}
```

A.5.3.2.3 Fixed application output header block

The following table presents the contents of the fixed application input header block:

Content	Description	Required	Acceptable values	XML attribute
{2:	Fixed application header identifier	Yes	{2:	Not available
O	Input/output identifier	Yes	O	Not available
ddd	Message type	Yes	001 to 999	Not available
dddd	Input time	Yes	hhmm	Not available
dddddd	Date sent	Yes	yymmdd	Not available
aaaaaaaaxxxx	Message input reference	Yes	Alphabetic (A to Z) Note: If the message input reference is not 12 characters long, add Xs until it is 12 characters long.	Not available
dddddd	Date (output date local to the receiver	Yes	yymmdd	Not available
dddd	Sender session number	Yes	0000 to 9999	Not available
dddddd	Sender session ISN	Yes	000000 to 999999	Not available
dddddd	Output date	Yes	yymmdd	Not available
dddd	Output time	Yes	hhmm	Not available
a	Message priority	Yes	S for user-to-system messages U for urgent messages N for all user-to-user messages	Not available

The following is an example output application header:

```
{2:O2020900970503BANJBEBBAXXX22221234569705030900S}
```

A.5.3.2.4 User header block

The following table presents the contents of the user header block:

Content	Description	Required	Acceptable values	XML attribute
{3:	User header identifier	Yes	{3:	Not available
103:	Service identifier	No	For any message which the user submits to a FIN copy service, block 3 requires an additional field 103 which contains a three-character service identifier unique to a specific FIN copy service. By using a unique identifier, it is possible to support access to multiple services within the same CBT.	Not available

Content	Description	Required	Acceptable values	XML attribute
113:	Banking priority Note: The banking priority is assigned by the sender of the message.	No	See SWIFT documentation for details.	Not available
108:	Message user reference Note: The banking priority is assigned by the sender of the message.	No	See SWIFT documentation for details.	Not available
119:	Validation flag	No	An indicator of whether a special validation needs to be performed by FIN. The following are examples of the values this field may take: • `ISITC` indicates validation of ISITC standards. This is only used with MT 521 and MT 523. • `REMIT` identifies the presence of field 77T. This is only used in MT 103. • `RFDD` indicates that the message is a request for direct debit. This is only used in MT 104. • `STP` indicates that the message is validated according to straight through processing principles. This is only used in MT 103. Note: This list is subject to change. The following codes may be used in MT 503, 504, 505, 506, and 507 to indicate exposure type or collateral reason: • `COMM` • `CRPR` • `CRSP` • `CRTL` • `EXTD` • `FIXI` • `FORX` • `LIQU` • `OTCD` • `PAYM` • `REPO` • `SBSB` • `SCRP` • `SECL` • `SLEB` • `TCRP`.	Not available

Content	Description	Required	Acceptable values	XML attribute
115:aaa …	Addressee information	No	Information from the central institution to the receiver of the payment message. Information is input in the MT 097 FIN Copy Message Authorization/Refusal Notification in Y-Copy mode. See FIN Copy Service Description for further information.	Not available

The following is an example output application header:

```
{3:{108:PRIORITY 2}}
```

A.5.3.3 SWIFT MT940 main message block

Block 4 contains the main message of a SWIFT MT940 message file. It usually begins with tag 20.

The following table presents the available tags that can be included in block 4:

Tag	Name	Required	Contents [1]	XML attribute
20	Transaction reference number	Yes	16x	
21	Related reference	No	16x	
25	Account identification	Yes	35x	
28C	Statement number/ sequence number	Yes	5n[/5n]	
60F	Opening balance	Yes	1!a6!n3!a15d	
61 [2]	Statement line	No	6!n[4!n]2a[1!a]15d1!a3!c16x[//16x] [34x]	
86 [3]	Information to account owner	No	6×65x	
62F	Closing balance (booked funds)	No	1!a6!n3!a15d	
64	Closing available balance (available balance)	No	1!a6!n3!a15d	
65 [4]	Forward available balance	No	1!a6!n3!a15d	
86	Information to account owner	No	6×65x	

Table notes:

1. For more information on the codes used in this column, see A.13.1 SWIFT field lengths on page 303 and A.13.2 SWIFT field types on page 304.
2. Loop tag 61 for each transaction in the statement. For more information on the contents of this tag, see A.13.4 SWIFT transaction type identification code mappings on page 305.
3. Loop tag 86 for each transaction in the statement.
4. Loop tag 65 for each forward available balance.

A.5.3.4 SWIFT MT940 validation

Validation is performed during the import and parsing of the file for import files and during the export and file creation process for export files.

For the most part, the validation rules below are based on the validation rules set out by the SWIFT specifications:

Fields	Validation types	Validation
Transaction reference number (tag 20)	Must exist	
Transaction reference number (tag 20)	Length	1 to 16
Transaction reference number (tag 20)	Structure	SWIFT X character set
Related reference number (tag 21)	Optional	
Related reference number (tag 21)	Length	1 to 16
Related reference number (tag 21)	Structure	SWIFT X character set
Account identification (tag 25)	Must exist	
Account identification (tag 25)	Length	1 to 35
Account identification (tag 25)	Structure	SWIFT X character set
Statement number (tag 28C)	Must exist	
Statement number (tag 28C)	Length	1 to 5
Statement number (tag 28C)	Structure	Digits 0 to 9
Sequence number (tag 28C)	Optional	
Sequence number (tag 28C)	Length	1 to 5
Sequence number (tag 28C)	Structure	Digits 0 to 9
C/D mark (tag 60F)	Must exist	
C/D mark (tag 60F)	Length	1
C/D mark (tag 60F)	Structure	`C` or `D`
Statement start date (tag 60F)	Must exist	
Statement start date (tag 60F)	Length	6
Statement start date (tag 60F)	Structure	Digits in YYMMDD format
Opening balance currency code (tag 60)	Must exist	
Opening balance currency code (tag 60)	Length	3
Opening balance currency code (tag 60)	Structure	Letters A to Z (uppercase)
Opening balance actual amount (tag 60) for transaction date	Must exist	
Opening balance actual amount (tag 60) for transaction date	Length	1 to 15
Opening balance actual amount (tag 60) for transaction date	Structure	Digits 0 to 9 and `,` and `.`
Transaction value date (tag 61)	Must exist	
Transaction value date (tag 61)	Length	6
Transaction value date (tag 61)	Structure	Digits 0 to 9 in YYMMDD format
Transaction effective date (tag 61)	Length	4
Transaction effective date (tag 61)	Structure	Digits 0 to 9 in MMDD format
Transaction type code (tag 61)	Domain	`P` or `R`

Fields	Validation types	Validation
Transaction amount (tag 61)	Must exist	
Transaction amount (tag 61)	Length	1 to 15
Transaction amount (tag 61)	Structure	Digits 0 to 9 and , and .
Transaction funds type code (tag 61)	Must exist	
Transaction customer reference number (tag 61)	Length	1 to 16
Transaction customer reference number (tag 61)	Structure	SWIFT X character set
Transaction bank reference number (tag 61)	Length	1 to 16
Transaction bank reference number (tag 61)	Structure	SWIFT X character set
Transaction description (tag 61)	Must exist	
Transaction description (tag 61)	Length	1 to 34
Transaction description (tag 61)	Structure	SWIFT X character set
Information to account owner (tag 86)	Length	1 to 390
Information to account owner (tag 86)	Structure	SWIFT X character set
C/D mark (tag 62)	Must exist	
C/D mark (tag 62)	Length	1
C/D mark (tag 62)	Structure	C or D
Statement end date (tag 62)	Must exist	
Statement end date (tag 62)	Length	6
Statement end date (tag 62)	Structure	Digits in YYMMDD format
Closing balance currency code (tag 62) for transaction date	Must exist	
Closing balance currency code (tag 62) for transaction date	Length	3
Closing balance currency code (tag 62) for transaction date	Structure	Letters A to Z (uppercase)
Closing balance actual amount (tag 62) for transaction date	Must exist	
Closing balance actual amount (tag 62) for transaction date	Length	1 to 15
Closing balance actual amount (tag 62) for transaction date	Structure	Digits 0 to 9 and , and .
Closing balance currency code (tag 62) for value date	Length	3
Closing balance currency code (tag 62) for value date	Structure	Letters A to Z (uppercase)
Closing balance actual amount (tag 62) for value date	Length	1 to 15
Closing balance actual amount (tag 62) for value date	Structure	Digits 0 to 9 and , and .
Closing available balance C/D mark (tag 64)	Optional	

Fields	Validation types	Validation
Closing available balance C/D mark (tag 64)	Length	1
Closing available balance C/D mark (tag 64)	Structure	C or D
Closing available balance date (tag 64)	Optional	
Closing available balance date (tag 64)	Length	6
Closing available balance (tag 64)	Structure	Digits in YYMMDD format
Closing available balance currency code (tag 64)	Optional	
Closing available balance currency code (tag 64)	Length	3
Closing available balance currency code (tag 64)	Structure	Letters A to Z (uppercase)
Closing available balance amount (tag 64)	Optional	
Closing available balance amount (tag 64)	Length	1 to 15
Closing available balance amount (tag 64)	Structure	Digits 0 to 9 and , and .
Forward available balance C/D mark (tag 65)	Optional	
Forward available balance C/D mark (tag 65)	Length	1
Forward available balance C/D mark (tag 65)	Structure	C or D
Forward available balance date (tag 65)	Optional	
Forward available balance date (tag 65)	Length	6
Forward available balance (tag 65)	Structure	Digits in YYMMDD format
Forward available balance currency code (tag 65)	Optional	
Forward available balance currency code (tag 65)	Length	3
Forward available balance currency code (tag 65)	Structure	Letters A to Z (uppercase)
Forward available balance amount (tag 65)	Optional	
Forward available balance amount (tag 65)	Length	1 to 15
Forward available balance amount (tag 65)	Structure	Digits 0 to 9 and , and .
Information to account owner (tag 86)	Length	1 to 390
Information to account owner (tag 86)	Structure	SWIFT X character set

A.5.4 SWIFT MT940 BCS

The Society for Worldwide Interbank Financial Telecommunication (SWIFT) is owned by the financial industry to cooperatively supply secure, standardized messaging across more than 200 countries. SWIFT supports many different formats. A few of the common message formats are MT940, MT942, and MT950. They are used for current day, previous day, and summary reporting on bank accounts with the bank. Many of Wallstreet's CMM customers do business with banks that report bank account information in these formats.

Certain financial institutions report bank account information in a modified format (based on the SWIFT formats) called SWIFT BCS (Banking Communication Standard). The German banking market supports a German SWIFT BCS format that is widely used and adopted by financial institutions in the region. These formats have been defined by the BCS. The German SWIFT BCS format follows the SWIFT message format specifications, but with a slight modification and exclusion of certain components. This format is used and recommended by ERPs such as SAP.

CMM can import and export SWIFT BCS message files. CMM's bank transaction import and export functionality supports both code-based and XML-template-based formats.

The following table presents the major features of the SWIFT BCS message file format:

Features	Description
Acknowledgements (imports)	These formats support positive and negative acknowledgements; however, they do not support SWIFT delivery notifications.
Transaction types	These formats support bank statements for internal (activity on bank accounts with the in-house bank) and external (activity on external bank accounts) transactions.
Bank statement file names	File names follow the CMM standard. In other words, CMM creates the file name as a concatenation of the date-time stamp and the bank name to ensure it is unique.
Interface component scope	The file is plain text only. There is no digital signature, encryption, or other security features required. If you want to use these feature, you can do so by adding them to the interchange (see 3.4 Managing interchanges on page 56).

A.5.4.1 SWIFT MT940 BCS layout

SWIFT BCS messages are different from typical SWIFT messages in that their files begin with tag 20. Similar to the SWIFT format, the SWIFT BCS format's main message content ends the message with a hyphen (-) but does not continue to end the block with a closing brace (}). In addition, the SWIFT BCS format does not include block 5.

The following is an example SWIFT BCS message file:

```
:20:cititest
:25:12345678
:28C:109/10
:60F:C040920USD0,
°
-
```

A.5.4.2 SWIFT MT940 BCS tags

The following table presents the available tags that can be included in SWIFT BCS messages:

No.	Name	Required	Contents [1]	XML attribute
Tag 20				
N/A	Transaction reference number	Yes	16x	
Tag 25 [2]				
N/A	ABA code (sort code) and bank account number	Yes	35x	
Tag 28C [3]				
N/A	Statement number and sequence number	Yes	5n[/5n]	
Tag 60F (Opening balance)				
1	Credit/debit indicator	Yes	1a	
2	Statement start date	Yes	6n	
3	ISO currency code	Yes	3a	
4	Actual amount	Yes	15d	

No.	Name	Required	Contents [1]	XML attribute
Tag 61 (Statement line) [4]				
1	Value date	Yes	6!n	
2	Transaction date (if different than the value date)	Yes	4!n	
3	Credit/debit indicator	Yes	2a	
4	[Not used]	Yes	1!a	
5	Amount	Yes	15d	
6	SWIFT transaction type code (hard coded)	Yes	3!c	
7 [5]	Cash flow type	Yes	16x	
N/A	Separator	Yes	//	
8 [6]	Bath ID and transaction ID	Yes	16x	
N/A	Separator	Yes	/	
9 [7]	Currency and amount	Yes	34x	
Tag 86 (Information to account owner) [8]				
1 [9]	GVC	Yes	3n	
2	Counterparty ID	Yes	?00x (An 25)	
3	[Not used]	Yes	?10x (An 6)	
4	Description	Yes	?20x (An 27)	
5	Description	Yes	?21x (An 27)	
6	Description	Yes	?22x (An 27)	
7	Description	Yes	?23x (An 27)	
8	Description	Yes	?24x (An 27)	
9	Description	Yes	?25x (An 27)	
10	Description	Yes	?26x (An 27)	
11	Description	Yes	?27x (An 27)	
12	Description	Yes	?28x (An 27)	
13	Description	Yes	?29x (An 27)	
14	Description	Yes	?32x (An 27)	
15	Description	Yes	?33x (An 27)	
16	Description	Yes	?60x (An 27)	
17	Description	Yes	?61x (An 27)	
18	Description	Yes	?62x (An 27)	
19	Description	Yes	?63x (An 27)	
Tag 62 F (Closing balance)				
1	Credit/debit indicator	Yes	1!a	
2	Statement end date	Yes	6!n	

No.	Name	Required	Contents [1]	XML attribute
3	ISO currency code	Yes	3!a	
4	Actual amount	Yes	15d	

Table notes:

1. For more information on the codes used in this column, see A.13.1 SWIFT field lengths on page 303 and A.13.2 SWIFT field types on page 304.
2. Tag 25 contains the ABA code (sort code), followed by a forward slash (/), followed by the bank account number.
3. Tag 28C contains the statement number, followed by a forward slash (/), followed by the sequence number.
4. Loop tag 61 for each transaction in the statement.
5. For more information on the contents of section 7 of tag 61, see A.13.4 SWIFT transaction type identification code mappings on page 305.
6. Section 8 of tag 61 contains the batch ID, followed by a semicolon (;), followed by the transaction ID.
7. Section 9 of tag 61 contains OCMT/, followed by the ISO code of the currency, followed by the amount in 13,2 format.
8. Loop tag 86 for each transaction in the statement.
9. For more information on the contents of section 1 of tag 86, see A.13.5 SWIFT business transaction code (GVC) mappings on page 306.

A.5.4.3 SWIFT MT940 BCS validation

Validation is performed during the import and parsing of the file for import files and during the export and file creation process for export files.

For the most part, the validation rules below are based on the validation rules set out by the SWIFT specifications:

Fields	Validation types	Validation
Transaction reference number (tag 20)	Must exist	
Transaction reference number (tag 20)	Length	1 to 16
Transaction reference number (tag 20)	Structure	SWIFT X character set
Account identification (tag 25)	Must exist	
Account identification (tag 25)	Length	1 to 35
Account identification (tag 25)	Structure	SWIFT X character set
Statement number (tag 28C)	Must exist	
Statement number (tag 28C)	Length	1 to 5
Statement number (tag 28C)	Structure	Digits 0 to 9
Sequence number (tag 28C)	Optional	
Sequence number (tag 28C)	Length	1 to 5
Sequence number (tag 28C)	Structure	Digits 0 to 9
C/D mark (tag 60F)	Must exist	
C/D mark (tag 60F)	Length	1

Fields	Validation types	Validation
C/D mark (tag 60F)	Structure	C or D
Statement start date (tag 60F)	Must exist	
Statement start date (tag 60F)	Length	6
Statement start date (tag 60F)	Structure	Digits in YYMMDD format
Opening balance currency code (tag 60)	Must exist	
Opening balance currency code (tag 60)	Length	3
Opening balance currency code (tag 60)	Structure	Letters A to Z (uppercase)
Opening balance actual amount (tag 60) for transaction date	Must exist	
Opening balance actual amount (tag 60) for transaction date	Length	1 to 15
Opening balance actual amount (tag 60) for transaction date	Structure	Digits 0 to 9 and , and .
Transaction value date (tag 61)	Must exist	
Transaction value date (tag 61)	Length	6
Transaction value date (tag 61)	Structure	Digits 0 to 9 in YYMMDD format
Transaction effective date (tag 61)	Length	4
Transaction effective date (tag 61)	Structure	Digits 0 to 9 in MMDD format
Transaction type code (tag 61)	Domain	P or R
Transaction amount (tag 61)	Must exist	
Transaction amount (tag 61)	Length	1 to 15
Transaction amount (tag 61)	Structure	Digits 0 to 9 and , and .
Transaction funds type code (tag 61)	Must exist	
Transaction customer reference number (tag 61)	Length	1 to 16
Transaction customer reference number (tag 61)	Structure	SWIFT X character set
Transaction bank reference number (tag 61)	Length	1 to 16
Transaction bank reference number (tag 61)	Structure	SWIFT X character set
Transaction description (tag 61)	Must exist	
Transaction description (tag 61)	Length	1 to 34
Transaction description (tag 61)	Structure	SWIFT X character set
Information to account owner (tag 86)	Length	1 to 390
Information to account owner (tag 86)	Structure	SWIFT X character set
C/D mark (tag 62)	Must exist	
C/D mark (tag 62)	Length	1
C/D mark (tag 62)	Structure	C or D
Statement end date (tag 62)	Must exist	
Statement end date (tag 62)	Length	6

Fields	Validation types	Validation
Statement end date (tag 62)	Structure	Digits in YYMMDD format
Closing balance currency code (tag 62) for transaction date	Must exist	
Closing balance currency code (tag 62) for transaction date	Length	3
Closing balance currency code (tag 62) for transaction date	Structure	Letters A to Z (uppercase)
Closing balance actual amount (tag 62) for transaction date	Must exist	
Closing balance actual amount (tag 62) for transaction date	Length	1 to 15
Closing balance actual amount (tag 62) for transaction date	Structure	Digits 0 to 9 and , and .
Closing balance currency code (tag 62) for value date	Length	3
Closing balance currency code (tag 62) for value date	Structure	Letters A to Z (uppercase)
Closing balance actual amount (tag 62) for value date	Length	1 to 15
Closing balance actual amount (tag 62) for value date	Structure	Digits 0 to 9 and , and .
Information to account owner (tag 86)	Length	1 to 390
Information to account owner (tag 86)	Structure	SWIFT X character set

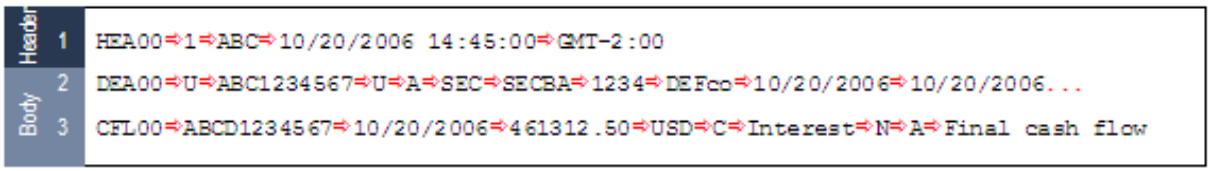
A.6 Deal import formats

CMM supports the following standard formats for deal imports:

Type	Wallstreet formats	EDIFACT formats	SWIFT formats	Other formats
Deal	• Flat file deal	N/A	N/A	N/A

A.6.1 Wallstreet flat file deal

The following is an example deal flat file (with tabs as the delimiters):



This file contains three lines grouped into two sections:

Line	Name	Description
Header		
1	File information	General information on the file.
Body		
2	Deal details	The individual deal. Note: Include one line 2 for each deal in the file.
3	Cash flow details	The individual cash flow. Note: Include one line 3 for each cash flow in a deal.

Each of these lines contains required and non-required components. While you do not have to provide information for non-required components, you must include delimiters (tabs) for them to indicate their place in the file.

Deal flat files must be tab delimited with file names in the following format: `DEAL[Date].tab`, where `[Date]` is the date of the deal in MMDDYY format.

Note: If the source system cannot generate tab-delimited files, contact Wallstreet for information on alternatives.

The deal flat file format supports the following deal types:

- Securities
- Deposits
- Borrowings
- Long-term securities
- Spot foreign exchanges
- Forward foreign exchanges
- Non-deliverable foreign exchanges.

As of this release, the deal flat file format does not support the following deal types:

- Swaps
- Options
- Swap deposits.

Note: The deal flat file format is intended for customers using the treasury management functionality of CMM instead of TRM.

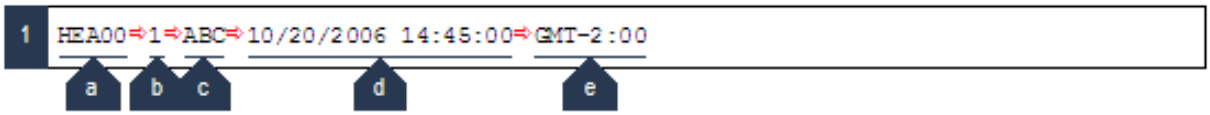
A.6.1.1 Wallstreet flat file deal header

The header section of a deal flat file consists of one line:

- File information (line 1).

A.6.1.1.1 File information (line 1)

Line 1 contains general information on the file:



The following table defines the components of this line:

No.	Name	Req'd	Max. size	Data type	Description	Example
1a	File information indicator	Yes	N/A	Constant	An indicator that the line contains general file information. The value of this component must be HEA00.	HEA00
1b	Batch ID	Yes	8	Integer	A unique identifier for the batch. The originating system must be able to generate a unique number per batch. Note: CMM checks the value of this component before loading the file to prevent accidentally reimporting the same batch twice.	1
1c	Originating system ID	Yes	20	String	A unique code for the originating system. If there are several instances of an originating system in an organization, the different instances must be labeled uniquely. For example, the name of the system could be concatenated with a sequence number.	ABC
1d	Creating date/time	Yes	20	Date/time	The date and time when the batch was created. For more information on dates in the deal flat file format, see A.6.1.3 Dates in Wallstreet flat file deal on page 240.	10/20/2006 14:45:00
1e	Time zone	No	8	String	The time zone of the originating system, expressed in reference to Greenwich Mean Time (GMT). Wallstreet recommend you include this component if you are working in an environment where the originating systems are in different time zones.	GMT-2:00

A.6.1.2 Wallstreet flat file deal body

The body section of a deal flat file consists of two lines:

- Deal details (line 2)
- Cash flow details (line 3).

A.6.1.2.1 Deal details (line 2)

Line 2 contains the individual deals. (Include one line 2 for each deal in the file.)

The components in line 2 must display in the following order:

Name	Req'd	Max. size	Data type	Description	Example
Deal detail indicator (Component 1)	Yes	N/A	Constant	An indicator that the line contains general file information. The value of this component must be `DEA00`.	`DEA00`
Action code (Component 2)	Yes	N/A	String	A code indicating the action CMM should take. Acceptable values: • `U` to insert a new deal or replace an existing deal • `D` to delete a deal. Note: Only use `D` for new and forecasted deals.	`U`
Deal ID (Component 3)	Yes	20	String	The deal's unique ID. Note: This component links the deal to its associated cash flow.	`ABC1234567`
Deal processing status (Component 4)	Yes	N/A	String	A code to indicate the actions that have already been taken on the deal. Acceptable values: • `U` for an unmodeled deal • `M` for a modeled deal. Confirmation and settlement of an unmodeled deal occur in CMM. An unmodeled deal is equivalent to a deal that is manually entered in CMM. An unmodeled deal does not need cash flow details, as CMM calculates them automatically. For CMM to import an unmodeled deal, it must be able to support (or "model") the deal's type. For more information, contact CMM. A modeled deal is a deal that has been precalculated. CMM expects the cash records resulting from the deal to be attached in subsequent cash flow details. CMM only supports modeled deals in the LTS instrument category.	`U`
Deal status (Component 5)	Yes	N/A	String	The deal's status. Acceptable values: • `F` for a forecasted deal • `A` for an actual deal • `C` for a confirmed deal • `S` for a settled deal • `M` for a matured deal • `D` for a deleted deal. Note: This component is only applicable for new deals. For existing deals, CMM uses the deals' statuses as recorded in the database.	`A`

Name	Req'd	Max. size	Data type	Description	Example
Instrument category (Component 6)	Yes	10	String	The ID of the deal's instrument category. Acceptable values for unmodeled deals: • DEP for deposit • BRW for borrowing • SEC for security • LOC for line of credit • FEXSPOT for FX spot • FEXFWD for FX forward • FEXND for non-deliverable FX • CEQT for common equity • LTS for long-term security. Acceptable value for modeled deals: • LTS for long-term security.	SEC
Instrument type (Component 7)	Yes	10	String	The ID of the deal's instrument type. This must be valid instrument type for the selected instrument category.	SECBA
Entity ID (Component 8)	Yes	25	String	The ID of the deal's entity.	1234
Counterparty ID (Component 9)	Yes	25	String	The ID of the deal's counterparty.	DEFco
Book date (Component 10)	Yes	N/A	Date	The deal's booking date. For more information on dates in the deal flat file format, see A.6.1.3 Dates in Wallstreet flat file deal on page 240.	10/20/2006
Value date (Component 11)	Yes	N/A	Date	The deal's value date. For more information on dates in the deal flat file format, see A.6.1.3 Dates in Wallstreet flat file deal on page 240.	10/20/2006
Maturity date (Component 12)	Yes	N/A	Date	The deal's maturity date. For more information on dates in the deal flat file format, see A.6.1.3 Dates in Wallstreet flat file deal on page 240.	10/20/2006
Previous interest date (Component 13)	Cond. [1]	N/A	Date	The deal's previous interest date. This date must be before the deal's value date. For more information on dates in the deal flat file format, see A.6.1.3 Dates in Wallstreet flat file deal on page 240.	09/20/2006
Next interest date (Component 14)	Cond. [1]	N/A	Date	The deal's next interest date. This date must on or after the deal's value date. For more information on dates in the deal flat file format, see A.6.1.3 Dates in Wallstreet flat file deal on page 240.	11/20/2006

Name	Req'd	Max. size	Data type	Description	Example
Amount (Component 15)	Yes	21	Double	The deal's amount (in the currency defined in component 16). Note: Do not include thousand separators in amounts.	`11000000.00`
Currency code (Component 16)	Yes	N/A	String	The **ISO** code of the deal's currency.	`USD`
Contra currency code (Component 17)	No	6	String	The ISO code of the deal's contra-currency (for foreign exchange deals).	`CAD`
Quoted against currency code (Component 18)	No	20	Double	The ISO code of the currency to use as the base for the foreign exchange rate (for foreign exchange deals). If you do not provide a value for this component, CMM uses the value of component 16.	`USD`
Commission amount (Component 19)	No	21	Double	The deal's commission amount.	`1000`
Fee amount (Component 20)	No	21	Double	The deal's fee amount.	`1000`
Trader ID (Component 21)	Yes	40	String	The user ID of the deal's trader.	`JSMITH`
Portfolio ID (Component 22)	Yes	10	String	The ID of the deal's portfolio.	`123456`
Buy/sell code (Component 23)	Yes	2	String	A code indicating whether the deal was bought or sold Acceptable values: • `B` for buy (in other words, purchase or investment) • `S` for sell (in other words, borrow or issue debt).	`B`
Original deal ID (Component 24)	No	20	String	The ID of the deal's original deal.	`ABC1234566`
Issuer ID (Component 25)	No	20	String	The ID of the deal's issuing entity.	`1233`
Issue ID (Component 26)	No	20	String	The ID of the deal's issue.	`123`
Issue ID type ID (Component 27)	No	4	Integer	The ID of the deal's issue type. Acceptable values: • `1` for COSIP • `2` for ISIN • `3` for SODOL • `4` for other.	`1`

Name	Req'd	Max. size	Data type	Description	Example
Aggregate position (Component 28)	No	2	String	The deal's aggregate position. Acceptable values: • C • D • I.	C
Interest discount code (Component 29)	Yes	20	String	The deal's interest discount code. Acceptable values for short-term instrument types (BRW, DEP, and SEC): • EXCTSMPINT for exact simple interest • SMPINT for ordinary simple interest • DSCNTQYLD for discount—quoted yield • DSCNTQPRICE for discount—quoted price • SMPDSCNT for discount. Acceptable values for long-term instrument types (LTS): • AMORTIZATION for amortization • FLOATINGRATE for floating rate • FIXEDRATE for fixed rate. Acceptable values for short- and long-term instrument types: • MODELED for modeled deals.	SMPINT
Interest type (Component 30)	No	20	Double	The deal's interest type.	
Day basis (Component 31)	Yes	3	Integer	The deal's day basis. Acceptable values: • 1 for 360 • 2 for 365 • 3 for 366 • 4 for 30/360 • 5 for 30E/360 • 6 for Actual/360 • 7 for Actual/365 • 8 for Actual/Actual • 9 for Actual W/252. Note: 1 through 4 are for backwards compatibility only.	8
Price (Component 32)	Cond. [2]	20	Double	The deal's quoted price. If you do not provide a value for this component, CMM uses 1.	1
Price divisor (Component 33)	No	20	Double	The divisor CMM uses for securities pricing if other than one.	.9

Name	Req'd	Max. size	Data type	Description	Example
Tax method code (Component 34)	Cond. [2]	4	Integer	The code of the deal's tax method.	0
Tax amount (Component 35)	Cond. [2]	20	Double	The deal's tax amount.	0
Tax amount withheld (Component 36)	Cond. [2]	20	Double	The deal's tax amount withheld.	0
Commission method (Component 37)	Cond. [2]	4	Integer	The deal's commission method.	0
Fee method (Component 38)	Cond. [2]	4	Integer	The deal's fee method.	0
Clearing system (Component 39)	No	25	String	The deal's clearing system.	ClearSys
Clearing agent (Component 40)	No	25	String	The deal's clearing agent.	ClearAgent
Safekeeper (Component 41)	No	25	String	The deal's safekeeper.	SafeKeeper
Withholding Agent (Component 42)	No	25	String	The deal's withholding agent.	WithHold Agent
Other charges (Component 43)	No	20	Double	The deal's other charges.	500
Rate (Component 44)	Cond. [3]	20	Double	The deal's quoted rate (for SEC, DEP, and LTS deals), contract rate (for FEXSPOT deals), or forward point (for FEXFWD deals).	0.0025
Spot rate (Component 45)	No	20	Double	The deal's spot rate (for foreign exchange deals).	0.0025
Entity paying bank account number (Component 46)	Cond. [3]	30	String	The number of the deal's paying entity bank account.	123-456-789
Entity paying bank account currency code (Component 47)	Cond. [3]	N/A	String	The ISO currency code of the deal's paying entity bank account.	USD
Entity paying account bank branch ID (Component 48)	Cond. [3]	15	String	The sort or ABA code of the deal's paying entity bank account.	
Entity receiving bank account number (Component 49)	Cond. [3]	30	String	The number of the deal's receiving entity bank account.	123-456-789

Name	Req'd	Max. size	Data type	Description	Example
Entity receiving bank account currency code (Component 50)	Cond. [3]	N/A	String	The ISO currency code of the deal's paying entity bank account.	USD
Entity receiving account bank branch ID (Component 51)	Cond. [3]	15	String	The sort or ABA code of the deal's paying entity bank account.	
Counterparty bank code (Component 52)	Cond. [3]	20	String	The SWIFT code of the deal's counterparty bank.	
Counterparty bank code type ID (Component 53)	Cond. [3]	20	String	The bank code type of the deal's counterparty bank.	
Counterparty bank account number (Component 54)	Cond. [3]	30	String	The number of the deal's counterparty bank account.	987-654-321
Counterparty receiving bank account currency code (Component 55)	Cond. [3]	N/A	String	The ISO currency code of the deal's counterparty bank account.	CAD
Counterparty receiving account bank branch ID (Component 56)	Cond. [3]	15	String	The sort or ABA code of the deal's counterparty bank account.	
Holiday country code (Component 57)	Cond. [3]	20	String	The code of the deal's bank holiday country. CMM uses this component to calculate payment and receipt dates.	US
DAN adv. type (Component 58)	No	N/A	N/A	This component is a placeholder for future development.	N/A
Payment frequency (Component 59)	Cond. [3]	N/A	Integer	The deal's principal frequency. Acceptable values: • 24 for maturity • 12 for monthly • 6 for bimonthly • 4 for quarterly • 3 for third year • 2 for semi-annually • 1 for annually.	12

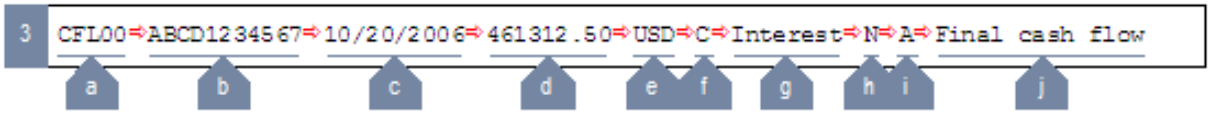
Name	Req'd	Max. size	Data type	Description	Example
Interest calculation frequency (Component 60)	No	N/A	Integer	The deal's interest calculation frequency. Acceptable values: • 12 for monthly • 6 for bimonthly • 4 for quarterly • 3 for third year • 2 for semi-annually • 1 for annually.	12
Put/call (Component 61)	No	N/A	N/A	This component is a placeholder for future development.	N/A
Compounding type (Component 62)	No	N/A	N/A	This component is a placeholder for future development.	N/A
Description (Component 63)	No	60	String	A description of the deal.	See Corp. Treasury for details
Interest calculation frequency (Component 64)	Cond. [4]	N/A	Integer	The deal's interest calculation frequency. Acceptable values: • 12 for monthly • 6 for bimonthly • 4 for quarterly • 3 for third year • 2 for semi-annually • 1 for annually.	12

Table notes:

1. This component is required for unmodeled long-term securities.
2. This component is required for unmodeled deals.
3. This component is required for confirmed and settled deals.
4. This component is required for unmodeled deals with component 29 set to Amortization or Floating Rate.

A.6.1.2.2 Cash flow details (line 3)

Line 3 contains the individual cash flow details:



(Include one line 3 for each cash flow detail in a deal.)

The following table defines the components of this line:

No.	Name	Req'd	Max. size	Data type	Description	Example
3a	Cash flow detail indicator	Yes	N/A	Constant	An indicator that the line contains cash flow details. The value of this component must be `CFL00`.	`CFL00`
3b	Deal ID	Yes	20	String	The ID of the cash flow detail's parent deal.	`ABCD1234567`
3c	Cash flow date	Yes	N/A	Date	The cash flow detail's value date. For more information on dates in the deal flat file format, see A.6.1.3 Dates in Wallstreet flat file deal on page 240.	`10/20/2006`
3d	Amount	Yes	N/A	Double	The cash flow detail's amount (in the currency defined in currency code). Note: Do not include thousand separators in amounts.	`461312.50`
3e	Currency code	Yes	N/A	String	The ISO code of the cash flow detail's currency.	`USD`
3f	Cash flow direction	Yes	N/A	String	The cash flow detail's direction. Acceptable values: • `C` for credits (inflows) • `D` for debits (outflows).	`C`
3g	Cash flow type	Yes	20	String	The cash flow detail's type. Acceptable values: • `Interest` for payment or receipt of interest • `Principal` for payment or receipt of principal • `Commissions` for payment or receipt of commission • `Fees` for payment or receipt of fees • `Other` for any other items.	`Interest`

No.	Name	Req'd	Max. size	Data type	Description	Example
3h	Cash flow status	Yes	N/A	String	The cash flow detail's status. Acceptable values: • `N` for new • `F` for forecast • `C` for confirmed • `S` for settled.	`N`
3i	Action code	Yes	N/A	String	The action to take. Acceptable values: • `A` to add the new cash flow detail as a new event • `U` to replace an event already in the database • `D` to delete an event from the database.	`A`
3j	Description	No	256	String	A description of the cash flow detail. Note: You must provide a tab even if you do not provide a description (assuming the file it tab delimited).	`Final cash flow`

A.6.1.3 Dates in Wallstreet flat file deal

Wallstreet recommends using dates with four digit years in the deal flat file format (for example, `2-Sep-2006`).

CMM can accept dates with two-digit years after 2000 (for example, `2-Sep-06`). However, this practice is not recommended due to possible misinterpretation.

Leading zeros on days and months (if numeric months are used) are not required.

Do not use the DD/MM/YY format as this can be misinterpreted as MM/DD/YY in ambiguous cases. For example, `01/02/03` could be interpreted as January 2, 2003; February 1, 2003; or February 3, 2001.

A.7 Transaction export formats

CMM supports the following standard formats for transaction exports:

Type	Wallstreet formats	EDIFACT formats	SWIFT formats	Other formats
Payment	N/A	• PAYEXT [1] • PAYMUL [1]	• MT100 [1] • MT101 • MT103 [1] • MT202 [1] • ISO 20022 XML credit transfer	• ANSI X12 820 [1] • BACS [1] • CFONB/AFB VIR106 [1]
Direct debit	N/A	• DIRDEB [1]	• SWIFT MT104 • ISO 20022 XML directdebit transfer	• CFONB/AFB PRE160 [1]
Letter of credit	N/A	N/A	N/A	• CFONB/AFB LCR240 [1]
Pre-advice	N/A	N/A	• SWIFT MT210	N/A

Table notes:

1. For information on formats not documented in this appendix, contact Wallstreet.

A.7.1 SWIFT MT101

The SWIFT MT101 message is sent by a financial institution on behalf of an account owner/ordering party that is not a financial institution. The SWIFT MT101 formats are the most commonly used SWIFT payment formats and support multiple types of commercial payments. They carry minimal information—just enough to allow the payment instruction to be correctly executed by the paying bank and deposited into the correct account by the beneficiary's bank. The SWIFT MT101 formats can be used to order the movement of funds:

- Between ordering customer accounts
- In favor of a third party, either locally or internationally.

CMM supports two SWIFT MT101 formats:

- Batch: Multiple payments are sent in one file using the batch debit/multiple credit approach.
- Transaction: Individual payments are sent in separate files.

The following table presents the major features of the SWIFT MT101 formats:

Features	Description
Payment types	The formats support domestic non-urgent, domestic urgent, cross-border non-urgent, cross-border urgent, cross-border intermediary non-urgent, and cross-border intermediary urgent payments.
Payment capture	Payments can be captured manually, imported from an accounts payable file, or created at the time a deal is settled.
Payment file names	File names follow the CMM standard. In other words, CMM creates the file name as a concatenation of the date-time stamp and the receiving bank ID to ensure it is unique.
Interface component scope	The file is plain text only. There is no digital signature, encryption, or other security features required. If you want to use these feature, you can do so by adding them to the interchange (see 3.4 Managing interchanges on page 56).

Features	Description
Interface content scope	The interface only handles payments/debits. Receipts/credits are not catered for.
Data assumption	Users capture valid SWIFT codes into the import file or static data. CMM does not verify the veracity of the SWIFT code, merely the length.
Payment group rules	CMM does not apply any payment grouping rule.
Bank payment message format	The SWIFT MT101 formats produced from the XML tool does not introduce or support any new optional fields or segments that were not supported in the Java code.

A.7.1.1 SWIFT MT101 payment scenarios

This page documents the payment scenarios CMM supports for SWIFT MT101.

A.7.1.1.1 Scenario 1A

In this scenario, a head office sends a payment on behalf of itself and multiple subsidiaries using a direct method.

To implement this scenario in CMM, do the following:

1. Create an interchange for the receiver bank.
2. In the interchange, set the head office entity as the sender.

   The head office entity's SWIFT code is posted in the Ordering Customer position in the SWIFT MT101 message file.

A.7.1.1.2 Scenario 1B

In this scenario, a head office sends a payment on behalf of itself and multiple subsidiaries using a relay method.

To implement this scenario in CMM, do the following:

1. Create an aggregate bank with a SWIFT code equal to the receiving bank BIC.
2. Create the head office serving bank as the aggregate bank's child.
3. Create an interchange for the aggregate bank.
4. In the interchange, set the head office entity as the sender.

   The head office entity's SWIFT code is posted in the Ordering Customer position in the SWIFT MT101 message file.

A.7.1.1.3 Scenario 2A

In this scenario, a head office sends a payment from a subsidiary bank account using a direct method.

To implement this scenario in CMM, do the following:

1. Create an interchange for the receiver bank.
2. In the interchange, set the head office entity as the sender.

   The head office entity's SWIFT code is posted in the Instructing Party position in the SWIFT MT101 message file.

   During the release process, CMM determines if the bank account belongs to the head office entity, which distinguishes this scenario from 1A. This is required because some of the field mappings differ between the two scenarios.

A.7.1.1.4 Scenario 2B

In this scenario, a head office sends a payment from a subsidiary bank account using a relay method.

To implement this scenario in CMM, do the following:

1. Create an aggregate bank with a SWIFT code equal to the receiving bank BIC.
2. Create the subsidiary serving bank as the aggregate bank's child.
3. Create an interchange for the aggregate bank.
4. In the interchange, set the head office entity as the sender.

   The head office entity's SWIFT code is posted in the Instructing Party position in the SWIFT MT101 message file.

   During the release process, CMM determines if the bank account belongs to the head office entity, which distinguishes this scenario from 1B. This is required because some of the field mappings differ between the two scenarios.

A.7.1.1.5 Scenario 3A

In this scenario, a shared service center or payment factory sends a payment using a direct method.

To implement this scenario in CMM, do the following:

1. Set the Enable Shared Service Center configuration parameter to `True`.
2. Create an interchange for the receiver bank.
3. In the interchange, enter the shared service center's BIC in the **Sender LT Address or Shared Service Center BIC** field.
4. In the interchange, set the head office entity as the sender.

   The head office entity's SWIFT code is posted in the Ordering Customer position in the SWIFT MT101 message file.

   The difference between this scenario and 1A is that the entity SWIFT code is not the same as the interchange sender ID.

   The difference between this scenario and 2A is that the transaction in this scenario is an on-behalf-of payment.

A.7.1.1.6 Scenario 3B

In this scenario, a shared service center or payment factory sends a payment using a relay method.

To implement this scenario in CMM, do the following:

1. Create an aggregate bank with a SWIFT code equal to the receiving bank BIC.
2. Create the subsidiary serving bank as the aggregate bank's child.
3. Set the Enable Shared Service Center configuration parameter to `True`.
4. Create an interchange for the aggregate bank.
5. In the interchange, enter the shared service center's BIC in the **Sender LT Address or Shared Service Center BIC** field.
6. In the interchange, set the head office entity as the sender.

   The head office entity's SWIFT code is posted in the Ordering Customer position in the SWIFT MT101 message file.

A.7.1.2 SWIFT MT101 layout

The SWIFT MT101 message file layout is identical to that of the SWIFT MT940 message file layout (see A.5.3.1 SWIFT MT940 layout on page 217).

A.7.1.3 SWIFT MT101 header blocks

SWIFT MT101 message files begin with three header blocks:

1. Fixed basic header block
2. Fixed application header block
3. User header block.

Header block 3 is usually omitted. CMM does not currently support this header block and ignore it if it is included in import files.

A.7.1.3.1 Fixed basic header block

The following table presents the contents of the fixed basic header block:

Content	Description	Required	Acceptable values	XML attribute
{1:	Fixed basic header identifier	Yes	`{1:`	Not available
a	Application identifier	Yes	F (All user-to-user, FIN system, and FIN service messages)	Not available
01	Service identifier	Yes	`01`	Not available
aaaaaaaaxxxx	ID of the ordering party bank	Yes	Alphabetic (A to Z) Note: If the ID of the ordering party bank is not 12 characters long, add Xs until it is 12 characters long.	`interchange_ interchange_ sender_id`
dddd	Number of data sets	Yes	0000	Not available
dddddd}	Transaction number in the file	Yes	000000	Not available

A.7.1.3.2 Fixed application header block

The following table presents the contents of the fixed application header block:

Content	Description	Required	Acceptable values	XML attribute
{2:	Fixed application header identifier	Yes	`{2:`	Not available
I	Input/output identifier	Yes	`I`	Not available
ddd	Message type	Yes	`101`	Not available

Content	Description	Required	Acceptable values	XML attribute
aaaaaaaaxxxx	Destination address	Yes	Alphabetic (A to Z) Note: If the destination address is not 12 characters long, add Xs until it is 12 characters long.	`interchange_interchange_recipient_id`
a	Message priority	Yes	S for user-to-system messages U for urgent messages. N for all user-to-user messages	Not available
d [1]	Message monitoring	No	N/A	N/A
ddd1	Obsolescence period	No	N/A	N/A

Table notes:

1. This content is currently out of scope.

A.7.1.4 SWIFT MT101 main message block

Block 4 contains the main message of a SWIFT message file. It usually begins with tag 20.

For the batch format, CMM groups the payments by party ID and payment currency (one single debit entry). This section considers two possible scenarios:

- The paying bank account is identical (hereinafter referred to as "pay-account-identical")
- The paying bank account is not identical (hereinafter referred to as "not pay-account-identical").

A.7.1.4.1 Supported tags

The following table presents the available tags that are supported:

Tag	Name	Required	Contents [1]	CMM mapping
Sequence A: General information				
20	Sender's reference	M	16x	• In the batch format: Batch ID • In the transaction format: Customer reference ID

Tag	Name	Required	Contents [1]	CMM mapping
21 R	Customer specified reference	O	16x	• In the batch format: Batch ID • In the transaction format: N/A
28 D	Message index/total	M	5n/5n	1/1 (static value)
50 a	Instructing party	O	C or L	• When the sender is the bank account owner: N/A (case 1A, 1B) • When the sender is not the bank account owner and is not the shared service center: Interchange sender ID (case 2A, 2B) • When the sender is the shared service center: N/A (case 3A, 3B)
50 a	Ordering customer	O	G or H	• When the sender is the shared service center: (case 3A, 3B) • For pay-account-identical: - When the sender is the bank account owner: Use 50G and map to the bank account number and entity BEI - When the sender is not the account owner and is not the shared service center: Use 50H and map to the bank account number, payor long name, and payor full address
52 a	Account servicing institution	O	A or C	When pay-account-identical and aggregated bank interchange: Use 52A and map to the bank BIC
51 a	Sending institution	O		N/A
30	Request execution date	M	6!n	Value date
25	Authorization	O	35x	N/A
Sequence B: Transaction details (repeat once for each transaction in the group)				
21	Transaction reference	M	16x	Customer reference ID
21 F	F/X deal reference	O	16x	N/A
23 E	Instruction code	O	4!c[/30x]	Bank instructions
32 B	Currency/transaction amount	M	3!a15d	Payment currency Payment amount
50 a	Instructing party	O	C or L (35x)	• When the sender is the bank account owner: Originator long name (case 1A, 1B) • When the sender is not the bank account owner and is not the shared service center: N/A (case 2A, 2B) • When the sender is the shared service center: Originator long name (case 3A, 3B) (If the instructing party is posted in sequence A, nothing is not posted here.)

Tag	Name	Required	Contents [1]	CMM mapping
50 a	Ordering customer	O	G (34x/4!a2!a2!c [3!c]) or H (34x/4*35)	• When the sender is the shared service center: use 50H and map to bank account number, payor long name, and payor full address • For not pay-account-identical: - When the sender is the bank account owner: Use 50G and map to bank account number and entity BEI - When the sender is not the bank account owner and is not the shared service center: Use 50H and map to bank account number, payor long name, and payor full address (If the instructing party is posted in sequence A, nothing is not posted here because the ordering customer is the same as the instructing party.)
52 a	Account servicing institution	O	A ([/1!a][/34x]4! a2!a2!c[3!c]) or C (/34x)	When not pay-account-identical and aggr-bank-interchange: Use 52A and map to the bank BIC (If the instructing party is posted in sequence A, nothing is not posted here because the account servicing institution is the same as the instructing party.)
56 a	Intermediary	O	A, C, or D	• If the intermediary bank SWIFT code exists, use 56A and map to the code • If the intermediary bank name and address exist, use 56D and map to the name and address
57 a	Account with institution	O	A, C, or D	• If UK domestic payment: Post :57D:/SC/payee bank branch ID • If not UK domestic payment and the payee bank BIC exists: Post :57A:payee bank BIC • Otherwise: Post :57D:payee bank long name and address
59 a	Beneficiary	M	A or no option letter	• If the payee has a BIC or BEI: Post :59A:+"/payee bank account number"+BIC or BEI • If the payee does not have a BIC or BEI: Post :59:+"/payee bank account number"+payee name and address

Tag	Name	Required	Contents [1]	CMM mapping
70	Remittance information	O	4*35x	Beneficiary message • INV: If remittance details are available, build INV based on the details (for example, /INV/AUG06-345//AUG06-346). • ROC: If remittance details are not available, build ROC with the customer reference ID (for example, /ROC/RA-PL-9876-103). • IPI: If the counterparty message is not empty, build IPI with the counterparty message trimmed to 20 characters. (Users can put a unique reference identifying a related international payment in the counterparty message.) • RFB: If the beneficiary customer BIC or BEI is available, build RFB with the BIC or BEI. Otherwise, build RFB with the beneficiary long name truncated to 16 characters. • ULTB: N/A. (It is assumed the beneficiary is always the ultimate beneficiary.) • INST: If the instructing party is different from the ordering customer, build INST with the instructing party. The order of these items in the tag is as follows: **1.** INV or DOC **2.** IPI **3.** RFB **4.** INST
77 B	Regulatory reporting	O	3*35x	Regulatory code (This tag is only posted for cross-border payments.)
33 B	Currency/original ordered amount	O	3!a15d	N/A
71 A	Details of charges	M	3!a	Financial charge allocation: • BEN: All transaction charges, including the charges of the financial institution serving the order customer's bank account, for the subsequent credit transfers are to be borne by the beneficiary customer. • OUR: All transaction charges for the subsequent credit transfer are to be borne by the ordering customer. • SHA: All transaction charges other than the charges of the financial institution servicing the ordering customer bank account are borne by the beneficiary customer.
25 A	Charges account	O	/34x	N/A
36	Exchange rate	O	12d	N/A

Table notes:

1. For more information on the codes used in this column, see A.13.1 SWIFT field lengths on page 303 and A.13.2 SWIFT field types on page 304.

A.7.1.4.2 Unsupported tags

The following table presents the available tags that are unsupported:

Tag	Name	Required	Contents [1]	CMM mapping
Mandatory sequence A general information				
21 R	Customer specified reference	O	16x	All debit items are posted individually.
50 a	Instructing party	O	C or L	Use this only when the instructing customer is not also the account owner. In CMM, the payor is always the owner of the account.
51 a	Sending institution	O		N/A as this field is only valid for IFT.
25	Authorization	O		Additional security provisions: digital signature which is not supported by CMM.
Mandatory repetitive sequence B transaction details				
50 a	Instructing party	O	C or L	Use this only when the instructing customer is not also the account owner. In CMM, the payor is always the owner of the account.
50 a	Ordering customer	O	G or H	This field is populated in the sequence A. Based on condition C3, this field will not be displayed in here.
52 a	Account servicing institution	O	A or C	This field is populated in the sequence A. Based on condition C6, this field will not be displayed in here.
70	Remittance information	O		The existing codes use CptyMessage. However, this is incorrect and should be modified.
77 B	Regulatory reporting	O		When the residence of either the ordering customer or beneficiary customer is to be identified. N/A in CMM.
33 B	Currency/original ordered amount	O		This field is only used when the currency and amount are different from those specified in field 32B. N/A in CMM.
25 A	Charges account	O		When used, the account number must be different from the account number specified in tag 50a. N/A in CMM.
36	Exchange rate	O		N/A in CMM.

Table notes:

1. For more information on the codes used in this column, see A.13.1 SWIFT field lengths on page 303 and A.13.2 SWIFT field types on page 304.

A.7.2 SWIFT MT101 validation

Validation is performed during the export and file creation process.

For the most part, the validation rules below are based on the validation rules set out by the SWIFT specifications:

Field	Validation type	Validation
CUSTOMER REFERENCE ID	Must exist	
	Length	1 to 16
VALUE DATE	Must exist	
PAYMENT CURRENCY	Must exist	
PAYMENT AMOUNT	Must exist	
PAYOR	Must exist	
PAYOR BANK	Must exist	
PAYOR BANK ACCT	Must exist	
PAYOR LONG NAME	Must exist	
PAYOR BANK ACCT NUMBER	Must exist	
	Length	1 to 34
PAYOR BANK SWIFT CODE	Must exist	
	Length	Either 8 to 11
PAYEE LONG NAME	Must exist	
PAYEE BANK ACCT NUMBER	Must exist	
	Length	1 to 34
PAYEE BANK SWIFT CODE or PAYEE BANK LONG NAME	At least one of them must exist	
PAYEE BANK SWIFT CODE	If it exists, its length must be	Either 8 or 11
INTERMEDIARY BANK SWIFT CODE	If it exists, its length must be	Either 8 or 11
INTERMEDIARY BANK ACCT NUMBER	If it exists, its length must be	1 to 34
Illegal Characters	Should not contain one of the following special characters: `:@!%&*#{}<=^>|;$~\\\"`	All of these are mapped to the ? character

A.7.3 SWIFT MT104

The SWIFT MT104 message is sent by a corporation to a financial institution to request the direct debit of the debtor's bank account in the receiver's country and, subsequently, to credit the creditor's bank account maintained by the receiving institution or one of its branches. The SWIFT MT104 message is only appropriate to a "request for debit transfer" scenario.

A.7.3.1 SWIFT MT104 header block

The following table presents the contents of the SWIFT MT104 header block:

Tag	CMM mapping	Example
Sender	Interchange sender ID	`PLATUS33`
Message type	Static value `104`	`104`
Receiver	Interchange receiver ID	`AAAAUS33`

A.7.3.2 SWIFT MT104 main message block

The main message block of a SWIFT MT104 message consists of the following sequences:

Letter	Name	Required	Number of occurrences	Contents
A	General information	Yes	One	Information to be applied to all individual transactions detailed in sequence B
B	Transaction details	Yes	Multiple	Details of one individual transaction
C	Settlement details	No	Multiple	Further settlement information for all transactions mentioned in sequence B Note: When the message is used as a request for direct debit, this sequence is not used. When the message is used as a direct debit, this sequence is mandatory.

The following table presents the available tags:

Tag	Name	Required	Contents [1]	CMM mapping	Example
Sequence A: General information					
20	Sender's reference	M	16x	Import/export ID	`:20:PLANT-12345`
21R	Customer specified reference	O	16x	• In the batch format: customer reference ID • In the transaction format: N/A	`:21R:PLTOL104-56`
23E	Instruction code	O	30x	The type of direct debit instructions contained in the message (for example, `:23E:RFDD`). Note: CMM inserts the static value `RFDD` in this tag, indicating that the message contains a request for direct debit instructions.	`:23E:RFDD`
21E	Registration reference	O	35x	N/A	N/A
30	Requested execution date	M	6!n	Value date	`:30:060929`
50A	Sending institution	O	N/A	N/A	N/A

Tag	Name	Required	Contents [1]	CMM mapping	Example
50a	Instructing party	O	C or L	• When the sender is the bank account owner: N/A • When the sender is not the bank account owner and is not the shared service center: Interchange sender ID • When the sender is the shared service center: N/A	:50C:SPRNGB2L
50a	Creditor	O	A or K	• When the sender is the shared service center: N/A • For pay-account-identical: - When the sender is the bank account owner: Use 50A and map to the bank account number and BEI - When the sender is not the account owner: Use 50K and map to the bank account number, party long name, and party local address	:50A:289523984:SPRNGB2L
52a	Creditor's bank	O	A, C, or D	When pay-account-identical and aggregated bank interchange: Use 52A and map to the bank BIC	:52A:ZZZZUS33
26T	Transaction type code	O	3!c	Transaction type code	:26T:COM
77B	Regulatory reporting	O	3*35x	The static value BENEFRES, followed by the debtor cpty bank country code (if it exists), followed by the party company identifier	:77B:BENEFRESUSSPRNGB2L
71A	Details of changes	O	3!a	N/A	N/A
72	Sender-to-receiver information	O	6*35x	N/A	N/A
Sequence B: Transaction details (repeat once for each transaction in the group)					
21	Transaction reference	M	16x	Customer reference ID	:21:TR1-PL
23E	Instruction code	O	4!c[/30x]	Static value AUTH	:23E:AUTH
21C	Mandate reference	O	35x	N/A	N/A
21D	Direct debit reference	O	35x	N/A	N/A
21E	Registration reference	O	35x	N/A	N/A

Tag	Name	Required	Contents [1]	CMM mapping	Example
32B	Currency/transaction amount	M	3!a15d	Currency and amount	:32B:USD118982,05
50a	Instructing party	O	C or L	Originator long name	N/A
50a	Creditor	O	A or K	Party bank account IBAN or BBAN then party long name and part local address	:50A:289523984:SPRNGB2L
52a	Creditor's bank	O	A ([/1!a][/34x]4!a2!a2!c[3!c]) or C (/34x)	Party bank account number	:52A:ZZZZUS33
57a	Debtor's bank	O	A, C, or D	• 57A: counterparty bank SWIFT code • 57D: counterparty bank local address	:57A:CCCCGB2L
59a	Debtor	M	A or no option letter	• If the payee has a BIC or BEI: Post :59A:+/counterparty bank account IBAN or BBAN+BIC or BEI • If the payee does not have a BIC or BEI: Post :59:+"counterparty bank account number"+counterparty name and address	:59:/GB111111THOMSON FACTORY GB-LIVERPOOL

Tag	Name	Required	Contents [1]	CMM mapping	Example
70	Remittance information	O	4*35x	Beneficiary message • INV: If remittance details are available, build INV based on the details. • ROC: If remittance details are not available, build ROC with the customer reference ID. • IPI: If the counterparty message is not empty, build IPI with the counterparty message trimmed to 20 characters. (Users can put a unique reference identifying a related international payment in the counterparty message.) • RFB: If the beneficiary customer BIC or BEI is available, build RFB with the BIC or BEI. Otherwise, build RFB with the beneficiary long name truncated to 16 characters. • ULTB: N/A. (It is assumed the beneficiary is always the ultimate beneficiary.) • INST: If the instructing party is different from the ordering customer, build INST with the instructing party. The order of these items in the tag is as follows: **1.** INV or DOC **2.** IPI **3.** RFB **4.** INST	:70:/INV/AUG06-345//AUG06-346 :70:/ROC/RA-PL-9876-87 /INSTPLANTOIL AXIM :70:ROC/RA-PL-9876-103 /INST/PLANTOIL PRODUCTIONS :70:/INVTH9876//TH9877 /INST/SPRINGS INC :70:INV/MLRAB98 /INST/SPRINGS INC :70:/ROC/BGF89765 /INST/BIOGEN FRANCE :70:/INV/BW654//BW655 //BW656 /INST/BIOGEN HONG KONG
26T	Transaction type code	O	3!c	Transaction type code	:26T:COM
77B	Regulatory reporting	O	3*35x	Regulatory code Note: Only post this tag for cross-border payments.	:77B:/BENEFRES/US//CVS//060
33B	Currency/original ordered amount	O	3!a15d	N/A	N/A

Tag	Name	Required	Contents [1]	CMM mapping	Example
71A	Details of charges	O	3!a	Financial charge allocation: • BEN: All transaction charges, including the charges of the financial institution serving the order customer's bank account, for the subsequent credit transfers are to be borne by the beneficiary customer. • OUR: All transaction charges for the subsequent credit transfer are to be borne by the ordering customer. • SHA: All transaction charges other than the charges of the financial institution servicing the ordering customer bank account are borne by the beneficiary customer.	`:71A:SHA`
71F	Sender's charges	O	3!a15d	N/A	N/A
71G	Receiver's charges	O	3!a15d	N/A	N/A
36	Exchange rate	O	12d	N/A	N/A

Table notes:

1. For more information on the codes used in this column, see A.13.1 SWIFT field lengths on page 303 and A.13.2 SWIFT field types on page 304.

A.7.4 SWIFT MT210

The SWIFT MT210 message is sent by a bank account owner (or a party authorized by the bank account owner) to one of its bank account service institutions. It is an advice notice to the bank account servicing institution that indicates that the bank account owner will receive funds to be credited to its bank account.

The SWIFT MT210 message provides the bank account servicing institution with early visibility into incoming funds, allowing the institution to mange its liquidity and ensure the bank account owner is credited with good value date.

A.7.4.1 SWIFT MT210 header block

The following table presents the contents of the SWIFT MT210 header block:

Tag	CMM mapping	Example
Sender	Interchange sender ID	`PLATUS33`
Message type	Static value `210`	`210`
Receiver	Interchange receiver ID	`AAAAUS33`

A.7.4.2 SWIFT MT210 main message block

The following table presents the available tags in the main message block of a SWIFT MT210 message:

Tag	Name	Required	Contents [1]	CMM mapping	Example
20	Transaction reference number	M	16x	Customer reference ID	`:20:txn0001`
25	Account identification	O	35X	Party bank account number Note: This tag is used when the receiver services more than one bank account for the sender	`:25:987678998`
30	Value date	M	6!n	Value date	`:30:070711`
21	Related reference	M	16x	Customer reference ID Note: This tag contains the same value as tag 20 (transaction reference number).	`:21:txn0001`
32B	Currency/transaction amount	M	3!a15d	Currency and amount	`:32B:GBP675100,`
50C 50	Ordering customer	O	4!a2!a2!c[3!c] (for 50C) 4*35x (for 50)	• If the ordering customer has a BEI: Party SWIFT code • If the ordering customer does not have a BEI: Party long name followed by party address lines 1 to 4 Note: The first option is preferred. The second option should only be used when the ordering customer does not have a BEI.	`:50C:CorPSWFT` `:50:corporation Corporation address`
52A 52D	Ordering institution	C2	[/1!a][/34x] 4!a2!a2!c[3!x] (for 52A) [/1!a][/34x] 4*35x (for 52D)	• If the financial institution has a BIC: Party SWIFT code • If the financial intuition does not have a BIC: Party long name followed by party address lines 1 to 4 Note: The first option is preferred. The second option should only be used when the financial institution does not have a BIC.	`:52A:BANKSWIFT01` `:52D:bank name address 1 address 2`

Tag	Name	Required	Contents [1]	CMM mapping	Example
56A 56D	Intermediary	O	[/1!a][/34x] 4!a2!a2!c[3!x] (for 56A) [/1!a][/34x] 4*35x (for 56D)	• If the intermediary has a BIC: Intermediary bank SWIFT code • If the intermediary does not have a BIC: Intermediary bank long name followed by intermediary bank local address lines 1 to 4 Note: The first option is preferred. The second option should only be used when the intermediary does not have a BIC.	:56A:INTERMBANK1 :56D:Intermediary_bank_name Intermediary_bank_address

Table notes:

1. For more information on the codes used in this column, see A.13.1 SWIFT field lengths on page 303 and A.13.2 SWIFT field types on page 304.

A.7.5 ISO 20022 XML credit transfer

The ISO 20022 XML standard is an effort of financial institution, corporations, and vendors to define standard message formats for financial transactions. SWIFT is heavily involved in the standard by providing expertise and facilitating meetings.

Using CMM's XML template tool, Wallstreet has implemented a subset of the initial ISO 20022 XML standard formats—including the credit transfer format. You can use Wallstreet's initial work as the basis for implementing the ISO 20022 XML standard. You can also customize the formats to accommodate your banks' unique interpretations of the standard.

The ISO 20022 XML credit transfer message is sent from the initiating party to the debtor agent or forwarding agent. It is used to request movement of funds from a debtor account to a creditor. Depending on the service level agreed between the debtor agent and the initiating party, the debtor agent may send a payment status message to inform the initiating party of the status of the initiation. (For more information on ISO 20022 XML payment status messages, see A.4.3 ISO 20022 XML payment status on page 201.)

The ISO 20022 XML credit transfer message:

- Can contain one or more customer credit transfer instructions
- Can be used by an initiating party that has authority to send the message on behalf of the debtor

  This allows for the scenarios where a payment factory initiates all payments on behalf of a large corporation.
- Can be used for domestic or cross-border scenarios.

Three types of customer roles can be played on the initiating side:

Name	Description
Initiating party	The party sending the message (payment factory or treasury)
Debtor	The debit account owner (for example, an in-house bank)
Ultimate debtor	The party that owes the cash to the creditor as a result of the receipt of goods or serves (for example, a subsidiary).

Two types of customer roles can be played on the receiving side:

Name	Description
Creditor	The credit account owner
Ultimate creditor	The party that is the ultimate beneficiary of the cash transfer

These roles can be played by the same actor or by different actors. The message allows inclusion of the different roles on the initiating side and receiving side.

Note: A relay credit transfer scenario consists of the initiating party sending a message to the forwarding agent, which will then forward the message to the debitor agent.

The ISO 20022 XML credit transfer message consists of the following sections:

Name	Required	Number of occurrences	Contents
Group header	Yes	One	Element such as MessageIdentification, CreationDateAndTime, and Grouping
Payment information	Yes	Multiple	Elements related to the debit side of the transaction, such as Debtor and PaymentTypeInformation
Credit transfer transaction information	Yes	Multiple	Elements related to the credit side of the transaction, such as Creditor and RemittanceInformation

A.7.5.1 ISO 20022 XML credit transfer format

The following table presents the format of ISO 20022 XML credit transfer messages:

Message item	XML tag	Mult/format	Data type	CMM mapping and posting
Group header	`<GrpHdr>`	[1..1]	N/A	N/A
MessageIdentification	`<MsgID>`	[1..1]/[1,35]	Text	Map to `import_export_id`
CreationDateTime	`<CreDtTm>`	[1//1]/	Date/time	Post current date/time in format `YYYY-MM-DDTHH:MM:SS`
Authorization	`<Authstn>`	[0..2]/[1,120]	Text	N/A
BatchBooking	`<BtchBookg>`	[0..1]/MeaningWhenTrue or MeaningWhenFalse	Indicator	Post one of the following: • `true` • `false`
NumberOfTransactions	`<NbOfTxs>`	[1..1]/[0-9]{1,15}	Text	Post valid transaction number
ControlSum	`<CtrlSum>`	[0..1]/17 or 18	Quantity	Post total of all individual amounts by runtime calculation

Message item	XML tag	Mult/format	Data type	CMM mapping and posting
Grouping	<Grpg>	[1..1]/GRPD, MIXD, SNGL	Code	Post default value MIXD Note: Mixed case (MIXD) covers both single and grouped cases. Mixed allows the payment information block to be present once or may be repeated. Each sequence of the payment information block may contain one or more credit transfer transaction information block(s).
InitiatingParty	<InitgPty>	[1..1]/<Nm> <PstlAdr> <Id> <CtryOfRes>	±	Post sending party's PartyIdentification information at the interchange level
ForwardingAgent	<FwdgAgt>	[0..1]/<FinInstnId> <Brnchld>	±	Post receiving institution's BranchAndFinancialInstitutionIdentification information at the interchange level if receiving is an aggregated bank
Payment information	<PmtInf>	[1..n]	N/A	N/A
PaymentInformationIdentification	<PmtInfId>	[0..1]/[1,35]	Text	N/A
PaymentMethod	<PmtMtd>	[1..1]	Code	Post one of the following: • CHK for cheque • TRA for transfer advice • TRF for credit transfer
PaymentTypeInformation	<PmtTpInf>	[0..1]	N/A	Only post <InstrPrty> (NORM or HIGH) based on priority_code

Message item	XML tag	Mult/format	Data type	CMM mapping and posting
RequestedExecutionDate	<ReqdExctnDt>	[1..1]	Date/time	Map to value_date in ISO date format
PoolingAdjustmentDate	<PoolgAdjstmntDt>	[0..1]	Date/time	N/A
Debtor	<Dbtr>	[1..1]	±	Post payor's PartyIdentification information
DebtorAccount	<DbtrAcct>	[1..1]	±	Post payor bank account's CashAccount information
DebtorAgent	<DbtrAgt>	[1..1]	±	Post payor bank's BranchAndFinancialInstituationIdentification information
DebtorAgentAccount	<DbtrAgtAcct>	[0..1]	±	N/A
UltimateDebtor	<UltmtDbtr>	[0..1]	±	N/A
ChargeBearer	<ChrgBr>	[0..1]	Code	N/A
ChargesAccount	<ChrgsAcct>	[0..1]	±	N/A
ChargesAccountAgent	<ChrgsAcctAgt>	[0..1]	±	N/A
CreditTransferTransactionInformation	<CdtTrfTxInf>	[1..n]	N/A	N/A
Credit transfer transaction information	<CdtTrfTxInf>	[1..n]	N/A	N/A
PaymentInformationIdentification	<PmtInfId>	[0..1]/[1,35]	Text	N/A
PaymentIdentification	<PmtId>	[1..1]	N/A	Post <EndToEndId> to map the customer reference ID
PaymentTypeInformation	<PmtTpInf>	[0..1]	N/A	Only post <InstrPrty> (NORM or HIGH) based on priority_code
Amount	<Amt>	[1..1]	N/A	Post <InstdAmt> based on payment_amount and payment_currency
ExchangeRateInformation	<XchgRateInf>	[0..1]	N/A	N/A

Message item	XML tag	Mult/format	Data type	CMM mapping and posting
ChargeBearer	<ChrgBr>	[0..1]	Code	Post default value SHAR
ChequeInstruction	<ChqInstr>	[0..1]	±	Post Cheque information if payment_method is CHQ
UltimateDebtor	<UltmtDbtr>	[0..1]	±	Post originator's PartyIdentification information if the originator is different than the debtor
IntermediaryAgent1	<IntrmyAgt1>	[0..1]	±	Post intermediary bank's BranchAndFinancialInstitutionIdentification information
IntermediaryAgent1Account	<IntrmyAgt1Acct>	[0..1]	±	Post intermediary bank account's CashAccount information
IntermediaryAgent2	<IntrmyAgt2>	[0..1]	±	N/A
IntermediaryAgent2Account	<IntrmyAgt2Acct>	[0..1]	±	N/A
IntermediaryAgent3	<IntrmyAgt3>	[0..1]	±	N/A
IntermediaryAgent3Account	<IntrmyAgt3Acct>	[0..1]	±	N/A
CreditorAgent	<CdtrAgt>	[0..1]	±	Post payee bank's BranchAndFinancialInstitutionIdentification information
CreditorAgentAccount	<CdtrAgtAcct>	[0..1]	±	N/A
Creditor	<Cdtr>	[0..1]	±	Post payee's PartyIdentification information
CreditorAccount	<CdtrAcct>	[0..1]	±	Post payee bank account's CashAccount information
UltimateCreditor	<UltmtCdtr>	[0..1]	±	N/A
InstructionForCreditorAgent	<InstrForCdtrAgt>	[0..n]	N/A	Map to beneficiary_message
InstructionForDebtorAgent	<InstrForDbtrAgt>	[0..1]	Text	Map to bank_instruction

Message item	XML tag	Mult/format	Data type	CMM mapping and posting
Purpose	<Purp>	[0..1]	N/A	Map to payment_description
RegulatoryReporting	<RgltryRptg>	[0..10]	N/A	N/A
Tax	<Tax>	[0..1]	±	Post TaxInformation (if any)
RelatedRemittanceInformation	<RltdRmInf>	[0..10]	N/A	N/A
RemittanceInformation	<RmtInf>	[0..1]	N/A	Post RemittanceInformation
PartyIdentification	N/A	N/A	N/A	N/A
Name	<Nm>	[0..1]	Text	Post party's long name
PostalAddress	<PstlAdr>	[0..1]	N/A	Post party's postal address
AddressType	<AdrTp>	[0..1]	Code	Post default value ADDR
AddresssLine	<AdrLine>	[0..5]	Text	Post party's ADDRESS1, ADDRESS2, ADDRESS3, ADDRESS4
StreetName	<StrtNm>	[0..1]	Text	N/A
BuildingNumber	<BldgNb>	[0..1]	Text	N/A
PostCode	<PstCd>	[0..1]	Text	Post party's POSTALCODE
TownName	<TwnNm>	[0..1]	Text	Post party's CITY
CountrySubDivision	<CtrySubDvsn>	[0..1]	Text	Post party's STATE
Country	<Ctry>	[1..1]	Code	Post party's COUNTRYCODE
Identification	<Id>	[0..1]	N/A	Post party's organization identification
BIC	<BIC>	[0..1]	Identifier	Post party's SWIFT code if the party is an institution
IBEI	<IBEI>	[0..1]	Identifier	N/A
BEI	<BEI>	[0..1]	Identifier	Post party's SWIFT code if the party is a corporation
EANGLN	<EANGLN>	[0..1]	Identifier	N/A

Message item	XML tag	Mult/format	Data type	CMM mapping and posting
CHIPSUniversalIdentification	<USCHU>	[0..1]	Identifier	N/A
DUNS	<DUNS>	[0..1]	Identifier	N/A
BankPartyIdentification	<BkPtyId>	[0..1]	Text	N/A
TaxIdentificationName	<TaxIdNb>	[0..1]	Text	N/A
ProprietaryIdentification	<PrtryId>	[0..1]	N/A	N/A
Identification	<Id>	[1..1]	Text	N/A
Issuer	<Issr>	[0..1]	Text	N/A
CountryOfResidence	<CtryOfRes>	[0..1]	Code	Post party's country code
BranchAndFinancialInstitutionIdentification	N/A	N/A	N/A	N/A
FinancialInstitutionIdentification	<FinInstnID>	[1..1]	N/A	Post financial institution's identification
BIC	<BIC>	[1..1]	Identifier	Post bank's SWIFT code
ClearingSystemMemberIdentification	<ClrSysMmbId>	[1..1]	N/A	N/A
NameAndAddress	<NmAndAdr>	[1..1]	N/A	N/A
Name	<Nm>	[1..1]	N/A	Post bank's name
Address	<PstlAdr>	[1..1]	N/A	Post bank's address
ProprietaryIdentification	<PrtryId>	[1..1]	N/A	N/A
CombinedIdentification	<CmbndId>	[1..1]	N/A	N/A
BranchIdentification	<BrnchId>	[0..1]	N/A	N/A
CashAccount	N/A	N/A	N/A	N/A
Identification	<Id>	[1..1]	N/A	N/A
IBAN	<IBAN>	[1..1]	Identifier	Post IBAN
BBAN	<BBAN>	[1..1]	Identifier	Post BBAN
UPIC	<UPIC>	[1..1]	Identifier	N/A
ProprietaryAccount	<PrtryAcct>	[1..1]	N/A	N/A

Message item	XML tag	Mult/format	Data type	CMM mapping and posting
Type	<Tp>	[0..1]	N/A	Post bank account's type ID
Currency	<Ccy>	[0..1]	Code	Post bank account's currency code
Name	<Nm>	[0..1]	Text	Post bank account's name
Cheque	N/A	N/A	N/A	N/A
ChequeType	<ChqTp>	[0..1]	Code	Post default value BCHQ (bank check)
ChequeNumber	<ChqNb>	[0..1]	Text	Post cheque_number
ChequeFrom	<ChqFr>	[0..1]	N/A	Post payor's long name and postal address
DeliveryMethod	<DlvryMtd>	[0..1]	N/A	N/A
DeliverTo	<DlvrTo>	[0..1]	N/A	Post payee's long name and postal address
InstructionPriority	<InstrPrty>	[0..1]	Code	Map to priority_code
ChequeMaturityDate	<ChqMtrtyDt>	[0..1]	Date/time	Map to value_date
FormsCode	<FrmsCd>	[0..1]	Text	N/A
MemoField	<MemoFld>	[0..1]	Text	N/A
RegionalClearingZone	<RgnlClrZone>	[0..1]	Text	N/A
PrintLocation	<PrtLctn>	[0..1]	Text	N/A
TaxInformation	N/A	N/A	N/A	N/A
CreditorTaxIdentification	<CdtrTaxId>	[0..1]	Text	N/A
CreditorTaxType	<CdtrTaxTp>	[0..1]	Text	N/A
DebtorTaxIdentficiation	<DbtrTaxId>	[0..1]	Text	Map to tax_payer_id
TaxReferenceNumber	<TaxRefNb>	[0..1]	Text	Map to tax_payer_verification_id
TotalTaxableBaseAmount	<TtlTaxblBaseAmt>	[0..1]	Amount	N/A
TotalTaxAmount	<TtlTaxAmt>	[0..1]	Amount	Map to payment_amount
TaxDate	<TaxDt>	[0..1]	Date/time	Map to value_date

Message item	XML tag	Mult/format	Data type	CMM mapping and posting
TaxTypeInformation	<TaxTpInf>	[0..n]	N/A	N/A
RemittanceInformation	<RmtInf>	N/A	N/A	N/A
Structured format	<Strd>	N/A	N/A	N/A
ReferredDocumentInformation	<RfrdDocInf>	[0..1]	N/A	N/A
Referred DocumentType	<RfrdDocTp>	[0..1]	N/A	Map to remittance_detail_message_type
ReferredDocument Number	<RfrdDocNb>	[0..1]	Type	Map to remittance_invoice_number
ReferredDocumentRelatedDate	<RfrdDocRltdDt>	[0..1]	Date/time	Map to remittance_date
ReferredDocumentAmount	<RfrdDocAmt>	[0..n]	N/A	Map to document_amount
CreditorReferenceInformation	<CdtrRefInf>	[0..1]	N/A	N/A
Invoicer	<Invcr>	[0..1]	±	N/A
Invoicee	<Invcee>	[0..1]	±	N/A
AdditionalRemittanceInformation	<AddtlRmtInf>	[0..1]	Text	N/A

A.7.5.2 ISO 20022 XML credit transfer rules

This section defines rules for ISO 20022 XML credit transfer messages.

A.7.5.2.1 Grouping

If GroupHeader/Grouping is present and equals GRPD, one and only one occurrence of PaymentInformation must be present.

If GroupHeader/Grouping is present and equals SNGL, each occurrence of PaymentInformation must contain one and only one occurrence of PaymentInformation/CreditTransferTransactionInformation.

A.7.5.2.2 ChargeBearer

If ChargeBearer is present, CreditTransferTransactionInformation/ChargeBearer is not allowed.

If CreditTransferTransactionInformation/ChargeBearer is present, ChargeBearer is not allowed.

CreditTransferTransactionInformation/ChargeBearer and ChargeBearer may both be absent.

A.7.5.2.3 ChargesAccountAgentRule

If ChargesAccountAgent is present, it must contain a branch of the DebtorAgent. It must not contain a completely different financial institution.

A.7.5.2.4 ChargesAccountRule

If ChargesAccountAgent is present, ChargesAccount must be present.

A.7.5.2.5 ChequeInstructionRule

If PaymentMethod is `CHK`, CreditTransferTransactionInformation/ChequeInstruction is optional.

If PaymentMethod is different from `CHK`, CreditTransferTransactionInformation/ChequeInstruction is not allowed.

Rule rationale: ChequeInstructionDetails may be present if the payment method is `Cheque`. It must not be present if the payment method is `Transfer`.

A.7.5.2.6 CreditorAgentRule

If PaymentMethod is `CHK` and if CreditTransferTransactionInformation/ChequeInstruction/DeliveryMethod is present and is equal to `MLFA`, `CRFA`, `RGFA`, or `PUFA`, CreditTransferTransactionInformation/ CreditorAgent is mandatory.

If PaymentMethod is `CHK` and if CreditTransferTransactionInformation/ ChequeInstruction/DeliveryMethod is not present or is not equal to `MLFA`, `CRFA`, `RGFA`, or `PUFA`, CreditTransferTransactionInformation/CreditorAgent is not allowed.

A.7.5.2.7 CreditorAndOrCreditorAccountRule

If PaymentMethod is `CHK`, CreditTransferTransactionInformation/CreditorAccount is not allowed.

If PaymentMethod is different from `CHK` and if CreditTransferTransactionInformation/Creditor is not present, CreditTransferTransactionInformation/CreditorAccount is mandatory.

If PaymentMethod is different from `CHK` and if CreditTransferTransactionInformation/Creditor is present, CreditTransferTransactionInformation/CreditorAccount is optional.

A.7.5.2.8 PaymentTypeInformationRule

If PaymentTypeInformation is present, CreditTransferTransactionInformation/PaymentTypeInformation is not allowed.

A.7.5.2.9 ChequeFromGuideline

CreditTransferTransactionInformation/ChequeInstruction/ChequeFrom may only be present if different from CreditTransferTransactionInformation/UltimateDebtor or Debtor.

A.7.5.2.10 ChequeInstructionDeliverToCreditorAgentGuideline

If CreditTransferTransactionInformation/ChequeInstruction/DeliveryMethod is present and is `CRFA`, `MLFA`, `PUFA`, or `RGFA`, CreditTransferTransactionInformation/ChequeInstruction/DeliverTo may only be present if different than CreditTransferTransactionInformation/Creditor.

A.7.5.2.11 ChequeInstructionDeliverToCreditorGuideline

If PaymentInformation/CreditTransferTransactionInformation/ChequeInstruction/DeliveryMethod is present and is CRCD or MLCD or PUCD or RGCD, CreditTransferTransactionInformation/ChequeInstruction/DeliverTo may only be present if different from CreditTransferTransactionInformation/ Creditor.

A.7.5.2.12 ChequeInstructionDeliverToDebtorGuideline

If CreditTransferTransactionInformation/ChequeInstruction/DeliveryMethod is present and if CreditTransferTransactionInformation/ChequeInstruction/DeliveryMethod/Code is `CRDB`, `MLDB`, `PUDB`, or `RGDB`, CreditTransferTransactionInformation/ChequeInstruction/DeliverTo may only be present if different than Debtor.

A.7.5.2.13 UltimateDebtorGuideline

UltimateDebtor may only be present if different from Debtor.

A.7.5.2.14 InstructionForCreditorAgentRule

If InstructionForCreditorAgent/Code contains `CHQB`, CreditorAccount is not allowed.

A.7.5.2.15 IntermediaryAgent1AccountRule

If IntermediaryAgent1 is not present, IntermediaryAgent1Account is not allowed.

A.7.5.2.16 IntermediaryAgent2AccountRule

If IntermediaryAgent2 is not present, IntermediaryAgent2Account is not allowed.

A.7.5.2.17 IntermediaryAgent2Rule

If IntermediaryAgent2 is present, IntermediaryAgent1 must be present.

A.7.5.2.18 IntermediaryAgent3AccountRule

If IntermediaryAgent3 is not present, IntermediaryAgent3Account is not allowed.

A.7.5.2.19 IntermediaryAgent3Rule

If IntermediaryAgent3 is present, IntermediaryAgent2 must be present.

A.7.5.2.20 UltimateCreditorGuideline

UltimateCreditor may only be present if different from Creditor.

A.7.5.2.21 UltimateDebtorGuideline

UltimateDebtor may only be present if different from Debtor.

A.7.5.2.22 ChequeMaturityDateRule

If ChequeType is present and is `DRFT` or `ELDR`, ChequeMaturityDate is optional.

If ChequeType is not present or is different from `DRFT` or `ELDR`, ChequeMaturityDate is not allowed.

Rule rationale: ChequeMaturityDate may be present only when ChequeType is `DRFT` or `ELDR`.

A.7.5.2.23 Country

The code is checked against the list of country names obtained from the United Nations (ISO 3166, Alpha-2 code).

A.7.6 ISO 20022 XML direct debit transfer

The ISO 20022 XML standard is an effort of financial institution, corporations, and vendors to define standard message formats for financial transactions. SWIFT is heavily involved in the standard by providing expertise and facilitating meetings.

Using CMM's XML template tool, Wallstreet has implemented a subset of the initial ISO 20022 XML standard formats—including the direct debit transfer format. You can use Wallstreet's initial work as

the basis for implementing the ISO 20022 XML standard. You can also customize the formats to accommodate your banks' unique interpretations of the standard.

The ISO 20022 XML direct debit transfer message is sent by the initiating party to the forwarding agent or creditor agent. It is used to request single or bulk collection(s) of funds from one or various debtor's account(s) for a creditor.

The ISO 20022 XML direct debit transfer message consists of the following sections:

Name	Required	Number of occurrences	Contents
Group header	Yes	One	Element such as MessageIdentification, CreationDateAndTime, and Grouping
Payment information	Yes	Multiple	Elements related to the credit side of the transaction, such as Creditor and PaymentTypeInformation
Credit transfer transaction information	Yes	Multiple	Elements related to the debit side of the transaction, such as Debitor and RemittanceInformation

A.7.6.1 ISO 20022 XML direct debit transfer format

The following table presents the format of ISO 20022 XML direct debit transfer messages:

Message item	XML tag	Mult/format	Data type	CMM mapping and posting
Group header	`<GrpHdr>`	[1..1]	N/A	N/A
MessageIdentification	`<MsgID>`	[1..1]/[1,35]	Text	Map to `import_export_id`
CreationDateTime	`<CreDtTm>`	[1//1]/	Date/time	Post current date/time in format `YYYY-MM-DDTHH:MM:SS`
Authorization	`<Authstn>`	[0..2]/[1,120]	Text	N/A
BatchBooking	`<BtchBookg>`	[0..1]/MeaningWhenTrue or MeaningWhenFalse	Indicator	Post one of the following: • `true` • `false`
NumberOfTransactions	`<NbOfTxs>`	[1..1]/[0-9]{1,15}	Text	Post valid transaction number
ControlSum	`<CtrlSum>`	[0..1]/17 or 18	Quantity	Post total of all individual amounts by runtime calculation
Grouping	`<Grpg>`	[1..1]/`GRPD`, `MIXD`, `SNGL`	Code	Post default value `MIXD` Note: Mixed case (`MIXD`) covers both single and grouped cases. Mixed allows the payment information block to be present once or may be repeated. Each sequence of the payment information block may contain one or more direct debit transfer transaction information block(s).

Message item	XML tag	Mult/format	Data type	CMM mapping and posting
InitiatingParty	<InitgPty>	[1..1]/<Nm> <PstlAdr> <Id> <CtryOfRes>	±	Post sending party's PartyIdentification information at the interchange level
ForwardingAgent	<FwdgAgt>	[0..1]/<FinInstnId> <Brnchld>	±	Post receiving institution's BranchAndFinancialInstitutionIdentification information at the interchange level if receiving is an aggregated bank
Payment information	<PmtInf>	[1..n]	N/A	N/A
PaymentInformationIdentification	<PmtInfId>	[0..1]/[1,35]	Text	N/A
PaymentMethod	<PmtMtd>	[1..1]	Code	Post DD
PaymentTypeInformation	<PmtTpInf>	[0..1]	N/A	Only post <InstrPrty> (NORM or HIGH) based on priority_code
RequestedCollectionDate	<ReqdCollDt>	[1..1]	Date/time	Map to value_date in ISO date format
Creditor	<Cdtr>	[1..1]	±	Post payor's PartyIdentification information
CreditorAccount	<CdtrAcct>	[1..1]	±	Post payor bank account's CashAccount information
CreditorAgent	<CdtrAgt>	[1..1]	±	Post payor bank's BranchAndFinancialInstituationIdentification information
CreditorAgentAccount	<CdtrAgtAcct>	[0..1]	±	N/A
UltimateCreditor	<UltmtCdtr>	[0..1]	±	N/A
ChargeBearer	<ChrgBr>	[0..1]	Code	N/A
ChargesAccount	<ChrgsAcct>	[0..1]	±	N/A
ChargesAccountAgent	<ChrgsAcctAgt>	[0..1]	±	N/A
CreditTransferTransactionInformation	<CdtTrfTxInf>	[1..n]	N/A	N/A
Direct debit transaction information	<CdtTrfTxInf>	[1..n]	N/A	N/A
PaymentIdentification	<PmtId>	[1..1]	N/A	Post <EndToEndId> to map the customer reference ID
PaymentTypeInformation	<PmtTpInf>	[0..1]	N/A	Only post <InstrPrty> (NORM or HIGH) based on priority_code
InstructedAmount	<Amt>	[1..1]	N/A	Post <InstdAmt> based on payment_amount and payment_currency

Message item	XML tag	Mult/format	Data type	CMM mapping and posting
ChargeBearer	<ChrgBr>	[0..1]	Code	Post default value SHAR
DirectDebitTransaction	<DrctDbtTx>	[0..1]	N/A	N/A
UltimateCreditor	<UltmtCdtr>	[0..1]	±	Post originator's PartyIdentification information if the originator is different than the creditor
DebitorAgent	<DbtrAgt>	[0..1]	±	Post counterparty bank's BranchAndFinancialInstitutionIdentification information
DebitorAgentAccount	<DbtrAgtAcct>	[0..1]	±	N/A
Debitor	<Dbtr>	[0..1]	±	Post counterparty's PartyIdentification information
DebitorAccount	<DbtrAcct>	[0..1]	±	Post countparty bank account's CashAccount information
UltimateDebitor	<UltmtDbtr>	[0..1]	±	N/A
InstructionForCreditorAgent	<InstrForCdtrAgt>	[0..n]	N/A	Map to bank_instruction
Purpose	<Purp>	[0..1]	N/A	Map to payment_description
RegulatoryReporting	<RgltryRptg>	[0..10]	N/A	N/A
Tax	<Tax>	[0..1]	±	N/A
RelatedRemittanceInformation	<RltdRmInf>	[0..10]	N/A	N/A
RemittanceInformation	<RmtInf>	[0..1]	N/A	Post unstructured format information Map to beneficiary_message
PartyIdentification	N/A	N/A	N/A	N/A
Name	<Nm>	[0..1]	Text	Post party's long name
PostalAddress	<PstlAdr>	[0..1]	N/A	Post party's postal address
AddressType	<AdrTp>	[0..1]	Code	Post default value ADDR
AddresssLine	<AdrLine>	[0..5]	Text	Post party's ADDRESS1, ADDRESS2, ADDRESS3, ADDRESS4
StreetName	<StrtNm>	[0..1]	Text	N/A
BuildingNumber	<BldgNb>	[0..1]	Text	N/A
PostCode	<PstCd>	[0..1]	Text	Post party's POSTALCODE
TownName	<TwnNm>	[0..1]	Text	Post party's CITY
CountrySubDivision	<CtrySubDvsn>	[0..1]	Text	Post party's STATE

Message item	XML tag	Mult/format	Data type	CMM mapping and posting
Country	<Ctry>	[1..1]	Code	Post party's COUNTRYCODE
Identification	<Id>	[0..1]	N/A	Post party's organization identification
BIC	<BIC>	[0..1]	Identifier	Post party's SWIFT code if the party is an institution
IBEI	<IBEI>	[0..1]	Identifier	N/A
BEI	<BEI>	[0..1]	Identifier	Post party's SWIFT code if the party is a corporation
EANGLN	<EANGLN>	[0..1]	Identifier	N/A
CHIPSUniversalIdentification	<USCHU>	[0..1]	Identifier	N/A
DUNS	<DUNS>	[0..1]	Identifier	N/A
BankPartyIdentification	<BkPtyId>	[0..1]	Text	N/A
TaxIdentificationName	<TaxIdNb>	[0..1]	Text	N/A
ProprietaryIdentification	<PrtryId>	[0..1]	N/A	N/A
Identification	<Id>	[1..1]	Text	N/A
Issuer	<Issr>	[0..1]	Text	N/A
CountryOfResidence	<CtryOfRes>	[0..1]	Code	Post party's country code
BranchAndFinancialInstitutionIdentification	N/A	N/A	N/A	N/A
FinancialInstitutionIdentification	<FinInstnID>	[1..1]	N/A	Post financial institution's identification
BIC	<BIC>	[1..1]	Identifier	Post bank's SWIFT code
ClearingSystemMemberIdentification	<ClrSysMmbId>	[1..1]	N/A	N/A
NameAndAddress	<NmAndAdr>	[1..1]	N/A	N/A
Name	<Nm>	[1..1]	N/A	Post bank's name
PostalAddress	<PstlAdr>	[1..1]	N/A	Post bank's address
ProprietaryIdentification	<PrtryId>	[1..1]	N/A	N/A
CombinedIdentification	<CmbndId>	[1..1]	N/A	N/A
BranchIdentification	<BrnchId>	[0..1]	N/A	N/A
CashAccount	N/A	N/A	N/A	N/A

Message item	XML tag	Mult/format	Data type	CMM mapping and posting
Identification	<Id>	[1..1]	N/A	N/A
IBAN	<IBAN>	[1..1]	Identifier	Post IBAN
BBAN	<BBAN>	[1..1]	Identifier	Post BBAN
UPIC	<UPIC>	[1..1]	Identifier	N/A
ProprietaryAccount	<PrtryAcct>	[1..1]	N/A	N/A
Type	<Tp>	[0..1]	N/A	Post bank account's type ID
Currency	<Ccy>	[0..1]	Code	Post bank account's currency code
Name	<Nm>	[0..1]	Text	Post bank account's name

A.8 Transaction message export formats

CMM supports the following standard formats for transaction message exports:

Type	Wallstreet formats	EDIFACT formats	SWIFT formats	Other formats
Transaction message	• XML payment message • XML receipt message	• BANSTA [1]	N/A	• HTML [1]

Table notes:

1. For information on formats not documented in this appendix, contact Wallstreet.

A.8.1 Wallstreet XML payment message

The following is an example Wallstreet XML payment message file:

```
<?xml version="1.0" encoding="UTF-8"?>
<transactions xmlns="http://www.trema.com/externalinterface/XMLschema"
document_reference_id="12" format_code="TREMA_PAYMENT">
    <transaction sequence="1" type="payment">
       <debit>
          <debit_basics>
             <aggregate_transaction_id>15004</aggregate_transaction_id>
             <aggregate_transaction_amount>5848889.1</aggregate_transaction_amount>
             <transaction_currency>USD</transaction_currency>
             <value_date>2005-03-01</value_date>
          </debit_basics>
          <debit_bank_account>
             <account_number>004857658</account_number>
             <account_iban/>
             <account_clearing_reference_number/>
             <account_name>ABCsubUSD</account_name>
             <account_currency>USD</account_currency>
             <account_residency/>
          </debit_bank_account>
          <debit_bank>
             <swift_code>BIGBAK32</swift_code>
             <branch_number>10098</branch_number>
```

```
        <party_identification/>
        <party_name>Big Bank</party_name>
        <street_address>2008 Colling Street NW</street_address>
        <city>New York</city>
        <province>NY</province>
        <country_code>US</country_code>
        <postal_code/>
     </debit_bank>
     <debit_party>
        <party_identification/>
        <party_name>ABC Subsidiary (USD)</party_name>
        <street_address/>
        <city>Honolulu</city>
        <province>HI</province>
        <country_code>US</country_code>
       <postal_code>30000345</postal_code>
        <phone_number>1 418 256 7890</phone_number>
        <fax_number>1 418 257 4567</fax_number>
        <email_account>test@yahoo.com</email_account>
     </debit_party>
  </debit>
 <credit sequence="1">
    <credit_basics>
       <transaction_id>5005</transaction_id>
       <customer_reference_number>5005</customer_reference_number>
        <transaction_amount>5848389.09</transaction_amount>
        <transaction_currency>USD</transaction_currency>
        <value_date>2005-03-01</value_date>
        <transaction_date>2005-03-01</transaction_date>
        <transaction_priority_code>1</transaction_priority_code>
        <original_transaction_method>EFT</original_transaction_method>
        <transaction_delivery_channel>ACH</transaction_delivery_channel>
        <domestic_crossborder_status>DOMESTIC</domestic_crossborder_status>
        <clearing_system_type>LVC</clearing_system_type>
        <is_intercompany_payment>false</is_intercompany_payment>
        <repetitive_code/>
        <beneficiary_message/>
        <transaction_description>test again</transaction_description>
        <bank_instruction>test payment 2</bank_instruction>
        <original_system_code>Manual</original_system_code>
        <original_source_batch_id>0</original_source_batch_id>
        <financial_charge_allocation>2</financial_charge_allocation>
        <regulatory_code/>
     </credit_basics>
     <credit_bank_account>
        <account_number>Ext4_BOA_USD00</account_number>
        <account_iban/>
        <account_clearing_reference_number/>
        <account_name>Ext 4&amp;apos;s external acct in USD</account_name>
        <account_currency>USD</account_currency>
        <account_residency>RESIDENT</account_residency>
     </credit_bank_account>
     <credit_bank>
        <swift_code>BOAUSAUS</swift_code>
        <branch_number/>
        <party_identification/>
       <party_name>Bank of America in US</party_name>
        <street_address>236 Green Avenue</street_address>
        <city>Boston</city>
        <province/>
        <country_code>US</country_code>
        <postal_code>4949849</postal_code>
```

```
            </credit_bank>
            <intermediary_bank>
               <swift_code/>
               <branch_number/>
               <party_name/>
               <street_address/>
               <city/>
               <province/>
               <country_code/>
               <postal_code/>
            </intermediary_bank>
            <credit_party>
               <party_identification/>
               <party_name>External Party 4</party_name>
               <street_address>226 Clinton Park</street_address>
               <city>Houston</city>
               <province/>
               <country_code>US</country_code>
               <postal_code/>
               <phone_number/>
               <fax_number/>
               <email_account/>
            </credit_party>
            <originating_party>
               <party_identification/>
               <party_name>ABC Subsidiary (USD)</party_name>
               <street_address/>
               <city>Honolulu</city>
               <province>HI</province>
               <country_code>US</country_code>
               <postal_code>30000345</postal_code>
               <phone_number>1 418 256 7890</phone_number>
               <fax_number>1 418 257 4567</fax_number>
               <email_account>test@yahoo.com</email_account>
            </originating_party>
            <additional_info>
               <transaction_contract_number/>
               <transaction_type_code/>
               <cheque_number/>
               <retrieval_location_id/>
               <retrieval_location_name/>
               <retrieval_location_address/>
            </additional_info>
            <remittance_details/>
         </credit>
      </transaction>
</transactions>
```

This file contains 16 basic elements:

Element	Description
transactions	The file's identifier, format code, and schema.
transaction	An individual payment in the file.
debit	The debit component of a payment.
debit_basics	Basic information for the debit component.
debit_bank_account	Bank account information for the debit component.
debit_bank	Bank information for the debit component.

Element	Description
debit_party	Party information for the debit component.
credit	A credit component of the payment.
credit_basics	Basic information for the credit component.
credit_bank_account	Bank account information for the credit component.
credit_bank	Bank information for the credit component.
intermediary_bank	Intermediary bank information for the credit component.
credit_party	Party information for the credit component.
originating_party	Originating party information for the credit component.
additional_info	Additional information for the credit component.
remittance_details	Remittance details.

Note: The elements must be nested as displayed in the above table, and the file must validate against the Wallstreet XML payment message schema. For more information on the Wallstreet XML payment message schema, see A.11 Wallstreet XML schemas on page 292.

A.8.1.1 transactions element

A Wallstreet XML payment message file contains only one `transactions` element.

The `transactions` element defines the file's identifier, format code, and schema as specified in the element's three attributes:

Attribute	Acceptable values
xmlns	http://www.trema.com/externalinterface/XMLschema
document_reference_id	[A unique identifier for the file]
format_code	TREMA_PAYMENT

A.8.1.2 transaction elements

A Wallstreet XML payment message file can contain multiple payments with each payment represented by a `transaction` element.

Each `transaction` element defines its payment's sequence number and type as specified in the element's two attributes:

Attribute	Acceptable values
sequence	[A unique sequence number for the payment (In a typical file, the first payment is numbered 1; the second, 2; the third, 3; and so on.)]
type	payment

A.8.1.3 debit elements

Each `transaction` element contains one `debit` element. The `debit` element defines the debit component of a payment.

A.8.1.4 debit_basics elements

Each `debit` element contains one `debit_basics` element.

Each `debit_basics` element defines its debit component's basic information as specified in the element's child elements:

Element	Acceptable values
`aggregate_transaction_id`	[The debit component's ID]
`aggregate_transaction_ amount`	[The debit component's amount]
`transaction_currency`	[The debit component's currency]
`value_date`	[The debit component's value date]

A.8.1.5 debit_bank_account elements

Each `debit` element contains one `debit_bank_account` element.

Each `debit_bank_account` element defines its debit component's bank account information as specified in the element's child elements:

Element	Acceptable values
`account_number`	[The BBAN of the debit component's bank account]
`account_iban`	[The IBAN of the debit component's bank account]
`account_clearing_ reference_number`	[The clearing reference number of the debit component's bank account]
`account_name`	[The name of the debit component's bank account]
`account_currency`	[The currency of the debit component's bank account]
`account_residency`	[The residency of the debit component's bank account]

A.8.1.6 debit_bank elements

Each `debit` element contains one `debit_bank` element.

Each `debit_bank` element defines its debit component's bank information as specified in the element's child elements:

Element	Acceptable values
`swift_code`	[The SWIFT code of the debit component's bank]
`branch_number`	[The branch number of the debit component's bank]
`party_identification`	[The ID of the debit component's bank]
`party_name`	[The name of the debit component's bank]
`street_address`	[The address of the debit component's bank]
`city`	[The city of the debit component's bank]
`province`	[The state or province of the debit component's bank]
`country_code`	[The country of the debit component's bank]
`postal_code`	[The ZIP or postal code of the debit component's bank]

A.8.1.7 debit_party elements

Each `debit` element contains one `debit_party` element.

Each `debit_party` element defines its debit component's party information as specified in the element's child elements:

Element	Acceptable values
`party_identification`	[The ID of the debit component's party]
`party_name`	[The name of the debit component's party]
`street_address`	[The address of the debit component's party]
`city`	[The city of the debit component's party]
`province`	[The state or province of the debit component's party]
`country_code`	[The country of the debit component's party]
`postal_code`	[The ZIP or postal code of the debit component's party]
`phone_number`	[The telephone number of the debit component's party]
`fax_number`	[The fax number of the debit component's party]
`email_account`	[The e-mail address of the debit component's party]

A.8.1.8 credit elements

Each `transaction` element contains one or more `credit` elements. A `credit` element defines a credit component of a payment.

Each `credit` element defines its credit component's sequence number with the payment as specified in the element's attribute:

Attribute	Acceptable values
`sequence`	[A unique sequence number for the credit component (In a typical payment, the first credit component is numbered `1`; the second, `2`; the third, `3`; and so on.)]

A.8.1.9 credit_basics elements

Each `credit` element contains one `credit_basics` element.

Each `credit_basics` element defines its credit component's basic information as specified in the element's child elements:

Element	Acceptable values
`transaction_id`	[The credit component's ID]
`customer_reference_number`	[The credit component's customer reference number]
`transaction_amount`	[The credit component's amount]
`transaction_currency`	[The credit component's currency]
`value_date`	[The credit component's value date]
`transaction_date`	[The credit component's transaction date]
`transaction_priority_code`	[The credit component's priority]
`original_transaction_ method`	[The credit component's transaction method]
`transaction_delivery_ channel`	[The credit component's transaction delivery channel]

Element	Acceptable values
domestic_crossborder_ status	DOMESTIC CROSSBORDER
clearing_system_type	[The credit component's clearing system]
is_intercompany_payment	true false
repetitive_code	[The credit component's repetitive code]
beneficiary_message	[A message to the credit component's beneficiary]
transaction_description	[A description of the credit component]
bank_instruction	[A message to the credit component's bank]
original_system_code	[The credit component's original system code]
original_source_batch_id	[The credit component's original source batch ID]
financial_charge_ allocation	[The credit component's financial charge allocation]
regulatory_code	[The credit component's regulatory code]

A.8.1.10 credit_bank_account elements

Each credit element contains one credit_bank_account element.

Each credit_bank_account element defines its credit component's bank account information as specified in the element's child elements:

Element	Acceptable values
account_number	[The BBAN of the credit component's bank account]
account_iban	[The IBAN of the credit component's bank account]
account_clearing_ reference_number	[The clearing reference number of the credit component's bank account]
account_name	[The name of the credit component's bank account]
account_currency	[The currency of the credit component's bank account]
account_residency	[The residency of the credit component's bank account]

A.8.1.11 credit_bank elements

Each credit element contains one credit_bank element.

Each credit_bank element defines its credit component's bank information as specified in the element's child elements:

Element	Acceptable values
swift_code	[The SWIFT code of the credit component's bank]
branch_number	[The branch number of the credit component's bank]
party_identification	[The ID of the credit component's bank]
party_name	[The name of the credit component's bank]
street_address	[The address of the credit component's bank]

Element	Acceptable values
city	[The city of the credit component's bank]
province	[The state or province of the credit component's bank]
country_code	[The country of the credit component's bank]
postal_code	[The ZIP or postal code of the credit component's bank]

A.8.1.12 intermediary_bank elements

Each credit element contains one intermediary_bank element.

Each intermediary_bank element defines its credit component's intermediary bank information as specified in the element's child elements:

Element	Acceptable values
swift_code	[The SWIFT code of the credit component's intermediary bank]
branch_number	[The branch number of the credit component's intermediary bank]
party_name	[The name of the credit component's intermediary bank]
street_address	[The address of the credit component's intermediary bank]
city	[The city of the credit component's intermediary bank]
province	[The state or province of the credit component's intermediary bank]
country_code	[The country of the credit component's intermediary bank]
postal_code	[The ZIP or postal code of the credit component's intermediary bank]

A.8.1.13 credit_party elements

Each credit element contains one credit_party element.

Each credit_party element defines its credit component's party information as specified in the element's child elements:

Element	Acceptable values
party_identification	[The ID of the credit component's party]
party_name	[The name of the credit component's party]
street_address	[The address of the credit component's party]
city	[The city of the credit component's party]
province	[The state or province of the credit component's party]
country_code	[The country of the credit component's party]
postal_code	[The ZIP or postal code of the credit component's party]
phone_number	[The telephone number of the credit component's party]
fax_number	[The fax number of the credit component's party]
email_account	[The e-mail address of the credit component's party]

A.8.1.14 originating_party elements

Each credit element contains one originating_party element.

Each originating_party element defines its credit component's originating party information as specified in the element's child elements:

Element	Acceptable values
party_identification	[The ID of the credit component's originating party]
party_name	[The name of the credit component's originating party]
street_address	[The address of the credit component's originating party]
city	[The city of the credit component's originating party]
province	[The state or province of the credit component's originating party]
country_code	[The country of the credit component's originating party]
postal_code	[The ZIP or postal code of the credit component's originating party]
phone_number	[The telephone number of the credit component's originating party]
fax_number	[The fax number of the credit component's originating party]
email_account	[The e-mail address of the credit component's originating party]

A.8.1.15 additional_info elements

Each credit element contains one additional_info element.

Each additional_info element defines its credit component's additional information as specified in the element's child elements:

Element	Acceptable values
transaction_contract_ number	[The credit component's contract number]
transaction_type_code	[The credit component's type]
cheque_number	[The credit component's check number]
retrieval_location_id	[The ID of credit component's retrieval location]
retrieval_location_name	[The name of credit component's retrieval location]
retrieval_location_address	[The address of credit component's retrieval location]

A.8.1.16 remittance_details elements

Each credit element contains one remittance_details element. In turn, each remittance_details element can contain one or more invoice elements.

Each invoice element defines its remittance detail as specified in the element's child elements:

Element	Acceptable values
invoice_number	[The remittance detail's number]
invoice_type	[The remittance detail's type]
invoice_amount	[The remittance detail's amount]
invoice_amount_type	[The type of the remittance detail's amount]
invoice_date	[The remittance detail's invoice date]

Element	Acceptable values
invoice_currency	[The remittance detail's currency]
invoice_reference	[The remittance detail's reference]

A.8.2 Wallstreet XML receipt message

The following is an example Wallstreet XML receipt message file:

```
<?xml version="1.0" encoding="UTF-8"?>
<transactions xmlns="http://www.trema.com/externalinterface/XMLschema"
document_reference_id="15" format_code="TREMA_DIRDEB">
    <transaction sequence="1" type="direct_debit">
       <credit>
          <credit_basics>
             <aggregate_transaction_id>15011</aggregate_transaction_id>
             <aggregate_transaction_amount>112.99</aggregate_transaction_amount>
             <transaction_currency>FRF</transaction_currency>
             <value_date>2005-03-01</value_date>
          </credit_basics>
          <credit_bank_account>
             <account_number>3000400074500025628778</account_number>
             <account_iban/>
             <account_clearing_reference_number/>
             <account_name>ent1 account1</account_name>
             <account_currency>EUR</account_currency>
          </credit_bank_account>
          <credit_bank>
             <swift_code/>
             <branch_number>3000400074</branch_number>
             <party_name>Bank of America in France</party_name>
             <street_address>301 Bull Street</street_address>
             <city/>
             <province/>
             <country_code>FR</country_code>
             <postal_code/>
          </credit_bank>
          <credit_party>
             <party_identification/>
             <party_name>Entity 1</party_name>
             <street_address>408 Green Ave</street_address>
             <city>Paris</city>
             <province/>
             <country_code>FR</country_code>
             <postal_code/>
             <phone_number/>
             <fax_number/>
             <email_account/>
          </credit_party>
       </credit>
       <debit sequence="1">
          <debit_basics>
             <transaction_id>15011</transaction_id>

            <customer_reference_number>DD_EXT_ONE_OFF_123</customer_reference_numbe
            r>
             <transaction_amount>112.99</transaction_amount>
             <transaction_currency>FRF</transaction_currency>
             <value_date>2005-03-01</value_date>
             <transaction_date>2005-03-01</transaction_date>
```

```
    <transaction_priority_code>1</transaction_priority_code>
    <original_transaction_method>DD</original_transaction_method>
    <repetitive_code/>
    <beneficiary_message/>
    <transaction_description/>
    <bank_instruction/>
    <original_system_code>Man-OneOff</original_system_code>
    <original_source_batch_id>0</original_source_batch_id>
    <financial_charge_allocation>1</financial_charge_allocation>
    <regulatory_code/>
</debit_basics>
<debit_bank_account>
    <account_number>300040132890001089902 2</account_number>
    <account_iban/>
    <account_clearing_reference_number>60089
   </account_clearing_reference_number>
    <account_name/>
    <account_currency/>
</debit_bank_account>
<debit_bank>
    <swift_code/>
    <branch_number>3000401328</branch_number>
    <party_name>Counterparty Bank</party_name>
    <street_address/>
    <city/>
    <province/>
    <country_code>FR</country_code>
    <postal_code/>
</debit_bank>
<intermediary_bank>
    <swift_code/>
    <branch_number/>
    <party_name/>
    <street_address/>
    <city/>
    <province/>
    <country_code/>
    <postal_code/>
</intermediary_bank>
<debit_party>
    <party_identification/>
    <party_name>DD Counterparty</party_name>
    <street_address/>
   <city/>
    <province/>
    <country_code>FR</country_code>
    <postal_code/>
    <phone_number/>
    <fax_number/>
    <email_account/>
</debit_party>
<originating_party>
    <party_identification/>
    <party_name>Entity 1</party_name>
    <street_address>408 Green Ave</street_address>
    <city>Paris</city>
    <province/>
    <country_code>FR</country_code>
    <postal_code/>
    <phone_number/>
    <fax_number/>
    <email_account/>
```

```
            </originating_party>
            <additional_info>
               <transaction_contract_number>2223</transaction_contract_number>
               <transaction_type_code>87</transaction_type_code>
               <bank_pre_authorization_id/>
               <bank_pre_authorization_status/>
               <revocable_status/>
            </additional_info>
         </debit>
      </transaction>
</transactions>
```

This file contains 15 basic elements:

Element	Description
transactions	The file's identifier, format code, and schema.
transaction	An individual payment in the file.
credit	The credit component of a payment.
credit_basics	Basic information for the credit component.
credit_bank_account	Bank account information for the credit component.
credit_bank	Bank information for the credit component.
credit_party	Party information for the credit component.
debit	A debit component of the payment.
debit_basics	Basic information for the debit component.
debit_bank_account	Bank account information for the debit component.
debit_bank	Bank information for the debit component.
intermediary_bank	Intermediary bank information for the debit component.
debit_party	Party information for the debit component.
originating_party	Originating party information for the debit component.
additional_info	Additional information for the debit component.

Note: The elements must be nested as displayed in the above table, and the file must validate against the Wallstreet XML receipt message schema. For more information on the Wallstreet XML receipt message schema, see A.11 Wallstreet XML schemas on page 292.

A.8.2.1 transactions element

A Wallstreet XML receipt message file contains only one `transactions` element.

The `transactions` element defines the file's identifier, format code, and schema as specified in the element's three attributes:

Attribute	Acceptable values
xmlns	http://www.trema.com/externalinterface/XMLschema
document_reference_id	[A unique identifier for the file]

Attribute	Acceptable values
format_code	TREMA_DIRDEB TREMA_LETTER_OF_CREDIT TREMA_PREADVICE

A.8.2.2 transaction elements

A Wallstreet XML receipt message file can contain multiple receipts with each receipt represented by a transaction element.

Each transaction element defines its receipt's sequence number and type as specified in the element's two attributes:

Attribute	Acceptable values
sequence	[A unique sequence number for the receipt (In a typical file, the first receipt is numbered 1; the second, 2; the third, 3; and so on.)]
type	direct_debit letter_of_credit pre_advice

A.8.2.3 credit elements

Each transaction element contains one credit element. The credit element defines the credit component of a receipt.

A.8.2.4 credit_basics elements

Each credit element contains one credit_basics element.

Each credit_basics element defines its credit component's basic information as specified in the element's child elements:

Element	Acceptable values
aggregate_transaction_id	[The credit component's ID]
aggregate_transaction_ amount	[The credit component's amount]
transaction_currency	[The credit component's currency]
value_date	[The credit component's value date]

A.8.2.5 credit_bank_account elements

Each credit element contains one credit_bank_account element.

Each credit_bank_account element defines its credit component's bank account information as specified in the element's child elements:

Element	Acceptable values
account_number	[The BBAN of the credit component's bank account]
account_iban	[The IBAN of the credit component's bank account]
account_clearing_ reference_number	[The clearing reference number of the credit component's bank account]
account_name	[The name of the credit component's bank account]

Element	Acceptable values
account_currency	[The currency of the credit component's bank account]

A.8.2.6 credit_bank elements

Each credit element contains one credit_bank element.

Each credit_bank element defines its credit component's bank information as specified in the element's child elements:

Element	Acceptable values
swift_code	[The SWIFT code of the credit component's bank]
branch_number	[The branch number of the credit component's bank]
party_name	[The name of the credit component's bank]
street_address	[The address of the credit component's bank]
city	[The city of the credit component's bank]
province	[The state or province of the credit component's bank]
country_code	[The country of the credit component's bank]
postal_code	[The ZIP or postal code of the credit component's bank]

A.8.2.7 credit_party elements

Each credit element contains one credit_party element.

Each credit_party element defines its credit component's party information as specified in the element's child elements:

Element	Acceptable values
party_identification	[The ID of the credit component's party]
party_name	[The name of the credit component's party]
street_address	[The address of the credit component's party]
city	[The city of the credit component's party]
province	[The state or province of the credit component's party]
country_code	[The country of the credit component's party]
postal_code	[The ZIP or postal code of the credit component's party]
phone_number	[The telephone number of the credit component's party]
fax_number	[The fax number of the credit component's party]
email_account	[The e-mail address of the credit component's party]

A.8.2.8 debit elements

Each transaction element contains one or more debit elements. A debit element defines a debit component of a receipt.

Each debit element defines its debit component's sequence number with the receipt as specified in the element's attribute:

Attribute	Acceptable values
sequence	[A unique sequence number for the debit component (In a typical receipt, the first debit component is numbered 1; the second, 2; the third, 3; and so on.)]

A.8.2.9 debit_basics elements

Each debit element contains one debit_basics element.

Each debit_basics element defines its debit component's basic information as specified in the element's child elements:

Element	Acceptable values
transaction_id	[The debit component's ID]
customer_reference_number	[The debit component's customer reference number]
transaction_amount	[The debit component's amount]
transaction_currency	[The debit component's currency]
value_date	[The debit component's value date]
transaction_date	[The debit component's transaction date]
transaction_priority_code	[The debit component's priority]
original_transaction_method	[The debit component's transaction method]
repetitive_code	[The debit component's repetitive code]
beneficiary_message	[A message to the debit component's beneficiary]
transaction_description	[A description of the debit component]
bank_instruction	[A message to the debit component's bank]
original_system_code	[The debit component's original system code]
original_source_batch_id	[The debit component's original source batch ID]
financial_charge_ allocation	[The debit component's financial charge allocation]
regulatory_code	[The debit component's regulatory code]

A.8.2.10 debit_bank_account elements

Each debit element contains one debit_bank_account element.

Each debit_bank_account element defines its debit component's bank account information as specified in the element's child elements:

Element	Acceptable values
account_number	[The BBAN of the debit component's bank account]
account_iban	[The IBAN of the debit component's bank account]
account_clearing_ reference_number	[The clearing reference number of the debit component's bank account]
account_name	[The name of the debit component's bank account]
account_currency	[The currency of the debit component's bank account]

A.8.2.11 debit_bank elements

Each `debit` element contains one `debit_bank` element.

Each `debit_bank` element defines its debit component's bank information as specified in the element's child elements:

Element	Acceptable values
`swift_code`	[The SWIFT code of the debit component's bank]
`branch_number`	[The branch number of the debit component's bank]
`party_name`	[The name of the debit component's bank]
`street_address`	[The address of the debit component's bank]
`city`	[The city of the debit component's bank]
`province`	[The state or province of the debit component's bank]
`country_code`	[The country of the debit component's bank]
`postal_code`	[The ZIP or postal code of the debit component's bank]

A.8.2.12 intermediary_bank elements

Each `debit` element contains one `intermediary_bank` element.

Each `intermediary_bank` element defines its debit component's intermediary bank information as specified in the element's child elements:

Element	Acceptable values
`swift_code`	[The SWIFT code of the debit component's intermediary bank]
`branch_number`	[The branch number of the debit component's intermediary bank]
`party_name`	[The name of the debit component's intermediary bank]
`street_address`	[The address of the debit component's intermediary bank]
`city`	[The city of the debit component's intermediary bank]
`province`	[The state or province of the debit component's intermediary bank]
`country_code`	[The country of the debit component's intermediary bank]
`postal_code`	[The ZIP or postal code of the debit component's intermediary bank]

A.8.2.13 debit_party elements

Each `debit` element contains one `debit_party` element.

Each `debit_party` element defines its debit component's party information as specified in the element's child elements:

Element	Acceptable values
`party_identification`	[The ID of the debit component's party]
`party_name`	[The name of the debit component's party]
`street_address`	[The address of the debit component's party]
`city`	[The city of the debit component's party]
`province`	[The state or province of the debit component's party]

Element	Acceptable values
country_code	[The country of the debit component's party]
postal_code	[The ZIP or postal code of the debit component's party]
phone_number	[The telephone number of the debit component's party]
fax_number	[The fax number of the debit component's party]
email_account	[The e-mail address of the debit component's party]

A.8.2.14 originating_party elements

Each debit element contains one originating_party element.

Each orginating_party element defines its debit component's originating party information as specified in the element's child elements:

Element	Acceptable values
party_identification	[The ID of the debit component's originating party]
party_name	[The name of the debit component's originating party]
street_address	[The address of the debit component's originating party]
city	[The city of the debit component's originating party]
province	[The state or province of the debit component's originating party]
country_code	[The country of the debit component's originating party]
postal_code	[The ZIP or postal code of the debit component's originating party]
phone_number	[The telephone number of the debit component's originating party]
fax_number	[The fax number of the debit component's originating party]
email_account	[The e-mail address of the debit component's originating party]

A.8.2.15 additional_info elements

Each debit element contains one additional_info element.

Each additional_info element defines its debit component's additional information as specified in the element's child elements:

Element	Acceptable values
transaction_contract_number	[The debit component's contract number]
transaction_type_code	[The debit component's type]
bank_pre_authorization_id	[The debit component's bank preauthorization ID]
bank_pre_authorization_ status	[The debit component's bank preauthorization status]
revocable_status	[The debit component's revocable status]

A.9 General ledger posting export formats

CMM supports the following standard formats for general ledger posting exports:

Type	Wallstreet formats	EDIFACT formats	SWIFT formats	Other formats
Deal	• XML general ledger posting	N/A	N/A	N/A

A.9.1 Wallstreet XML general ledger posting

The following is an example Wallstreet XML general ledger posting file:

```
<transactions xmlns="http://www.trema.com/externalinterface/XMLschema"
format_code="TREMA_COMPANY_GL" document_reference_id="6">
    <transaction sequence="1">
       <accounting_period_number>1</accounting_period_number>
       <gl_group_id>123667</gl_group_id>
       <gl_account_code>Code1</gl_account_code>
       <txn_direction>C</txn_direction>
       <txn_date>2006-02-07</txn_date>
       <value_date>2006-07-24</value_date>
       <txn_amount>73732.08</txn_amount>
       <txn_currency>USD</txn_currency>
       <base_currency_amount>764376.00</base_currency_amount>
       <base_currency>CAD</base_currency>
       <description/>
       <customer_reference_id>cus-ref-2</customer_reference_id>
       <originating_system_code>Manual</originating_system_code>
       <party_id>ABCco</party_id>
       <bank_account_number>Ent1_IHB_EUR</bank_account_number>
       <bank_account_name>Ent1 IHB EUR 1</bank_account_name>
       <cpty_id>Ext1</cpty_id>
       <instrument_category>COMM</instrument_category>
       <instrument_type_code>COMMP</instrument_type_code>
       <treasure_gl_id>0</treasure_gl_id>
       <intercompany_txn>0</intercompany_txn>
       <set_of_books_name/>
       <cash_deal>C</cash_deal>
       <source_reference_id>3</source_reference_id>
    </transaction>
    <transaction sequence="2">
       <accounting_period_number>2</accounting_period_number>
       <gl_group_id>123554</gl_group_id>
       <gl_account_code>Code2</gl_account_code>
       <txn_direction>D</txn_direction>
       <txn_date>2006-02-07</txn_date>
       <value_date>2006-07-24</value_date>
       <txn_amount>832873.99</txn_amount>
       <txn_currency>USD</txn_currency>
       <base_currency_amount>6535.00</base_currency_amount>
       <base_currency>CAD</base_currency>
       <description/>
       <customer_reference_id>cus ref-1</customer_reference_id>
       <originating_system_code>Manual</originating_system_code>
       <party_id>ABCco</party_id>
       <bank_account_number>Ent1_IHB_EUR</bank_account_number>
       <bank_account_name>Ent1 IHB EUR 1</bank_account_name>
       <cpty_id>Ext2</cpty_id>
       <instrument_category>COMM</instrument_category>
       <instrument_type_code>COMMFEE</instrument_type_code>
```

```
      <treasure_gl_id>0</treasure_gl_id>
       <intercompany_txn>0</intercompany_txn>
       <set_of_books_name/>
       <cash_deal>C</cash_deal>
       <source_reference_id>1</source_reference_id>
    </transaction>
</transactions>
```

This file contains two basic elements:

Element	Description
transactions	The file's identifier, format code, and schema.
transaction	An individual general ledger posting in the file.

Note: The elements must be nested as displayed in the above table, and the file must validate against the Wallstreet XML general ledger posting schema. For more information on the Wallstreet XML general ledger posting schema, see A.11 Wallstreet XML schemas on page 292.

A.9.1.1 transactions element

A Wallstreet XML general ledger posting file contains only one `transactions` element.

The `transactions` element defines the file's identifier, format code, and schema as specified in the element's three attributes:

Attribute	Acceptable values
xmlns	http://www.trema.com/externalinterface/XMLschema
format_code	TREMA_COMPANY_GL
document_reference_id	[A unique identifier for the file]

A.9.1.2 transaction elements

A Wallstreet XML general ledger posting file can contain multiple general ledger postings with each general ledger posting represented by a `transaction` element.

Each `transaction` element defines its general ledger posting's sequence number as specified in the element's attribute:

Attribute	Acceptable values
sequence	[A unique sequence number for the general ledger posting (In a typical file, the first general ledger posting is numbered 1; the second, 2; the third, 3; and so on.)]

In addition, each `transaction` element defines its general ledger posting's details as specified in the element's child elements:

Element	Acceptable values
accounting_period_number	[The general ledger posting's accounting period]
gl_group_id	[The general ledger posting's group ID]
gl_account_code	[The general ledger posting's account code]
txn_direction	C for credit D for debit

Element	Acceptable values
txn_date	[The general ledger posting's transaction date]
value_date	[The general ledger posting's value date]
txn_amount	[The general ledger posting's amount in the transaction currency]
txn_currency	[The general ledger posting's transaction currency]
base_currency_amount	[The general ledger posting's amount in the base currency]
base_currency	[The general ledger posting's base currency]
description	[A description of the general ledger posting]
customer_reference_id	[The general ledger posting's customer reference number]
originating_system_code	[The general ledger posting's originating system]
party_id	[The general ledger posting's party]
bank_account_number	[The number of the general ledger posting's party bank account]
bank_account_name	[The name of the general ledger posting's party bank account]
cpty_id	[The general ledger posting's counterparty]
instrument_category	[The general ledger posting's instrument category]
instrument_type_code	[The general ledger posting's instrument type]
treasure_gl_id	[The general ledger posting's treasury general ledger ID]
intercompany_txn	0 for external 1 for intercompany
set_of_books_name	[The general ledger posting's set of books]
cash_deal	C for cash D for deal
source_reference_id	[The general ledger posting's source reference ID]

A.10 Free formats

CMM supports the following free formats:

Type	Wallstreet formats	EDIFACT formats	SWIFT formats	Other formats
Free format	N/A	N/A	• MT199 • MT999	N/A

A.10.1 SWIFT MT199 and MT999

The SWIFT MT199 and MT999 messages are sent by a financial institution or a corporation to deliver information for which there is no other applicable SWIFT message type.

Note: The SWIFT MT199 and MT999 messages can be used in place of the SWIFT MT195 and MT196 messages.

A.10.1.1 SWIFT MT199 and MT999 header block

The following table presents the contents of the SWIFT MT199 and MT999 header block:

Tag	CMM mapping	Example
Sender	Interchange sender ID	`PLATUS33`
Message type	Static value `199` or `999`	`199`
Receiver	Interchange receiver ID	`AAAAUS33`

A.10.1.2 SWIFT MT199 and MT999 main message block

The following table presents the available tags in the main message blocks of SWIFT MT199 and MT999 messages:

Tag	Name	Required	Contents [1]	CMM mapping	Example
20	Transaction reference number	M	16x	Message ID	`:20:497`
21	Related reference	O	16x	Related reference	`:21:29472394`
79	Narrative	M	35*50x	Message text	`:79:The referenced payment has an incorrect bank account and should be rejected. We will send a revised payment later today.`

Table notes:

1. For more information on the codes used in this column, see A.13.1 SWIFT field lengths on page 303 and A.13.2 SWIFT field types on page 304.

A.11 Wallstreet XML schemas

Each of the Wallstreet XML formats must validate against one of the following six schemas:

- Transaction
- Transaction acknowledgement
- Bank statement
- Payment message
- Receipt message
- General ledger posting.

A.11.1 Transaction schema

The Wallstreet XML transaction schema is defined by the `tremaattributestransaction.xsd` file. The following are the contents of this file:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.trema.com/externalinterface/XMLschema"
elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.trema.com/externalinterface/XMLschema">
    <xsd:annotation>
       <xsd:documentation xml:lang="en">
          This is the standard Trema XML attribute transactions schema definition.
         This schema can be used for any type of transactions imported to CMM
         including AP, AR, DD, Bank Balance and transaction, bank message as long as
         the supported transaction attributes are provided. Copyright 2005
         trema.com. All rights reserved.
       </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="transactions">
       <xsd:complexType>
          <xsd:sequence>
             <xsd:element name="transaction" maxOccurs="unbounded"
           type="transaction_definition"/>
          </xsd:sequence>
          <xsd:attribute name="document_reference_id" use="required"
         type="xsd:string"/>
          <xsd:attribute name="format_code" use="required" fixed="ATTRS_TXN"/>
       </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="transaction_definition">
       <xsd:sequence>
          <xsd:element name="attribute" maxOccurs="unbounded"
         type="attribute_definition"/>
       </xsd:sequence>
       <xsd:attribute name="sequence" use="optional" type="xsd:positiveInteger"/>
       <xsd:attribute name="error" use="optional" type="xsd:string"/>
       <xsd:attribute name="type" use="required" type="transaction_types"/>
    </xsd:complexType>
    <xsd:complexType name="attribute_definition">
       <xsd:sequence>
          <xsd:element name="component_transaction" minOccurs="0"
         maxOccurs="unbounded" type="transaction_definition"/>
       </xsd:sequence>
       <xsd:attribute name="name" use="required" type="xsd:string"/>
       <xsd:attribute name="value" use="optional" default="" type="xsd:string"/>
       <xsd:attribute name="composite" use="optional" type="xsd:boolean"/>
    </xsd:complexType>
    <xsd:simpleType name="transaction_types">
       <xsd:restriction base="xsd:string">
          <xsd:enumeration value="aprecord"/>
          <xsd:enumeration value="arrecord"/>
          <xsd:enumeration value="ddrecord"/>
          <xsd:enumeration value="remittancedetail"/>
          <xsd:enumeration value="bankmessage"/>
          <xsd:enumeration value="bankbalance"/>
          <xsd:enumeration value="banktxn"/>
          <xsd:enumeration value="forecast"/>
       </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```

A.11.2 Transaction acknowledgement schema

The Wallstreet XML transaction acknowledgement schema is defined by the `trematransactionacknowledgement.xsd` file. The following are the contents of this file:

```
<?xml version="1.0" encoding="UTF-8"?>
°
<xsd:schema targetNamespace="http://www.trema.com/externalinterface/XMLschema"
elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.trema.com/externalinterface/XMLschema">
   <xsd:annotation>
       <xsd:documentation xml:lang="en">
          This is the standard Trema XML transaction acknowledgement message schema
         definition Copyright 2004 trema.com. All rights reserved.
       </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="transactions">
       <xsd:complexType>
          <xsd:sequence>
             <xsd:element name="transaction" maxOccurs="unbounded"
           type="transaction_definition"/>
          </xsd:sequence>
          …
          <xsd:attribute name="document_reference_id" use="required"
         type="xsd:string"/>
          <xsd:attribute name="format_code" use="required" fixed="TREMA_ACK"/>
       </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="transaction_definition">
       <xsd:sequence>
          <xsd:element name="transaction_id" type="xsd:positiveInteger"
         minOccurs="0"/>
          <xsd:element name="customer_reference_number" type="xsd:string"/>
          <xsd:element name="external_reference_number" type="xsd:string"
         minOccurs="0"/>
          <xsd:element name="transaction_success" type="xsd:boolean"/>
          <xsd:element name="rejection_reason" type="xsd:string" minOccurs="0"/>
       </xsd:sequence>
       <xsd:attribute name="sequence" use="required" type="xsd:positiveInteger"/>
       <xsd:attribute name="type" use="required" fixed="acknowledgement"/>
    </xsd:complexType>
</xsd:schema>
```

A.11.3 Bank statement schema

The Wallstreet XML bank statement schema is defined by the `tremabankaccountstatement.xsd` file. The following are the contents of this file:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.trema.com/externalinterface/XMLschema"
elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.trema.com/externalinterface/XMLschema">
    <xsd:annotation>
      <xsd:documentation xml:lang="en">
          This is the standard Trema XML bank account statement (balance and
         transactions) schema definition Copyright 2004 trema.com. All rights
         reserved.
       </xsd:documentation>
    </xsd:annotation>
    <xsd:element name="bankaccountstatementholder">
```

```
    <xsd:complexType>
       <xsd:sequence>
          <xsd:element name="account_statement" maxOccurs="unbounded"
         type="statement_definition"/>
       </xsd:sequence>
       <xsd:attribute name="document_reference_id" use="required"
      type="xsd:string"/>
       <xsd:attribute name="format_code" use="required"
      fixed="TREMA_BANK_STATEMENT"/>
       <xsd:attribute name="statement_type" use="required"
      type="statement_type_definition"/>
    </xsd:complexType>
 </xsd:element>
 <xsd:simpleType name="statement_type_definition">
    <xsd:restriction base="xsd:string">
       <xsd:enumeration value="P"/>
       <xsd:enumeration value="C"/>
    </xsd:restriction>
 </xsd:simpleType>
 <xsd:complexType name="statement_definition">
    <xsd:sequence>
       <xsd:element name="account" type="account_definition"/>
       <xsd:element name="balance" type="balance_definition" minOccurs="0"
      maxOccurs="unbounded"/>
       <xsd:element name="transaction" type="transaction_definition"
      minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="sequence" use="required" type="xsd:positiveInteger"/>
    <xsd:attribute name="statement_number" use="optional" type="xsd:string"/>
 </xsd:complexType>
 <xsd:complexType name="account_definition">
    <xsd:sequence>
       <xsd:element name="bank_swift_code" type="xsd:string" minOccurs="0"/>
       <xsd:element name="bank_branch_number" type="xsd:string" minOccurs="0"/>
       <xsd:element name="account_number" type="xsd:string" minOccurs="0"/>
       <xsd:element name="account_iban" type="xsd:string" minOccurs="0"/>
       <xsd:element name="account_currency" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
 </xsd:complexType>
<xsd:complexType name="balance_definition">
    <xsd:sequence>
       <xsd:element name="date" type="xsd:date"/>
       <xsd:element name="amount" type="xsd:decimal"/>
    </xsd:sequence>
    <xsd:attribute name="type" use="required" type="balance_type"/>
 </xsd:complexType>
 <xsd:simpleType name="balance_type">
    <xsd:restriction base="xsd:positiveInteger">
       <xsd:enumeration value="3"/>
       <xsd:enumeration value="6"/>
       <xsd:enumeration value="8"/>
       <xsd:enumeration value="19"/>
       <xsd:enumeration value="1001"/>
       <xsd:enumeration value="1002"/>
    </xsd:restriction>
 </xsd:simpleType>
 <xsd:complexType name="transaction_definition">
    <xsd:sequence>
       <xsd:element name="customer_reference_number" type="xsd:string"/>
       <xsd:element name="bank_reference_number" type="xsd:string"
      minOccurs="0"/>
       <xsd:element name="check_number" type="xsd:string" minOccurs="0"/>
```

```
            <xsd:element name="other_reference_number" type="xsd:string"
           minOccurs="0"/>
            <xsd:element name="book_date" type="xsd:date"/>
            <xsd:element name="value_date" type="xsd:date"/>
            <xsd:element name="amount" type="xsd:decimal"/>
            <xsd:element name="description" type="xsd:string" minOccurs="0"/>
         </xsd:sequence>
         <xsd:attribute name="type" use="optional" type="transaction_type"/>
      </xsd:complexType>
      <xsd:simpleType name="transaction_type">
         <xsd:restriction base="xsd:string">
            <xsd:enumeration value="COMMP"/>
            <xsd:enumeration value="COMMR"/>
            <xsd:enumeration value="BANKFEE"/>
            <xsd:enumeration value="ZBA"/>
            <xsd:enumeration value="RCNADJ"/>
            <xsd:enumeration value="TAX"/>
            <xsd:enumeration value="BANKINT"/>
            <xsd:enumeration value="POOL"/>
            <xsd:enumeration value="COMMFEE"/>
            <xsd:enumeration value="INTER_ACCT"/>
            <xsd:enumeration value="OVDRFEE"/>
           <xsd:enumeration value="AGGRP"/>
         </xsd:restriction>
      </xsd:simpleType>
</xsd:schema>
```

A.11.4 Payment message schema

The Wallstreet XML payment message schema is defined by the `tremapayment.xsd` file. The following are the contents of this file:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.trema.com/externalinterface/XMLschema"
elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.trema.com/externalinterface/XMLschema">
      <xsd:annotation>
         <xsd:documentation xml:lang="en">
            This is the standard Trema XML payment message schema definition Copyright
           2004 trema.com. All rights reserved.
         </xsd:documentation>
      </xsd:annotation>
      <xsd:element name="transactions">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="transaction" maxOccurs="unbounded"
              type="transaction_definition"/>
            </xsd:sequence>
            <xsd:attribute name="document_reference_id" use="required"
           type="xsd:string"/>
            <xsd:attribute name="format_code" use="required" fixed="TREMA_PAYMENT"/>
         </xsd:complexType>
      </xsd:element>
      <xsd:complexType name="transaction_definition">
         <xsd:sequence>
            <xsd:element name="debit" type="debit_definition"/>
            <xsd:element name="credit" maxOccurs="unbounded"
           type="credit_definition"/>
         </xsd:sequence>
         <xsd:attribute name="sequence" use="required" type="xsd:positiveInteger"/>
```

```
   <xsd:attribute name="type" use="required" fixed="payment"/>
</xsd:complexType>
<xsd:complexType name="debit_definition">
   <xsd:sequence>
      <xsd:element name="debit_basics" type="debit_basics_definition"/>
      <xsd:element name="debit_bank_account" type="bank_account_definition"/>
      <xsd:element name="debit_bank" type="bank_definition"/>
      <xsd:element name="debit_party" type="party_definition"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="credit_definition">
   <xsd:sequence>
      <xsd:element name="credit_basics" type="credit_basics_definition"/>
      <xsd:element name="credit_bank_account" type="bank_account_definition"
     minOccurs="0"/>
      <xsd:element name="credit_bank" type="bank_definition" minOccurs="0"/>
      <xsd:element name="intermediary_bank" type="bank_definition"
     minOccurs="0"/>
      <xsd:element name="credit_party" type="party_definition"/>
      <xsd:element name="originating_party" type="party_definition"/>
      <xsd:element name="additional_info" minOccurs="0"
     type="additional_info_definition"/>
      <xsd:element name="remittance_details" minOccurs="0"
     type="remittance_details_definition"/>
   </xsd:sequence>
   <xsd:attribute name="sequence" use="required" type="xsd:positiveInteger"/>
</xsd:complexType>
<xsd:complexType name="debit_basics_definition">
   <xsd:sequence>
      <xsd:element name="aggregate_transaction_id" type="xsd:positiveInteger"/>
      <xsd:element name="aggregate_transaction_amount" type="xsd:decimal"/>
      <xsd:element name="transaction_currency" type="xsd:string"/>
      <xsd:element name="value_date" type="xsd:date"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="credit_basics_definition">
   <xsd:sequence>
      <xsd:element name="transaction_id" type="xsd:positiveInteger"/>
      <xsd:element name="customer_reference_number" type="xsd:string"/>
      <xsd:element name="transaction_amount" type="xsd:decimal"/>
      <xsd:element name="transaction_currency" type="xsd:string"/>
      <xsd:element name="value_date" type="xsd:date"/>
      <xsd:element name="transaction_date" type="xsd:date"/>
      <xsd:element name="transaction_priority_code"
     type="xsd:positiveInteger"/>
      <xsd:element name="original_transaction_method" type="xsd:string"/>
      <xsd:element name="transaction_delivery_channel" type="xsd:string"/>
      <xsd:element name="domestic_crossborder_status" type="xsd:string"/>
      <xsd:element name="clearing_system_type" type="xsd:string"/>
      <xsd:element name="is_intercompany_payment" type="xsd:boolean"/>
      <xsd:element name="repetitive_code" type="xsd:string"/>
      <xsd:element name="beneficiary_message" type="xsd:string"/>
      <xsd:element name="transaction_description" type="xsd:string"/>
      <xsd:element name="bank_instruction" type="xsd:string"/>
      <xsd:element name="original_system_code" type="xsd:string"/>
      <xsd:element name="original_source_batch_id"
     type="xsd:nonNegativeInteger"/>
      <xsd:element name="financial_charge_allocation"
     type="xsd:positiveInteger"/>
      <xsd:element name="regulatory_code" type="xsd:string"/>
   </xsd:sequence>
</xsd:complexType>
```

```
<xsd:complexType name="bank_account_definition">
   <xsd:sequence>
      <xsd:element name="account_number" type="xsd:string"/>
      <xsd:element name="account_iban" type="xsd:string"/>
      <xsd:element name="account_clearing_reference_number" type="xsd:string"/>
      <xsd:element name="account_name" type="xsd:string"/>
      <xsd:element name="account_currency" type="xsd:string"/>
      <xsd:element name="account_residency" type="xsd:string"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="bank_definition">
   <xsd:sequence>
      <xsd:element name="swift_code" type="xsd:string"/>
      <xsd:element name="branch_number" type="xsd:string"/>
      <xsd:element name="party_identification" type="xsd:string" minOccurs="0"/>
      <xsd:element name="party_name" type="xsd:string"/>
      <xsd:element name="street_address" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
      <xsd:element name="province" type="xsd:string"/>
      <xsd:element name="country_code" type="xsd:string"/>
      <xsd:element name="postal_code" type="xsd:string"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="party_definition">
   <xsd:sequence>
      <xsd:element name="party_identification" type="xsd:string"/>
      <xsd:element name="party_name" type="xsd:string"/>
      <xsd:element name="street_address" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
      <xsd:element name="province" type="xsd:string"/>
      <xsd:element name="country_code" type="xsd:string"/>
      <xsd:element name="postal_code" type="xsd:string"/>
      <xsd:element name="phone_number" type="xsd:string"/>
      <xsd:element name="fax_number" type="xsd:string"/>
      <xsd:element name="email_account" type="xsd:string"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="additional_info_definition">
   <xsd:sequence>
      <xsd:element name="transaction_contract_number" type="xsd:string"
     minOccurs="0"/>
      <xsd:element name="transaction_type_code" type="xsd:string"
     minOccurs="0"/>
      <xsd:element name="cheque_number" type="xsd:string" minOccurs="0"/>
      <xsd:element name="retrieval_location_id" type="xsd:string"
     minOccurs="0"/>
      <xsd:element name="retrieval_location_name" type="xsd:string"
     minOccurs="0"/>
      <xsd:element name="retrieval_location_address" type="xsd:string"
     minOccurs="0"/>
      <xsd:any minOccurs="0" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="remittance_details_definition">
   <xsd:sequence>
      <xsd:element name="invoice" minOccurs="0" maxOccurs="unbounded"
     type="invoice_definition"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="invoice_definition">
   <xsd:sequence>
      <xsd:element name="invoice_number" type="xsd:string"/>
```

```
            <xsd:element name="invoice_type" type="xsd:string"/>
            <xsd:element name="invoice_amount" type="xsd:decimal"/>
            <xsd:element name="invoice_amount_type" type="xsd:positiveInteger"/>
            <xsd:element name="invoice_date" type="xsd:date"/>
            <xsd:element name="invoice_currency" type="xsd:string"/>
            <xsd:element name="invoice_reference" type="xsd:string"/>
        </xsd:sequence>
     </xsd:complexType>
</xsd:schema>
```

A.11.5 Receipt message schema

The Wallstreet XML receipt message schema is defined by the `tremareceipt.xsd` file. The following are the contents of this file:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.trema.com/externalinterface/XMLschema"
elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.trema.com/externalinterface/XMLschema">
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
         This is the standard Trema XML receipt message (Direct Debit, Letter of
         Credit and Pre advice) schema definition Copyright 2004 trema.com. All
         rights reserved.
      </xsd:documentation>
   </<xsd:annotation>
   <xsd:element name="transactions">
       <xsd:complexType>
          <xsd:sequence>
             <xsd:element name="transaction" maxOccurs="unbounded"
            type="transaction_definition"/>
          </xsd:sequence>
          <xsd:attribute name="document_reference_id" use="required"
         type="xsd:string"/>
          <xsd:attribute name="format_code" use="required">
             <xsd:simpleType>
                <xsd:restriction base="xsd:string">
                   <xsd:enumeration value="TREMA_DIRDEB"/>
                   <xsd:enumeration value="TREMA_LETTER_OF_CREDIT"/>
                   <xsd:enumeration value="TREMA_PREADVICE"/>
                </xsd:restriction>
             </xsd:simpleType>
          </xsd:attribute>
       </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="transaction_definition">
       <xsd:sequence>
          <xsd:element name="credit" type="credit_definition"/>
          <xsd:element name="debit" type="debit_definition" maxOccurs="unbounded"/>
       </xsd:sequence>
       <xsd:attribute name="sequence" use="required" type="xsd:positiveInteger"/>
       <xsd:attribute name="type" use="required">
          <xsd:simpleType>
             <xsd:restriction base="xsd:string">
                <xsd:enumeration value="direct_debit"/>
                <xsd:enumeration value="letter_of_credit"/>
                <xsd:enumeration value="pre_advice"/>
             </xsd:restriction>
          </xsd:simpleType>
       </xsd:attribute>
```

```
 </xsd:complexType>
 <xsd:complexType name="credit_definition">
    <xsd:sequence>
       <xsd:element name="credit_basics" type="credit_basics_definition"/>
       <xsd:element name="credit_bank_account" type="bank_account_definition"
      minOccurs="0"/>
       <xsd:element name="credit_bank" type="bank_definition" minOccurs="0"/>
       <xsd:element name="credit_party" type="party_definition"/>
    </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="debit_definition">
    <xsd:sequence>
       <xsd:element name="debit_basics" type="debit_basics_definition"/>
       <xsd:element name="debit_bank_account" type="bank_account_definition"/>
       <xsd:element name="debit_bank" type="bank_definition"/>
       <xsd:element name="intermediary_bank" type="bank_definition"
      minOccurs="0"/>
       <xsd:element name="debit_party" type="party_definition"/>
       <xsd:element name="originating_party" type="party_definition"/>
       <xsd:element name="additional_info" minOccurs="0"
      type="additional_info_definition"/>
    </xsd:sequence>
    <xsd:attribute name="sequence" use="required" type="xsd:positiveInteger"/>
</xsd:complexType>
 <xsd:complexType name="credit_basics_definition">
    <xsd:sequence>
       <xsd:element name="aggregate_transaction_id" type="xsd:positiveInteger"/>
       <xsd:element name="aggregate_transaction_amount" type="xsd:decimal"/>
       <xsd:element name="transaction_currency" type="xsd:string"/>
       <xsd:element name="value_date" type="xsd:date"/>
    </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="debit_basics_definition">
    <xsd:sequence>
       <xsd:element name="transaction_id" type="xsd:positiveInteger"/>
       <xsd:element name="customer_reference_number" type="xsd:string"/>
       <xsd:element name="transaction_amount" type="xsd:decimal"/>
       <xsd:element name="transaction_currency" type="xsd:string"/>
       <xsd:element name="value_date" type="xsd:date"/>
       <xsd:element name="transaction_date" type="xsd:date"/>
       <xsd:element name="transaction_priority_code"
      type="xsd:positiveInteger"/>
       <xsd:element name="original_transaction_method" type="xsd:string"/>
       <xsd:element name="repetitive_code" type="xsd:string"/>
       <xsd:element name="beneficiary_message" type="xsd:string"/>
       <xsd:element name="transaction_description" type="xsd:string"/>
       <xsd:element name="bank_instruction" type="xsd:string"/>
       <xsd:element name="original_system_code" type="xsd:string"/>
       <xsd:element name="original_source_batch_id"
      type="xsd:nonNegativeInteger"/>
       <xsd:element name="financial_charge_allocation"
      type="xsd:positiveInteger"/>
       <xsd:element name="regulatory_code" type="xsd:string"/>
    </xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="bank_account_definition">
    <xsd:sequence>
       <xsd:element name="account_number" type="xsd:string"/>
       <xsd:element name="account_iban" type="xsd:string"/>
       <xsd:element name="account_clearing_reference_number" type="xsd:string"/>
       <xsd:element name="account_name" type="xsd:string"/>
       <xsd:element name="account_currency" type="xsd:string"/>
```

```
            <xsd:element name="account_residency" type="xsd:string" minOccurs="0"/>
         </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="bank_definition">
         <xsd:sequence>
            <xsd:element name="swift_code" type="xsd:string"/>
            <xsd:element name="branch_number" type="xsd:string"/>
            <xsd:element name="party_identification" type="xsd:string" minOccurs="0"/>
            <xsd:element name="party_name" type="xsd:string"/>
            <xsd:element name="street_address" type="xsd:string"/>
            <xsd:element name="city" type="xsd:string"/>
            <xsd:element name="province" type="xsd:string"/>
            <xsd:element name="country_code" type="xsd:string"/>
            <xsd:element name="postal_code" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="party_definition">
         <xsd:sequence>
            <xsd:element name="party_identification" type="xsd:string"/>
            <xsd:element name="party_name" type="xsd:string"/>
            <xsd:element name="street_address" type="xsd:string"/>
            <xsd:element name="city" type="xsd:string"/>
            <xsd:element name="province" type="xsd:string"/>
            <xsd:element name="country_code" type="xsd:string"/>
            <xsd:element name="postal_code" type="xsd:string"/>
            <xsd:element name="phone_number" type="xsd:string"/>
            <xsd:element name="fax_number" type="xsd:string"/>
            <xsd:element name="email_account" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="additional_info_definition">
         <xsd:sequence>
            <xsd:element name="transaction_contract_number" type="xsd:string"
           minOccurs="0"/>
            <xsd:element name="transaction_type_code" type="xsd:string"
           minOccurs="0"/>
            <xsd:element name="bank_pre_authorization_id" type="xsd:string"
           minOccurs="0"/>
            <xsd:element name="bank_pre_authorization_status" type="xsd:string"
           minOccurs="0"/>
            <xsd:element name="revocable_status" type="xsd:string" minOccurs="0"/>
            <xsd:any minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
</xsd:schema>
```

A.11.6 General ledger posting schema

The Wallstreet XML general ledger posting schema is defined by the `tremaCompanyGLExport.xsd` file. The following are the contents of this file:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.trema.com/externalinterface/XMLschema"
elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.trema.com/externalinterface/XMLschema">
    <xsd:annotation>
       <xsd:documentation xml:lang="en">
          This is the standard Trema XML receipt message (Direct Debit, Letter of
         Credit and Pre advice) schema definition Copyright 2006 trema.com. All
         rights reserved.
```

```
      </xsd:documentation>
   </xsd:annotation>
   <xsd:element name="transactions">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="transaction" maxOccurs="unbounded"
           type="transaction_definition"/>
         </xsd:sequence>
         <xsd:attribute name="document_reference_id" use="required"
        type="xsd:string"/>
         <xsd:attribute name="format_code" use="required"
        fixed="TREMA_COMPANY_GL"/>
      </xsd:complexType>
   </xsd:element>
   <xsd:complexType name="transaction_definition">
      <xsd:attribute name="sequence" use="required" type="xsd:positiveInteger"/>
      <xsd:sequence>
         <xsd:element name="accounting_period_number" type="xsd:positiveInteger"/>
         <xsd:element name="gl_group_id" type="xsd:positiveInteger"/>
         <xsd:element name="gl_account_code" type="xsd:string"/>
         <xsd:element name="txn_direction" type="txn_direction_definition"/>
         <xsd:element name="txn_date" type="xsd:date"/>
         <xsd:element name="txn_amount" type="xsd:decimal"/>
         <xsd:element name="txn_currency" type="xsd:string"/>
         <xsd:element name="base_currency_amount" type="xsd:decimal"
        minOccurs="0"/>
         <xsd:element name="base_currency" type="xsd:string" minOccurs="0"/>
         <xsd:element name="description" type="xsd:string" minOccurs="0"/>
         <xsd:element name="party_id" type="xsd:string" minOccurs="0"/>
         <xsd:element name="bank_account_number" type="xsd:string" minOccurs="0"/>
         <xsd:element name="cpty_id" type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:simpleType name="txn_direction_definition">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="C"/>
         <xsd:enumeration value="D"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:schema>
```

A.12 Supplemental information for EDIFACT formats

This section contains supplemental information for EDIFACT formats.

A.12.1 EDIFACT Level A character set

The PAYEXT message file format supports the EDIFACT Level A character set. The tables in the previous section indicated the type of characters supported by each element:

ID	Name
a	Alphabetic characters
n	Numeric characters
an	Alphanumeric characters

A.12.1.1 Alphabetic characters (a)

Alphabetic characters include letters A to Z (uppercase only) and the following additional characters:

```
SPACE * . , - ( ) / = + : ’
```

Do not use EDIFACT syntax characters (+, :, and ’) in EDI message data as they are interpreted as separators. If you need to use one of these characters, proceed it with a question mark (?).

Certain Latin characters can be imported into CMM and, at payment release, are translated into the appropriate characters:

```
À Á Â Ã Ç È É Ê Ë Ì Í Î Ï Ð Ñ Ò Ó Ô Õ × Ù Ú Û · Ý à á â ã ç è é ê ë ì í î ï ð ñ ò
ó ô õ ÷ ù ú û ý þ
```

A.12.1.2 Numeric characters (n)

Numeric characters include numerals 0 to 9, the negative sign (-), the thousand separator (,) and the decimal point (.).

Note: The length of the number does not include the decimal point or negative sign. For example, an element of length n..5 could contain `-12345` or `123.45`.

A.12.1.3 Alphabetic characters (an)

Alphabetic characters include all alphabetic and number characters.

A.12.1.4 Element lengths

For variable-length elements, each of the above abbreviations is followed by two periods (..) and a number, where the number denotes the maximum number of digits or characters in the element.

For fixed-length elements, the periods are omitted and the number denotes the exact number of digits or characters that must be in the data element.

A.13 Supplemental information for SWIFT formats

This section contains supplemental information for SWIFT formats.

A.13.1 SWIFT field lengths

The following table presents field lengths supported by the SWIFT message file formats:

Field length	Description
nn	Maximum length (minimum is one)
nn to nn	Minimum and maximum lengths
nn!	Fixed length
nnxnn	Maximum number of lines times maximum line length

A.13.2 SWIFT field types

The following table presents field types supported by the SWIFT message file formats:

Field type	Name	Allowed characters
n	Numeric	0 to 9 only.
d	Amount	0 to 9, comma (,), and period (.).
a	Alphabetic	A to Z (uppercase only).
x	SWIFT X character set	See A.13.3 SWIFT character sets on page 304.
y	SWIFT Y character set	See A.13.3 SWIFT character sets on page 304.
z	SWIFT Z character set	See A.13.3 SWIFT character sets on page 304.

A.13.3 SWIFT character sets

SWIFT supports three character sets:

- X
- Y
- Z.

A.13.3.1 SWIFT X character set

The following table presents the allowed characters in the SWIFT X character set:

Category	Allowed characters
Alphabetic	A to Z (uppercase), a to z (lowercase)
Numeric	0 to 9
Special	/ - ? : ( ) . , ’ + SPACE CrLF

A.13.3.2 SWIFT Y character set

The following table presents the allowed characters in the SWIFT Y character set:

Category	Allowed characters
Alphabetic	A to Z (uppercase)
Numeric	0 to 9
Special	SPACE . , - ( ) / = ’ + : ?
Special (incompatible with international telex)	! " % & * ; < >

A.13.3.3 SWIFT Z character set

The following table presents the allowed characters in the SWIFT Z character set:

Category	Allowed characters
Alphabetic	A to Z (uppercase), a to z (lowercase)
Numeric	0 to 9
Special	. , - ( ) / = ’ + : ? @ # CrLF SPACE {

Category	Allowed characters
Special (incompatible with international telex)	! " % & * ; < >

A.13.4 SWIFT transaction type identification code mappings

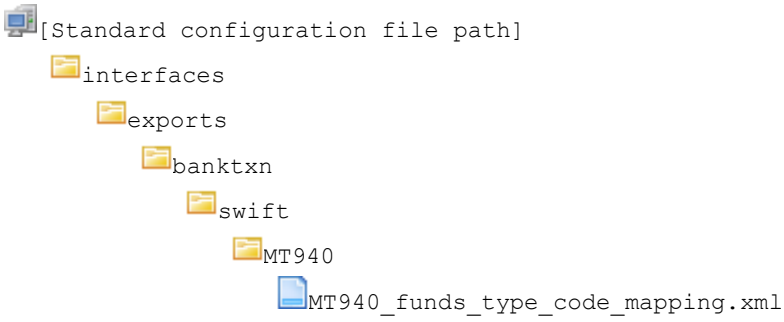
The SWIFT transaction type identification codes in tag 61 are mapped to a subset of CMM cash flow types as defined in the `MT940_funds_type_code_mapping.xml` file.

The following table presents the default mapping of SWIFT transaction type identification codes to CMM cash flow types in the `MT940_funds_type_code_mapping.xml` file:

SWIFT transaction type identification code	CMM cash flow type
TRF	AGGRP
CHG	BANKFEE
INT	BANKINT
TRF	COMMP
COL	COMMR
CHG	COMMFEE
TRF	INTER_ACCT
CHG	OVDRFEE
TRF	POOL
TRF	RCNADJ
CHG	TAX
CMZ	ZBA

If you create a new instrument type, you need to add it to the `MT940_funds_type_code_mapping.xml` file to include it in SWIFT MT940 or MT940 BCS exported bank statements:

1. Open the following configuration file:

   For instructions on opening configuration files, see “Opening configuration files” on page32.

2. Add a mapping element for the new cash flow type, where the value of the `attribute_value` attribute is the cash flow type ID and the value of the `spec_value` attribute is the SWIFT transaction type identification code.

   The following is an example:

   ```
   <mapping attribute_value="GVTTAX" spec_value="CHG"/>
   ```

3. Save and close the file.

A.13.5 SWIFT business transaction code (GVC) mappings

This mapping is used in section 1 of tag 86 in SWIFT MT940 BCS messages:

Cash flow type			GVC	
ID	**Name**	**Category**	**ID**	**Name**
BANKFEE	Bank Fee	COMM	809	Provisionen
BANKINT	Bank Interest	COMM	814	Zinsen
COMMFEE	Commitment Fee	COMM	809	Provisionen
COMMP	Commercial Payment – Not Deal	COMM	020	Überweisungsauftrag
COMMR	Commercial Receipt – Not Deal	COMM	051	Überweisungsgutschrift
OVDRFEE	Overdraft Fee	COMM	809	Provisionen
CM-CHEQUES	CM-CHEQUES	COMM	074	TC (Scheckbelastung)
CM-CHEQCOL	CM-CHEQUE-COLLECTION	COMM	070	Scheckeinreichung
CM-CHEQPAF	CM-CHEQUE-PAYM-FACT	COMM	209	Zahlung per Scheck
CM-CHQDCNA	CM-CHEQUE-DC-NACOS	COMM	074	TC (Scheckbelastung)
CM-PAFDOME	CM-PAYM-FACT-DOMESTIC	COMM	020	Überweisungsauftrag
CM-PAFXBOR	CM-PAYM-FACT-XBORDER	COMM	201	Zahlungsauftrag
CM-PFRVDOM	CM-PAYM-FACT-NOT-SETTL-DOMEST	COMM	009	Retourenhülle (Lastschrift)
CM-PFRVXBO	CM-PAYM-FACT-NOT-SETTL-XBORDER	COMM	009	Retourenhülle (Lastschrift)
CM-CACONIC	CM-CASH-CONCENTRATION-IC	COMM	833	Cash Concentrating: Buchung Hauptkonten
CM-CACONDC	CM-CASH-CONCENTRATION-DC	COMM	833	Cash Concentrating: Buchung Hauptkonten
CM-ICPROBO	CM-IC-PMT-REC-ON-BEHALF	COMM	051	Überweisungsgutschrift
CM-ICCLEAR	CM-IC-CLEARING	COMM	834	Cash Concentration: Avisinformation Nebenkonten
CM-ICSETPF	CM-IC-SETTL-PAYM-FACT	COMM	012	Zahlungsanweisung zur Verrechnung
CM-ICSETMA	CM-IC-SETTLEMENT-MANUAL	COMM	012	Zahlungsanweisung zur Verrechnung
CM-ICINTER	CM-IC-INTEREST	COMM	814	Zinsen
CM-ICCOMMT	CM-IC-COMMITMENT-FEE	COMM	809	Provisionen
CM-ICOVERD	CM-IC-OVERDRAFT-FEE	COMM	809	Provisionen
CM-ICGUFEE	CM-IC-GUARANTEE-FEE	COMM	809	Provisionen
GL-BANKFEE	GL-BANK-FEE	COMM	809	Provisionen
GL-BANKINT	GL-BANK-INTEREST	COMM	814	Zinsen
GL-COMMFEE	GL-COMMITMENT-FEE	COMM	809	Provisionen

Cash flow type			GVC	
GL-OVDRFEE	GL-OVERDRAFT-FEE	COMM	809	Provisionen
GL-LCREFEE	GL-LETTER-OF-CREDIT-FEE	COMM	809	Provisionen
GL-COMMCOP	GL-COMMISSION-FOR-CP	COMM	809	Provisionen
GL-PAYMRET	GL-PAYMENT-RETURN	COMM	009	Retourenhülle (Lastschrift)
TR-OA-LOAN-DEPOSIT-IAM	TR-OA-LOAN-DEPOSIT-IAM	TRM	823	Festgeld
TR-OA-LOAN-DEPOSIT-FIXED	TR-OA-LOAN-DEPOSIT-FIXED	TRM	823	Festgeld
TR-OA-LOAN-DEPOSIT-FLOAT	TR-OA-LOAN-DEPOSIT-FLOAT	TRM	823	Festgeld
TR-CALL-MONEY	TR-CALL-MONEY	TRM	823	Festgeld
TR-CP-INVEST	TR-CP-INVEST	TRM	823	Festgeld
TR-ABCP-INVEST	TR-ABCP-INVEST	TRM	823	Festgeld
TR-OA-DISCOUNT	TR-OA-DISCOUNT	TRM	823	Festgeld
TR-STOCK	TR-STOCK	TRM	303	Effekten
TR-SPECIAL-FUND	TR-SPECIAL-FUND	TRM	303	Effekten
TR-INVESTMENT-FUND	TR-INVESTMENT-FUND	TRM	303	Effekten
TR-WARRANT	TR-WARRANT	TRM	303	Effekten
TR-CERTIFICATE	TR-CERTIFICATE	TRM	303	Effekten
TR-LI-LOAN-DEPOSIT-IAM	TR-LI-LOAN-DEPOSIT-IAM	TRM	823	Festgeld
TR-LI-LOAN-DEPOSIT-FIXED	TR-LI-LOAN-DEPOSIT-FIXED	TRM	823	Festgeld
TR-LI-LOAN-DEPOSIT-FLOAT	TR-LI-LOAN-DEPOSIT-FLOAT	TRM	823	Festgeld
TR-CP-ISSUE	TR-CP-ISSUE	TRM	823	Festgeld
TR-ABCP-ISSUE	TR-ABCP-ISSUE	TRM	823	Festgeld
TR-LI-DISCOUNT	TR-LI-DISCOUNT	TRM	823	Festgeld
TR-IR-SWAP-IRS	TR-IR-SWAP-IRS	TRM	405	Finanzinnovation
TR-IR-SWAP- CCIRS	TR-IR-SWAP-CCIRS	TRM	405	Finanzinnovation
TR-IR-SWAP-LEG	TR-IR-SWAP-LEG	TRM	405	Finanzinnovation
TR-FRA	TR-FRA	TRM	405	Finanzinnovation
TR-CAP-FLOOR-COLLAR	TR-CAP-FLOOR-COLLAR	TRM	405	Finanzinnovation
TR-SWAPTION-IRS	TR-SWAPTION-IRS	TRM	405	Finanzinnovation
TR-MM-FUTURE	TR-MM-FUTURE	TRM	405	Finanzinnovation
TR-OPTION-MM-FUTURE	TR-OPTION-MM-FUTURE	TRM	405	Finanzinnovation
TR-BOND-FUTURE	TR-BOND-FUTURE	TRM	405	Finanzinnovation

Cash flow type			GVC	
TR-OPTION-BOND-FUTURE	TR-OPTION-BOND-FUTURE	TRM	405	Finanzinnovation
TR-CO-SPOT-METAL	TR-CO-SPOT-METAL	TRM	405	Finanzinnovation
TR-CO-SPOT-OTHER	TR-CO-SPOT-OTHER	TRM	405	Finanzinnovation
TR-CO-SWAP-METAL	TR-CO-SWAP-METAL	TRM	405	Finanzinnovation
TR-CO-SWAP-OTHER	TR-CO-SWAP-OTHER	TRM	405	Finanzinnovation
TR-CO-OPTION	TR-CO-OPTION	TRM	405	Finanzinnovation
TR-FX-SPOT	TR-FX-SPOT	TRM	402	Termindevisen
TR-FX-FORWARD	TR-FX-FORWARD	TRM	402	Termindevisen
TR-FX-SWAP	TR-FX-SWAP	TRM	402	Termindevisen
TR-FX-OPTION	TR-FX-OPTION	TRM	402	Termindevisen
TR-EQ-FUTURE	TR-EQ-FUTURE	TRM	405	Finanzinnovation
TR-OPTION-STOCK	TR-OPTION-STOCK	TRM	405	Finanzinnovation
TR-OPTION-INDEX	TR-OPTION-INDEX	TRM	405	Finanzinnovation
All other cash flow types			835	

A.14 Syntactic validation of incoming and outgoing MX messages

The ESIAdapter SWIFT connectivity module supports syntax checking of ISO20022 messages sent or received over FileAct. Currently, only WebSuite supports ISO20022 messages.

This validation checks incoming and outgoing messages against corresponding XSDs.

Here is the list of XSDs currently associated with this checking:

camt_026_001_03

camt_027_001_03

camt_028_001_03

camt_029_001_03

camt_030_001_03

camt_031_001_03

camt_032_001_02

camt_033_001_03

camt_034_001_03

camt_035_001_02

camt_036_001_02

camt_037_001_03

camt_038_001_02

camt_039_001_03

camt_052_001_02

camt_053_001_02

camt_054_001_02

camt_055_001_01

camt_056_001_01

pain_001_001_02

pain_001_001_03

pain_002_001_03

pain_007_001_02

pain_008_001_02

pain_009_001_01

pain_010_001_01

pain_011_001_01

pain_012_001_01

pacs_002_001_03

pacs_003_001_02

pacs_004_001_02

pacs_007_001_02

pacs_008_001_02

pacs_009_001_02

Should you require that this list be amended, please contact Wallstreet Systems Customer Support.

Appendix B Attributes

CMM supports a set of attributes that you can use in files based on Wallstreet standard XML formats or customized formats:

- For information on Wallstreet standard XML formats, see Appendix A Standard formats on page 133.
- For information on customized formats, see Chapter 7 Using XML-template-based formats on page 119.

In these formats, a value can be derived in one of the following ways (called "map types"):

Name	Description	Example
CMM No Map	• The value is identified by an attribute ID. • The value is obtained from CMM through the attribute provider. • The value is not mapped (in other words, the value in CMM is the value in the import or export).	ISO country codes that reside in CMM are the same as those in EDI messages.
CMM Map	• The value is identified by an attribute ID. • The value is obtained from CMM through the attribute provider. • The value is mapped in CMM. • The mapping logic resides in transaction subtype mapping or controller preprocessing.	Remittance detail codes are mapped from CMM internal codes to EDI standard codes using transaction sub type mapping.
Spec Map	• The value is identified by an attribute ID. • The value is obtained from CMM through the attribute provider. • The value is mapped in the template message generator. • The mapping logic resides in XML files similar to the format/validation specifications.	The domestic or international flag is mapped to format standard and bank required values through mappings defined in XML files.
Spec Derived	• The value is identified by an attribute ID. • The value is derived in the template message generator.	The number of successful transactions written to the file can by derived from specifications.
Static No Map	• The value is hard-coded in the specification. • The value is not based on a CMM value but on a segment/element specification.	EDIFACT segment tags like FII and NAD are static constants in specifications.

B.1 Import attributes

CMM supports the following import attributes:

- Forecast
- Accounts payable
- Accounts receivable
- Bank transaction
- Bank balance
- Bank message
- Interchange
- File.

B.1.1 Forecast import attributes

The following table presents common forecast import attributes supported by CMM:

Attribute	Description	Value	Constraints	Req.	Source	Map type
Basic						
`currency_code`	The forecast's currency.	Currency ID	• None	Yes	User	CMM No Map
`amount`	The forecast's amount (in the forecast's currency).	Floating point number	• None	Yes	User	CMM No Map
`bucket`	The forecast's time bucket, which indicates whether the forecast is short-term or long-term.	One of the following values: • `day` • `month`	• None	Yes	User	CMM No Map
`cash_flow_direction`	A value that indicates whether the forecast is a payment or a receipt.	One of the following values: • `P` if the forecast is a payment • `R` if the forecast is a receipt	• None	Yes	User	CMM No Map

Attribute	Description	Value	Constraints	Req.	Source	Map type
instrument_type	The forecast's cash flow type. By assigning cash flow types to forecasts and other activity in CMM, you can categorize this activity for a variety of purposes. In particular, you can run several reports in the module that display activity by cash flow type. Note: In CMM, a cash flow type is an instrument type with the **Status** attribute set to Enabled.	Instrument type ID	• None	No	User	CMM No Map
payment_method	The forecast's payment method. A forecast's payment method defines how the forecast is to be paid (for example, by EFT or by check) if it becomes a releasable transaction.	Payment method ID	• None	No	User	CMM No Map
value_date	The forecast's starting value date. Note: Medium- and long-term forecasts can span a range of value dates (for example, a medium-term forecast can span a month), while short-term forecasts usually only span one value date.	Date	• None	Yes	User	CMM No Map
invoice_date	The forecast's invoice date.	Date	• None	No	User	CMM No Map

Attribute	Description	Value	Constraints	Req.	Source	Map type
work_flow_action	The action CMM should take upon importing the forecast. Note: If you do not specify a value in this column, CMM creates a new record for the forecast upon importing it.	One of the following values: • edit • cancel	• None	No	User	CMM No Map
description	A relevant description of or other information pertaining to the forecast.	Text (2,000 characters maximum)	• None	No	User	CMM No Map
Party						
party_id	The forecast's party.	Entity ID	• None	Yes	User	CMM No Map
Party bank						
party_bank_id	The bank where the forecast's party bank account is held.	In-house bank ID External bank ID	• Must be provided if the party bank account number is also provided	No	User	CMM No Map
Party bank account						
party_bank_account_number	The forecast's party bank account. It is not usually necessary to define a party bank account for a forecast. However, this attribute is provided for situations in which you need to define a party bank account for a short-term forecast (or "cash position"). Note: Do not include format characters (_, -, and so on) in this column.	Entity bank account number	• None	No	System	CMM No Map

Attribute	Description	Value	Constraints	Req.	Source	Map type
party_bank_account_currency_code	The curency of the forecast's party bank account.	Currency ID	• Must be provided if the party bank account number is also provided	No	User	CMM No Map
Counterparty						
cpty_id	The forecast's counterparty. It is not usually necessary to define a counterparty for a forecast. However, this attribute is provided for situations in which you need to define a counterparty for a short-term forecast (or "cash position").	Entity ID Counterparty ID	• None	CMM No Map		
Counterparty bank						
cpty_bank_id	The ID of the bank where the forecast's counterparty bank account is held.	In-house bank ID External bank ID	• Must be provided if the counterparty bank account number is also provided	No	User	CMM No Map
Counterparty bank account						
cpty_bank_account_number	The bank account number of the forecast's counterparty bank account. Note: Do not include format characters (_, -, and so on) in this column.	Entity bank account number Counterparty bank account number	• None	No	User	CMM No Map

Attribute	Description	Value	Constraints	Req.	Source	Map type
cpty_bank_account_currency_code	The ISO currency code of the forecast's counterparty bank account.	Currency ID	• Must be provided if the counterparty bank account number is also provided	No	User	CMM No Map
Other						
customer_reference_id	The forecast's source reference text ID.	Text (50 characters maximum)	• None	No	User	CMM No Map
customer_attribute_0 customer_attribute_1 customer_attribute_2 customer_attribute_3 customer_attribute_4 customer_attribute_5 customer_attribute_6 customer_attribute_7 customer_attribute_8 customer_attribute_9	The forecast's custom attributes. Note: You or another user in your organization can define custom attributes (or "parameters") for forecasts in the Parameter Editor function.	Text (50 characters maximum per field)	• None	No	User	CMM No Map

B.1.2 Accounts payable import attributes

The following table presents common accounts payable import attributes supported by CMM:

Attribute	Description	Value	Constraints	Req.	Source	Map type
Basic						
currency	The transaction's currency.	Currency ID	• None	Yes	User	CMM No Map
amount	The transaction's absolute value (in the transaction's currency).	Floating point number	• None	Yes	User	CMM No Map

Attribute	Description	Value	Constraints	Req.	Source	Map type
priority_code	The transaction's priority.	One of the following values: • 1 if the transaction is not urgent • 2 if the transaction is urgent Note: 1 is the default if you do not specify a value.	• None	No	User	CMM No Map
cash_flow_type_id	The transaction's cash flow type. By assigning cash flow types to transactions and other activity in CMM, you can categorize this activity for a variety of purposes. In particular, you can run several reports in the module that display activity by cash flow type. Note: In CMM, a cash flow type is an instrument type with the **Status** attribute set to Enabled.	Instrument type ID Note: COMMP (Commercial Payment) is the default if you do not specify a value.	• None	No	User	CMM No Map
payment_method	The transaction's payment method. A transaction's payment method defines how the transaction is to be paid when it is released (for example, by EFT or by check).	Payment method ID	• None		User	CMM No Map
txn_date	The transaction's transaction date.	Date Note: The current date is the default if you do not specify a value.	• None	No	User	CMM No Map

Attribute	Description	Value	Constraints	Req.	Source	Map type
value_date	The transaction's value date.	Date Note: The transaction date is the default if you do not specify a value.	• None	No	User	CMM No Map
description	A relevant description of or other information pertaining to the transaction.	Text (2,000 characters maximum)	• None	No	User	CMM No Map
Party						
external_source_party_id [1]	The external system ID of the transaction's party.	Text (25 characters maximum)	• None	No	User	CMM Map
party_id [2]	The transaction's party.	Entity ID	• None	Yes	User	CMM No Map
Party bank						
party_bank_branch_id	The branch ID of the transaction's party bank (as assigned by an agency).	Text (15 characters maximum)	• None	No	User	CMM No Map
Party bank account						
party_bank_account_number [3]	The transaction's party bank account. Note: If you do not provide a party bank account, CMM can select one using routing rules.	Entity bank account primary number	• Must be held by the transaction's party	No	User	CMM No Map
party_bank_account_currency_code [4]	The currency of the transaction's party bank account.	Currency ID	• None	No	User	CMM No Map
Counterparty						
cpty_id [5]	The transaction's counterparty.	Entity ID Counterparty ID	• None	No	User	CMM No Map
cpty_name	The name of the transaction's counterparty.	Text (35 characters maximum)	• None	No	User	CMM No Map

Attribute	Description	Value	Constraints	Req.	Source	Map type
cpty_local_address_line_1 cpty_local_address_line_2 cpty_local_address_line_3 cpty_local_address_line_4	The address of the transaction's counterparty.	Text (35 characters maximum per field)	• None	No	User	CMM No Map
cpty_city_name	The city of the transaction's counterparty.	Text (35 characters maximum)	• None	No	User	CMM No Map
cpty_state_name	The state of the transaction's counterparty.	State ID	• None	No	User	CMM No Map
cpty_country_code	The country of the transaction's counterparty.	Country ID	• None	No	User	CMM No Map
cpty_postal_code	The postal or ZIP code of the transaction's counterparty.	Text (10 characters maximum)	• None	No	User	CMM No Map
Counterparty bank						
cpty_bank_id [6]	The ID of the transaction's counterparty bank.	In-house bank ID External bank ID	• None	No	User	CMM No Map
cpty_bank_swift_code	The SWIFT code of the transaction's counterparty bank account.	Text (20 characters maximum)	• None	No	User	CMM No Map
cpty_bank_branch_id	The branch ID of the transaction's counterparty bank (as assigned by an agency).	Text (15 characters maximum)	• None	No	User	CMM No Map
cpty_bank_name	The name of the transaction's counterparty bank.	Text (35 characters maximum)	• None	No	User	CMM No Map
cpty_bank_address_1 cpty_bank_address_2	The address of the transaction's counterparty bank.	Text (35 characters maximum per field)	• None	No	User	CMM No Map
cpty_bank_city_name	The city of the transaction's counterparty bank.	Text (35 characters maximum)	• None	No	User	CMM No Map
cpty_bank_state_name	The state of the transaction's counterparty bank.	State ID	• None	No	User	CMM No Map
cpty_bank_postal_code	The postal or ZIP code of the transaction's counterparty bank.	Text (10 characters maximum)	• None	No	User	CMM No Map

Attribute	Description	Value	Constraints	Req.	Source	Map type
Counterparty bank account						
cpty_bank_account_id [7]	The ID of the transaction's counterparty bank.	In-house bank ID External bank ID	• None	No	User	CMM No Map
cpty_bank_account_number	The primary number (usually, BBAN) of the transaction's counterparty bank account.	Text (60 characters maximum)	• Must validate against the BBAN rules of the bank account's country	No	User	CMM No Map
cpty_bank_account_iban	The secondary number (usually IBAN) of the transaction's counterparty bank account.	Text (60 characters maximum)	• Must validate against the IBAN rules	No	User	CMM No Map
cpty_bank_account_country_code	The country of the transaction's counterparty bank.	Country ID	• None	No	User	
cpty_bank_account_currency_code	The currency of the transaction's counterparty bank account.	Currency ID	• None	No	User	CMM No Map
Intermediary bank						
intermediary_bank_id [8]	The transaction's primary intermediary bank.	In-house bank ID External bank ID	• None	No	User	CMM No Map
intermediary_bank_swift_code	The SWIFT code of the transaction's intermediary bank account.	Text (40 characters maximum)	• None	No	User	CMM No Map
intermediary_bank_branch_id	The branch ID of the transaction's intermediary bank (as assigned by an agency).	Text (15 characters maximum)	• None	No	User	CMM No Map
intermediary_bank_name	The name of the transaction's intermediary bank.	Text (35 characters maximum)	• None	No	User	CMM No Map
intermediary_bank_address_line_1 intermediary_bank_address_line_2	The address of the transaction's intermediary bank.	Text (35 characters maximum per field)	• None	No	User	CMM No Map
intermediary_bank_city_name	The city of the transaction's intermediary bank.	Text (35 characters maximum)	• None	No	User	CMM No Map
intermediary_bank_state_name	The state of the transaction's intermediary bank.	State ID	• None	No	User	CMM No Map

Attribute	Description	Value	Constraints	Req.	Source	Map type
intermediary_bank_country_code	The country of the transaction's intermediary bank.	Country ID	• None	No	User	CMM No Map
intermediary_bank_postal_code	The postal or ZIP code of the transaction's intermediary bank.	Text (10 characters maximum)	• None	No	User	CMM No Map
Intermediary bank account						
intermediary_bank_account_id [9]	The transaction's primary intermediary bank account.	Entity bank account ID	• None	No	User	CMM No Map
intermediary_bank_account_number [10]	The primary number (usually BBAN) of the transaction's intermediary bank account.	Text (60 characters maximum)	• Must validate against the BBAN rules of the bank account's country	No	User	CMM No Map
Secondary intermediary bank						
second_intermediary_bank_id [11]	The transaction's secondary intermediary bank.	In-house bank ID External bank ID	• None	No	User	CMM No Map
second_intermediary_bank_swift_code	The SWIFT code of the transaction's intermediary bank account.	Text (40 characters maximum)	• None	No	User	CMM No Map
second_intermediary_bank_branch_id	The branch ID of the transaction's intermediary bank (as assigned by an agency).	Text (15 characters maximum)	• None	No	User	CMM No Map
second_intermediary_bank_name	The name of the transaction's intermediary bank.	Text (35 characters maximum)	• None	No	User	CMM No Map
second_intermediary_bank_address_line_1 second_intermediary_bank_address_line_2	The address of the transaction's intermediary bank.	Text (35 characters maximum per field)	• None	No	User	CMM No Map
second_intermediary_bank_city_name	The city of the transaction's intermediary bank.	Text (35 characters maximum)	• None	No	User	CMM No Map
second_intermediary_bank_state_name	The state of the transaction's intermediary bank.	State ID	• None	No	User	CMM No Map
second_intermediary_bank_country_code	The country of the transaction's intermediary bank.	Country ID	• None	No	User	CMM No Map

Attribute	Description	Value	Constraints	Req.	Source	Map type
second_intermediary_bank_postal_code	The postal or ZIP code of the transaction's intermediary bank.	Text (10 characters maximum)	• None	No	User	CMM No Map
Second intermediary bank account						
second_intermediary_bank_account_id [12]	The transaction's secondary intermediary bank account.	Entity bank account ID	• None	No	User	CMM No Map
second_intermediary_bank_account_number [13]	The primary number (usually BBAN) of the transaction's intermediary bank account.	Text (60 characters maximum)	• Must validate against the BBAN rules of the bank account's country	No	User	CMM No Map
Aggregation						
aggregate_detail_code	A value that indicates whether the transaction is an aggregate or detail.	One of the following values: • A if the transaction is an aggregate • D if the transaction is a detail	• None	No	User	CMM No Map
aggregate_reference_id	If the transaction is an aggregate, the transaction's unique reference number. If the transaction is a detail, the reference number of the aggregate transaction in which the detail transaction is contained.	Text (50 characters maximum)	• None	No	User	CMM No Map
Tax						
tax_payment_type_code	A code identifying the type of tax being paid.	Text (255 characters maximum)	• None	No	User	CMM No Map
tax_payment_subtype_code	A code identifying the subtype of tax being paid.	Text (255 characters maximum)	• None	No	User	CMM No Map

Attribute	Description	Value	Constraints	Req.	Source	Map type
tax_payer_id	A number assigned to a purchaser by a taxing jurisdiction. Note: This ID is also known as a tax exemption number or certificate number.	Text (255 characters maximum)	• None	No	User	CMM No Map
tax_payer_verification_id	An ID the receiver can use to confirm the taxpayer's identity.	Text (255 characters maximum)	• None	No	User	CMM No Map
Regulatory reporting						
regulatory_code	The transaction's regulatory code.	Regulatory code	• None	No	User	CMM No Map
Other						
bank_preauthorization_id	The transaction's pre-authorization ID.	Text (255 characters maximum)	• None	No	User	CMM No Map
bank_preauthorization_status	The transaction's pre-authorization status.	One of the following: • AUTHORIZED • UNAUTHORIZED	• None	No	User	CMM No Map
cheque_number	The transaction's check number (if the transaction is paid by check).	Text (35 characters maximum)	• None	No	User	CMM No Map
cpty_message	Instructions or other messages to the transaction's counterparty.	Text (100 characters maximum)	• None	No	User	CMM No Map
customer_batch_reference_id	The customer's batch reference ID for the transaction.	Text (30 characters maximum)	• None	No	User	CMM No Map
customer_reference_id	The transaction's customer reference ID.	Text (50 characters maximum)	• None	No	User	CMM No Map
customer_txn_reference_id	The customer's reference ID for the transaction. Note: This ID does not have to be unique.	Text (30 characters maximum)	• None	No	User	CMM No Map

Attribute	Description	Value	Constraints	Req.	Source	Map type
external_reference	The reference number or numbers of an external transaction as supplied by an external system (includes TRM).	String. List of values delimited by ; (semicolons) which can be escaped with \ (backslash). Each value must not exceed 255 characters.	• None	No	User	CMM No Map
external_transaction_type_code	The ID of the transaction's type. Note: This is country and bank specific.	Text (10 characters maximum)	• None	No	User	CMM No Map
financial_allocation_charge	The financial allocation charge of the transaction.	One of the following values: • 1 if the financial charges are being shared equally between the sender and receiver • 2 if the financial charges are being paid entirely by the sender • 3 if the financial charges are being paid entirely by the receiver Note: 1 is the default if you do not specify a value.	• None	No	User	CMM No Map
perosnal_identification_number	The personal identification number of the transaction.	Text (255 characters maximum)	• None	No	User	CMM No Map
remittance_details	The transaction's remittance details. Remittance detail information can be captured for each transaction. Multiple remittance detail information is possible for one transaction.	Text	• None	No	User	CMM No Map
repetitive_code	A bank-defined code that uniquely identifies the static attributes of the transaction.	Text (255 characters maximum)	• None	No	User	CMM No Map

Attribute	Description	Value	Constraints	Req.	Source	Map type
retrieval_location_address_1 retrieval_location_address_2	The address of the transaction's pickup location. Note: This attribute is used in eastern Europe.	Text (255 characters maximum per field)	• None	No	User	CMM No Map
retrieval_location_id	The ID of the transaction's pickup location. Note: This attribute is used in eastern Europe.	Text (255 characters maximum)	• None	No	User	CMM No Map
retrieval_location_name	The name of the transaction's pickup location. Note: This attribute is used in eastern Europe.	Text (255 characters maximum)	• None	No	User	CMM No Map

Table notes:

1. This attribute and party_id are mutually exclusive.
2. This attribute and external_source_party_id are mutually exclusive.
3. If you do not provide this attribute, you must provide either external_source_party_id or party_id.
4. This attribute is required if party_bank_account_number is also provided.
5. This attribute and all other counterparty attributes are mutually exclusive.
6. This attribute and all other counterparty bank attributes are mutually exclusive.
7. This attribute and all other counterparty bank account attributes are mutually exclusive.
8. This attribute and all other intermediary bank attributes are mutually exclusive.
9. This attribute and intermediary_bank_account_number are mutually exclusive.
10. This attribute and intermediary_bank_account_id are mutually exclusive.
11. This attribute and all other second intermediary bank attributes are mutually exclusive.
12. This attribute and second_intermediary_bank_account_number are mutually exclusive.
13. This attribute and second_intermediary_bank_account_id are mutually exclusive.

B.1.3 Accounts receivable import attributes

The following table presents common accounts receivable import attributes supported by CMM:

Attribute	Description	Value	Constraints	Req.	Source	Map type
Basic						
currency	The transaction's currency.	Currency ID	• None	Yes	User	CMM No Map

Attribute	Description	Value	Constraints	Req.	Source	Map type
amount	The transaction's absolute value (in the transaction's currency).	Floating point number	• None	Yes	User	CMM No Map
cash_flow_type_id	The transaction's cash flow type. By assigning cash flow types to transactions and other activity in CMM, you can categorize this activity for a variety of purposes. In particular, you can run several reports in the module that display activity by cash flow type. Note: In CMM, a cash flow type is an instrument type with the **Status** attribute set to Enabled.	Instrument type ID Note: COMMR (Commercial Receipt) is the default if you do not specify a value.	• None	No	User	CMM No Map
payment_method	The transaction's payment method. A transaction's payment method defines how the transaction is to be paid when it is released (for example, by EFT or by check).	Payment method ID	• None	Yes	User	CMM No Map
txn_date	The transaction's transaction date.	Date Note: The current date is the default if you do not specify a value.	• None	No	User	CMM No Map
value_date	The transaction's value date.	Date Note: The transaction date is the default if you do not specify a value.	• None	No	User	
description	A relevant description of or other information pertaining to the transaction.	Text (2,000 characters maximum)	• None	No	User	CMM No Map
Party						

Attribute	Description	Value	Constraints	Req.	Source	Map type
external_source_party_id [1]	The external system ID of the transaction's party.	Text (25 characters maximum)	• None	No	User	CMM Map
party_id [2]	The transaction's party.	Entity ID	• None	Yes	User	CMM No Map
Party bank						
party_bank_branch_id	The branch ID of the transaction's party bank (as assigned by an agency).	Text (15 characters maximum)	• None	No	User	CMM No Map
Party bank account						
party_bank_account_number [3]	The transaction's party bank account. Note: If you do not provide a party bank account, CMM can select one using routing rules.	Entity bank account primary number	• Must be held by the transaction's party	No	User	CMM No Map
party_bank_account_currency_code [4]	The currency of the transaction's party bank account.	Currency ID	• None	No	User	CMM No Map
Counterparty						
external_source_cpty_id [5]	The external system ID of the transaction's party.	Text (25 characters maximum)	• None	No	User	CMM Map
cpty_id [6]	The transaction's counterparty.	Entity ID Counterparty ID	• None	No	User	CMM No Map
Other						
customer_reference_id	The transaction's customer reference ID.	Text (50 characters maximum)	• None	No	User	CMM No Map
invoice_number	The transaction's invoice number.	Text (20 characters maximum)	• Must be provided if existing references are deleted.	No	User	CMM No Map

Table notes:

1. This attribute and party_id are mutually exclusive.
2. This attribute and external_source_party_id are mutually exclusive.
3. If you do not provide this attribute, you must provide either external_source_party_id or party_id.
4. This attribute is required if party_bank_account_number is also provided.
5. This attribute and cpty_id are mutually exclusive.
6. This attribute and external_source_cpty_id are mutually exclusive.

B.1.4 Bank transaction import attributes

The following table presents common bank transaction import attributes supported by CMM:

Attribute	Description	Value	Constraints	Req.	Source	Map type
Basic						
`currency_code`	The bank transaction's currency.	Currency ID	• None	Yes	User	CMM No Map
`amount`	The bank transaction's absolute value (in the bank transaction's currency).	Floating point number	• None	Yes	User	CMM No Map
`type_code`	A value that indicates whether the bank transaction is a payment or a receipt.	One of the following: • `P` if the bank transaction is a payment • `R` if the bank transaction is a receipt	• None	No	User	CMM Map
`funds_type_code`	The bank transaction's cash flow type. By assigning cash flow types to bank transactions and other activity in CMM, you can categorize this activity for a variety of purposes. In particular, you can run several reports in the module that display activity by cash flow type. Note: In CMM, a cash flow type is an instrument type with the **Status** attribute set to `Enabled`.	Instrument type ID	• Is mutually exclusive with the counterparty file code ID	No	User	CMM Map
`external_txn_sub_type`	The bank transaction's counterparty file code.	Counterparty file code ID	• Is mutually exclusive with the cash flow type ID	No	User	CMM Map
`txn_date`	The bank transaction's transaction date.	Date	• None	Yes	User	CMM No Map

Attribute	Description	Value	Constraints	Req.	Source	Map type
value_date	The bank transaction's value date.	Date	• None	Yes	User	CMM No Map
description	A relevant description of or other information pertaining to the bank transaction.	Text (2,000 characters maximum)	• None	No	User	CMM No Map
bank_txn_stmt_start_date	Sets the statement start date. Use the time stamp 00:00:00 for the given day.	Date/Time	• None	No	User	CMM no Map
bank_txn_stmt_end_date	Sets the statement end date. Use the time stamp 23:59:59 for the given day.	Date/Time	• None	No	User	CMM no Map
Entity						
bank_acct_number	The primary number (usually, BBAN) of the bank transaction's party bank account.	Entity bank account primary number Counterparty bank account primary number	• None	Yes	User	CMM Map
bank_acct_name	The name of the bank transaction's party bank account.	Entity bank account primary name Counterparty bank account primary name	• None	No	User	CMM Map
originating_bank_id	The bank transaction's party bank.	In-house bank ID External bank ID	• None	No	User	CMM No Map
Counterparty						
cpty_bank_account_number	The primary number (usually, BBAN) of the bank transaction's counterparty bank account.	Counterparty bank account primary number	• None	No	User	CMM No Map
Other						

Attribute	Description	Value	Constraints	Req.	Source	Map type
customer_reference_id	The bank transaction's customer reference ID. Note: If you do not specify a value in this attribute, CMM cannot automatically reconcile the bank transaction.	Text (50 characters maximum)	• None	No	User	CMM No Map
bank_reference_number	The bank transaction's bank reference ID.	Text (50 characters maximum)	• None	No	User	CMM No Map
cheque_number	The bank transaction's check number (if the bank transaction is paid by check).	Text (35 characters maximum)	• None	No	User	CMM No Map

B.1.5 Bank balance import attributes

The following table presents common bank balance import attributes supported by CMM:

Attribute	Description	Value	Constraints	Req.	Source	Map type
Basic						
currency_code	The bank balance's currency.	Currency ID	• None	Yes	User	CMM No Map
actual_amount	The bank balance's value (in the bank balance's currency). Note: The value can be positive or negative.	Floating point number	• None	Yes	User	CMM No Map

Attribute	Description	Value	Constraints	Req.	Source	Map type
txn_sub_type_id	The bank balance's internal type.	One of the following: • 8 if the bank balance is opening transaction date • 19 if the bank balance is closing transaction date • 3 if the bank balance is opening value date • 6 if the bank balance is closing value date	• Is mutually exclusive with the external type	No	User	CMM No Map
external_balance_type	The bank balance's external type.	Text	• Is mutually exclusive with the internal type	No	User	CMM Map
balance_date	The bank balance's date.	Date	• None	Yes	User	CMM No Map
Entity						
bank_acct_number	The primary number (usually, BBAN) of the bank balance's party bank account.	Entity bank account primary number Counterparty bank account primary number	• None	Yes	User	CMM Map
bank_acct_name	The name of the bank balance's party bank account.	Entity bank account primary name Counterparty bank account primary name	• None	No	User	CMM Map

B.1.6 Bank message import attributes

The following table presents common bank message import attributes supported by CMM:

Attribute	Description	Value	Constraints	Req.	Source	Map type
File level						
interchange_control_ref_num	The file-level bank reference identifier.	Text (35 characters maximum)	• None	No	User	CMM No Map

Attribute	Description	Value	Constraints	Req.	Source	Map type
Transaction level						
orig_file_export_reference_id	The export log identifier in CMM for the original file exported to and sent back from the bank.	Integer	• None	No	User	CMM No Map
external_message_code	The status of the message. Note: The value in this attribute must be mapped to an internal status code in CMM, such as PA, PR, PH, or 400.	Text	• None	Yes	User	CMM Map
bank_reference_id	The bank's ID for the transaction.	Text (50 characters maximum)	• None	No	User	CMM No Map
message_text_desc	A description of the message's status.	Text	• None	No	User	CMM No Map
customer_reference_id	The originating system's unique ID for the transaction.	Text	• None	No	User	CMM No Map

B.1.7 Interchange import attributes

The following table presents common interchange import attributes supported by CMM:

Attribute	Description	Value	Constraints	Req.	Source	Map type
Basic attributes						
interchange_sender_id	The ID of the interchange's EDI sender.	Text (35 characters maximum)	• None	No	User	CMM No Map
sender_id_qualifier_code	The qualifier code of the interchange's EDI sender.	Text (4 characters maximum)	• None	No	User	CMM No Map
interchange_recipient_id	The ID of the interchange's EDI recipient.	Text (35 characters maximum)	• None	No	User	CMM No Map
recipient_id_qualifier_code	The qualifier code of the interchange's EDI receiver.	Text (4 characters maximum)	• None	No	User	CMM No Map
functional_group_sender_id	The functional group of the interchange's EDI sender.	Text (35 characters maximum)	• None	No	User	CMM No Map
functional_group_receiver_id	The functional group of the interchange's EDI receiver.	Text (35 characters maximum)	• None	No	User	CMM No Map

Attribute	Description	Value	Constraints	Req.	Source	Map type
Runtime-specific attributes						
import_export_id	A unique sequential ID assigned to every interface with CMM. Note: The value of this attribute is derived at runtime and provided by the controller.	Integer	• None	No	User	CMM No Map

B.1.8 File import attributes

The following table presents common file import attributes supported by CMM:

Attribute	Description	Value	Constraints	Req.	Source	Map type
file_id	The file's ID.	Text	• Must be unique within files imported for the select import type	Yes	User	CMM No Map
file_creation_time	The file's creation date.	Date Note: The current date is the default if you do not specify a value.	• None	No	User	CMM No Map
file_type	The file's type.	One of the following: • C if the file is intraday (CDR) • P if the file is previous-day (PDR)	• Must be provided for bank statement files	No	User	CMM No Map
business_type	The file's business type.	Text	• Must be provided for bank statement files	No	User	CMM No Map

B.2 Export attributes

CMM supports the following export attributes:

- Payment
- Receipt
- Credit advice
- Remittance detail
- Bank statement
- Company general ledger
- Interchange.

B.2.1 Payment export attributes

The following table presents common payment export attributes supported by CMM:

Name	Data type	Map type	Description	Example
Basic attributes				
`payment_currency`	String	CMM No Map	The **ISO** currency code of the transaction.	`USD`
`payment_amount`	Double	CMM No Map	The absolute value of the transaction (in the currency defined in `payment_currency`).	`5000.00`
`bank_account_currency_amount`	Double	CMM No Map	The absolute value of the transaction (in the currency of the transaction's bank account.)	`5000.00`
`is_foreign_currency_txn`	String	CMM No Map	The foreign currency status of the transaction. Acceptable values: • `YES` if the transaction currency is different than the paying entity bank account's currency • `NO` if the transaction currency is the same as the paying entity bank account's currency. Note: This attribute was originally provided to support foreign transactions in CMM 6.5.x.	`NO`

Name	Data type	Map type	Description	Example
foreign_txn_currency	String	CMM No Map	The ISO currency code of the transaction. Note: This attribute was originally provided to support foreign transactions in CMM 6.5.x. As of CMM 7.1.x, this attribute returns the same value as the payment_currency attribute. Therefore, Wallstreet recommends you use the payment_currency attribute rather than this attribute if possible.	USD
foreign_txn_amount	Double	CMM No Map	The absolute value of the transaction (in the currency defined in foreign_txn_currency). Note: This attribute was originally provided to support foreign transactions in CMM 6.5.x. As of CMM 7.1.x, this attribute returns the same value as the payment_amount attribute. Therefore, Wallstreet recommends you use the payment_amount attribute rather than this attribute if possible.	5000.00
originator_country_code	String	CMM No Map	The ISO country code of the transaction's originating entity.	CA
cash_flow_type	String	CMM No Map	The type of the transaction. Acceptable values: • P for payment • R for receipt.	P
transaction_type_code	String	CMM No Map	The cash flow type of the transaction.	COMMP

Name	Data type	Map type	Description	Example
priority_code	Integer	Spec Map	The priority of the transaction. Acceptable values: • 1 for not urgent • 2 for urgent.	2
payment_method	String	CMM Map	The payment method of the transaction.	EFT
payment_primary_delivery_channel	String	Spec Map	The primary delivery channel of the transaction. Acceptable values: • ACH for automated clearing system • WT for wire transfer • CHQ for check. Note: This attribute is similar to but more accurate than payment_method. Wallstreet recommends you use it rather than payment_method when possible. The logic for the derivation is defined in paymentPrimaryDeliveryChannelInstruction.xml.	ACH
clearing_system_type	String	Spec Map	The clearing system type of the transaction. Acceptable values: • LVC for low-value clearing system • HVC for high-value clearing system. Note: This attribute is only applicable to transactions delivered through an automated clearing system. The logic for the derivation is defined in clearingSystemSelectionInstruction.xml.	LVC

Name	Data type	Map type	Description	Example
domestic_crossborder_status	String	Spec Map	The domestic/cross-border status of the transaction. Acceptable values: • DOMESTIC for domestic transactions • CROSS_BORDER for cross-border (international) transactions. Note: The logic for this derivation is defined in domesticInternationalInstruction.xml.	DOMESTIC
payment_residency_flow_status	String	Spec Map	The residency flow status of the transaction. Acceptable values: • RESIDENT TO RESIDENT • RESIDENT TO NON_RESIDENT • NON_RESIDENT TO RESIDENT • NON_RESIDENT TO NON_RESIDENT. Note: This attribute is only applicable to domestic transactions.	RESIDENT TO RESIDENT
foreign_exchange_status	String	Spec Map	The foreign exchange status of the transaction. Acceptable values: • REQUIRED for transactions requiring foreign exchange • NOT_REQUIRED for transactions not required foreign exchange. Note: The logic for this derivation is defined in foreignExchangeStatusInstruction.xml.	NOT_REQUIRED
payment_destination_country_code	String	CMM No Map	The ISO country code of the destination of the transaction.	US

Name	Data type	Map type	Description	Example
transaction_date	Date	CMM No Map	The transaction date of the transaction.	20-Oct-2006
value_date	Date	CMM No Map	The value date of the transaction.	20-Oct-2006
is_valid_transaction	String	CMM No Map	A flag that indicates whether the transaction is valid. Acceptable values: • YES • NO.	YES
is_intercompany_payment	String	CMM No Map	A flag that indicates whether the transaction is intercompany Acceptable values: • YES • [Blank].	YES
payment_description	String	CMM No Map	A relevant description of or other information pertaining to the transaction.	Invoice No. 9876
beneficiary_message	String	CMM No Map	A message to the transaction's beneficiary.	
bank_instruction	String	CMM No Map	Instructions to the transaction's bank.	
payment_details	String	Static Map	A concatenation of payment_description, beneficiary_message, and bank_instruction.	Invoice No. 9876
Paying entity attributes				
payor_long_name	String	CMM No Map	The name of the transaction's paying entity.	Acme USA
payor_local_address_line1	String	CMM No Map	The first address line of the transaction's paying entity.	Suite 100
payor_local_address_line2	String	CMM No Map	The second address line of the transaction's paying entity.	123 Main St.
payor_local_address_line3	String	CMM No Map	The third address line of the transaction's paying entity.	
payor_local_address_line4	String	CMM No Map	The fourth address line of the transaction's paying entity.	

Name	Data type	Map type	Description	Example
payor_local_address	String	CMM No Map	A concatenation of payor_local_address_line1 to payor_local_address_line4.	Suite 100123 Main St.
payor_city_name	String	CMM No Map	The city of the transaction's paying entity.	New York
payor_state_name	String	CMM No Map	The state or province of the transaction's paying entity.	NY
payor_country_code	String	CMM No Map	The ISO country code of the transaction's paying entity.	US
payor_postal_code	String	CMM No Map	The ZIP or postal code of the transaction's paying entity.	12345
payor_phone_number	String	CMM No Map	The telephone number of the paying entity.	
payor_fax_number	String	CMM No Map	The fax number of the paying entity.	
payor_email_account	String	CMM No Map	The e-mail account of the paying entity.	
payor_branch_id	String	CMM No Map	The branch ID of the paying entity. Note: This attribute is only relevant to financial institution counterparties.	
payor_swift_code	String	CMM No Map	The SWIFT code of the paying entity. Note: This attribute is only relevant to financial institution counterparties.	
Originating entity attributes				
originator_long_name	String	CMM No Map	The name of the transaction's originating entity.	Acme Canada
originator_local_address_line_1	String	CMM No Map	The first address line of the transaction's originating entity.	Suite 100
originator_local_address_line_2	String	CMM No Map	The second address line of the transaction's originating entity.	123 Main St.
originator_local_address_line_3	String	CMM No Map	The third address line of the transaction's originating entity.	

Name	Data type	Map type	Description	Example
originator_local_address_line_4	String	CMM No Map	The fourth address line of the transaction's originating entity.	
originator_local_address	String	CMM No Map	A concatenation of originator_local_address_line1 to originator_local_address_line4.	Suite 100123 Main St.
originator_city_name	String	CMM No Map	The city of the transaction's originating entity.	Toronto
originator_state_name	String	CMM No Map	The state or province of the transaction's originating entity.	ON
originator_postal_code	String	CMM No Map	The ZIP or postal code of the transaction's originating entity.	A1B 2C3
Paying entity bank attributes				
payor_bank_swift_code	String	CMM No Map	The SWIFT code of the transaction's paying entity bank.	
payor_bank_branch_id	String	CMM No Map	The branch ID of the transaction's paying entity bank (as assigned by an agency). Note: This ID is also known as a bank sort code.	
payor_bank_branch_qualifier_code	String	CMM No Map	The branch ID type of the transaction's paying entity bank. Note: This attribute is equivalent to the SWIFT clearing code and depends on countries and clearing systems.	
payor_bank_branch_agency_code	String	CMM No Map	The agency that assigned the branch ID to the transaction's paying entity bank.	
payor_bank_long_name	String	CMM No Map	The name of the transaction's paying entity bank.	BankOfComm
payor_bank_address_1	String	CMM No Map	The first address line of the transaction's paying entity bank.	Suite 100
payor_bank_address_2	String	CMM No Map	The second address line of the transaction's paying entity bank.	123 Main St.

Name	Data type	Map type	Description	Example
payor_bank_address_3	String	CMM No Map	The third address line of the transaction's paying entity bank.	
payor_bank_address_4	String	CMM No Map	The fourth address line of the transaction's paying entity bank.	
payor_bank_name_and_adddress	String	CMM No Map	A concatenation of payor_bank_long_name and payor_bank_address_1 to payor_bank_address_4.	Suite 100123 Main St.
payor_bank_local_address	String	CMM No Map	A concatenation of payor_bank_address_1 to payor_bank_address_4.	Suite 100123 Main St.
payor_bank_city_name	String	CMM No Map	The city of the transaction's paying entity bank.	Boston
payor_bank_state_name	String	CMM No Map	The state or province of the transaction's paying entity bank.	MA
payor_bank_country_code	String	CMM No Map	The ISO country code of the transaction's paying entity bank.	US
payor_bank_postal_code	String	CMM No Map	The ZIP or postal code of the transaction's paying entity bank.	12345
payor_bank_national_currency_code	String	CMM Map	The national currency of the transaction's paying entity bank.	USD
Paying entity bank account attributes				
payor_bank_account_number	String	CMM No Map	The **BBAN** of the transaction's paying entity bank account.	12345678
payor_bank_account_iban	String	CMM No Map	The **IBAN** of the transaction's paying entity bank account.	US9012345678
payor_bank_account_name	String	CMM No Map	The name of the paying entity bank account.	AcmeUSBA
payor_bank_account_type	String	CMM No Map	The type of the paying entity bank account.	General
payor_bank_account_currency_code	String	CMM No Map	The currency of the paying entity bank account.	USD

Name	Data type	Map type	Description	Example
payor_bank_account_resident_status	String	Spec Map	The resident status of the transaction's paying entity bank account. Acceptable values: • RESIDENT if the paying entity bank account is a resident account. • NON_RESIDENT if the paying entity bank account is a non-resident account. Note: The logic for this derivation is defined in bankaccountResidencyStatusInstruction.xml.	RESIDENT
payor_bank_acctount_clearing_number	String	CMM No Map	A unique identifier assigned to the paying entity bank account by a clearing system (for example, a Giro number).	
Counterparty attributes				
payee_long_name	String	CMM No Map	The name of the transaction's counterparty.	Smith Company
payee_local_address_line_1	String	CMM No Map	The first address line of the transaction's counterparty.	Suite 100
payee_local_address_line_2	String	CMM No Map	The second address line of the transaction's counterparty.	123 Main St.
payee_local_address_line_3	String	CMM No Map	The third address line of the transaction's counterparty.	
payee_local_address_line_4	String	CMM No Map	The fourth address line of the transaction's counterparty.	
payee_name_and_address	String	CMM No Map	A concatenation of payee_long_name and payee_local_address_line_1 to payee_local_address_line_4.	Smith CompanySuite 100123 Main St.
payee_local_address	String	CMM No Map	A concatenation of payee_local_address_line_1 to payee_local_address_line_4.	Suite 100123 Main St.

Name	Data type	Map type	Description	Example
payee_city_name	String	CMM No Map	The city of the transaction's counterparty.	Chicago
payee_state_name	String	CMM No Map	The state or province of the transaction's counterparty.	IL
payee_country_code	String	CMM No Map	The ISO country code of the transaction's counterparty.	US
payee_country_name	String	CMM No Map	The country name of the transaction's counterparty.	United States
payee_postal_code	String	CMM No Map	The ZIP or postal code of the transaction's counterparty.	12345
payee_phone_number	String	CMM No Map	The telephone number of the counterparty.	234-567-8901
payee_branch_id	String	CMM No Map	The branch ID of the counterparty. Note: This attribute is only relevant to financial institution counterparties.	
payee_swift_code	String	CMM No Map	The SWIFT code of the counterparty. Note: This attribute is only relevant to financial institution counterparties.	
Counterparty bank attributes				
payee_bank_swift_code	String	CMM No Map	The SWIFT code of the transaction's counterparty bank.	
payee_bank_branch_id	String	CMM No Map	The branch ID of the transaction's counterparty bank (as assigned by an agency). Note: This ID is also known as a bank sort code.	
payee_bank_branch_qualifier_code	String	CMM No Map	The branch ID type of the transaction's counterparty bank. Note: This attribute is equivalent to the SWIFT clearing code and depends on countries and clearing systems.	

Name	Data type	Map type	Description	Example
payee_bank_branch_agency_code	String	CMM No Map	The agency that assigned the branch ID to the transaction's counterparty bank.	
payee_bank_long_name	String	CMM No Map	The name of the transaction's counterparty bank.	First National Bank
payee_bank_address_1	String	CMM No Map	The first address line of the transaction's counterparty bank.	Suite 100
payee_bank_address_2	String	CMM No Map	The second address line of the transaction's counterparty bank.	123 Main St.
payee_bank_address_3	String	CMM No Map	The third address line of the transaction's counterparty bank.	
payee_bank_address_4	String	CMM No Map	The fourth address line of the transaction's counterparty bank.	
payee_bank_name_and_address	String	CMM No Map	A concatenation of payee_bank_long_name and payee_bank_address_1 to payee_bank_address_4.	First National BankSuite 100123 Main St.
payee_bank_local_address	String	CMM No Map	A concatenation of payee_bank_address_1 to payee_bank_address_4.	Suite 100123 Main St.
payee_bank_city_name	String	CMM No Map	The city of the transaction's counterparty bank.	Chicago
payee_bank_state_name	String	CMM No Map	The state or province of the transaction's counterparty bank.	IL
payee_bank_country_code	String	CMM No Map	The ISO country code of the transaction's counterparty bank.	US
payee_bank_postal_code	String	CMM No Map	The ZIP or postal code of the transaction's counterparty bank.	12345
Counterparty bank account attributes				

Name	Data type	Map type	Description	Example
counter_party_bank_account_requiremnt	String	Spec Map	A flag indicating if counterparty bank account information is required. Acceptable values: • REQUIRED for transactions requiring a counterparty bank account • NOT_REQUIRED for transactions not requiring a counterparty bank account.	REQUIRED
payee_bank_account_number	String	CMM No Map	The primary bank account number of the transaction's counterparty bank account. Note: This can be either BBAN or IBAN.	12345678
payee_bank_account_bban	String	CMM No Map	The BBAN of the transaction's counterparty bank account.	12345678
payee_bank_account_iban	String	CMM No Map	The IBAN of the transaction's counterparty bank account.	US9012345678
payee_bank_account_name	String	CMM No Map	The name of the counterparty bank account.	AcmeUSBA
payee_bank_account_type	String	CMM No Map	The type of the counterparty bank account.	General
payee_bank_account_currency_code	String	CMM No Map	The currency of the counterparty bank account.	USD

Name	Data type	Map type	Description	Example
payee_bank_account_resident_status	String	Spec Map	The resident status of the transaction's counterparty bank account. Acceptable values: • RESIDENT if the counterparty bank account is a resident account. • NON_RESIDENT if the counterparty bank account is a non-resident account. Note: The logic for this derivation is defined in bankaccountResidencyStatusInstruction.xml.	RESIDENT
payee_bank_acctount_clearing_number	String	CMM No Map	A unique identifier assigned to the counterparty bank account by a clearing system (for example, a Giro number).	
Intermediary bank attributes				
intermediary_bank_swift_code	String	CMM No Map	The SWIFT code of the transaction's intermediary bank.	
intermediary_bank_branch_id	String	CMM No Map	The branch ID of the transaction's intermediary bank (as assigned by an agency). Note: This ID is also known as a bank sort code.	
intermediary_bank_branch_qualifier_code	String	CMM No Map	The branch ID type of the transaction's intermediary bank. Note: This attribute is equivalent to the SWIFT clearing code and depends on countries and clearing systems.	
intermediary_bank_branch_agency_code	String	CMM No Map	The agency that assigned the branch ID to the transaction's intermediary bank.	
intermediary_bank_long_name	String	CMM No Map	The name of the transaction's intermediary bank.	TransWorld Bank

Name	Data type	Map type	Description	Example
intermediary_bank_address_1	String	CMM No Map	The first address line of the transaction's intermediary bank.	Suite 100
intermediary_bank_address_2	String	CMM No Map	The second address line of the transaction's intermediary bank.	123 Main St.
intermediary_bank_address_3	String	CMM No Map	The third address line of the transaction's intermediary bank.	
intermediary_bank_address_4	String	CMM No Map	The fourth address line of the transaction's intermediary bank.	
intermediary_bank_name_and_address	String	CMM No Map	A concatenation of intermediary_bank_long_name and intermediary_bank_address_1 to intermediary_bank_address_4.	TransWorld BankSuite 100123 Main St.
intermediary_bank_local_address	String	CMM No Map	A concatenation of intermediary_bank_address_1 to intermediary_bank_address_4.	Suite 100123 Main St.
intermediary_bank_city_name	String	CMM No Map	The city of the transaction's intermediary bank.	Atlanta
intermediary_bank_state_name	String	CMM No Map	The state or province of the transaction's intermediary bank.	GA
intermediary_bank_country_code	String	CMM No Map	The ISO country code of the transaction's intermediary bank.	US
intermediary_bank_postal_code	String	CMM No Map	The ZIP or postal code of the transaction's intermediary bank.	12345
Intermediary bank account attributes				
intermediary_bank_account_number	String	CMM No Map	The primary bank account number of the intermediary bank account.	
intermediary_bank_account_name	String	CMM No Map	The name of the intermediary bank account.	
Second intermediary bank attributes				
second_intermediary_bank_swift_code	String	CMM No Map	The SWIFT code of the transaction's intermediary bank.	

Name	Data type	Map type	Description	Example
second_intermediary_bank_branch_id	String	CMM No Map	The branch ID of the transaction's intermediary bank (as assigned by an agency). Note: This ID is also known as a bank sort code.	
second_intermediary_bank_long_name	String	CMM No Map	The name of the transaction's intermediary bank.	TransWorld Bank
second_intermediary_bank_address_1	String	CMM No Map	The first address line of the transaction's intermediary bank.	Suite 100
second_intermediary_bank_address_2	String	CMM No Map	The second address line of the transaction's intermediary bank.	123 Main St.
second_intermediary_bank_address_3	String	CMM No Map	The third address line of the transaction's intermediary bank.	
second_intermediary_bank_address_4	String	CMM No Map	The fourth address line of the transaction's intermediary bank.	
second_intermediary_bank_name_and_address	String	CMM No Map	A concatenation of intermediary_bank_long_name and intermediary_bank_address_1 to intermediary_bank_address_4.	TransWorld BankSuite 100123 Main St.
second_intermediary_bank_local_address	String	CMM No Map	A concatenation of intermediary_bank_address_1 to intermediary_bank_address_4.	Suite 100123 Main St.
second_intermediary_bank_city_name	String	CMM No Map	The city of the transaction's intermediary bank.	Atlanta
second_intermediary_bank_state_name	String	CMM No Map	The state or province of the transaction's intermediary bank.	GA
second_intermediary_bank_country_code	String	CMM No Map	The ISO country code of the transaction's intermediary bank.	US
second_intermediary_bank_postal_code	String	CMM No Map	The ZIP or postal code of the transaction's intermediary bank.	12345
Second intermediary bank account attributes				
second_intermediary_bank_account_number	String	CMM No Map	The primary bank account number of the intermediary bank account.	

Name	Data type	Map type	Description	Example
second_intermediary_bank_account_name	String	CMM No Map	The name of the intermediary bank account.	
Tax attributes				
tax_payment_type_code	String	CMM No Map	A code identifying the type of tax being paid.	
tax_payment_subtype_code	String	CMM No Map	A code identifying the subtype of tax being paid.	
tax_payer_id	String	CMM No Map	A number assigned to a purchaser by a taxing jurisdiction. Note: This ID is also known as a tax exemption number or certificate number.	
tax_payer_verification_id	String	CMM No Map	An ID the receiver can use to confirm the taxpayer's identity.	
Aggregation attributes				
is_aggregate_payment	String	Derived Map	A flag that indicates whether the transaction is an aggregate. Acceptable values: • YES • NO.	YES
aggregate_status	Boolean	Derived Map	The aggregation status of the transaction. Acceptable values: • true • false.	true
payment_aggregation_type	String	CMM No Map	The aggregation type of the transaction. Acceptable values: • SINGLE_CREDIT • MULTI_CREDIT • NO_AGGREGATION.	SINGLE_CREDIT
number_of_component_payments	Integer	CMM No Map	The total number of component payments contained in the aggregate transaction.	50
component_payments	Vector	CMM No Map	A list of the component payment objects of the transaction.	
Other attributes				

Name	Data type	Map type	Description	Example
bank_reference_number	String	CMM No Map	The sending institution's reference number for the transaction.	
central_bank_reporting_reason_code	String	CMM No Map	The reason code of the transaction supplied by the central bank. Note: Wallstreet recommends not to use this attribute as the code mapping is not fully in place yet.	
cheque_number	String	CMM No Map	The check number of the transaction.	
customer_batch_reference_id	String	CMM No Map	The customer batch reference ID of the transaction from the ERP system.	
customer_reference_id	String	CMM No Map	The customer's unique ID for the transaction.	123456
customer_txn_reference_id	String	CMM No Map	The customer transaction reference ID from the ERP system.	
financial_charge_allocation	String	CMM No Map	The financial charge allocation of the transaction. Note: This attribute defaults to 1 (both payor and payee share charges).	1
interface_type	String	Static No Map	The interface type of the transaction.	payment
number_of_remittance_details	Integer	CMM No Map	The number of remittance details for the transaction.	50
originator_identification_number	String	CMM No Map	The originating party identification number of the transaction.	
payee_bank_company_identifier	String	CMM No Map	The payee bank company identifier of the transaction.	
payee_identification_number	String	CMM No Map	The payee identification number of the transaction.	
payment_contract_number	String	CMM No Map	The payment contract number of the transaction.	
payor_bank_company_identifier	String	CMM No Map	The payor bank company identifier of the transaction.	

Name	Data type	Map type	Description	Example
payor_company_identifier	String	CMM No Map	The payor identification number of the transaction.	
repetitive_code	String	CMM No Map	A bank-defined code that uniquely identifies the static attributes of the transaction.	
retrieval_location_address	String	CMM No Map	The address of the location where the transaction can be picked up. Note: This attribute is used in eastern Europe.	
retrieval_location_id	String	CMM No Map	The ID of the location where the transaction can be picked up. Note: This attribute is used in eastern Europe.	
retrieval_location_name	String	CMM No Map	The name of the location where the transaction can be picked up. Note: This attribute is used in eastern Europe.	
second_company_identifier	String	CMM No Map	The payor secondary identification number of the transaction.	
source_batch_id	String	CMM No Map	The source batch ID of the transaction.	

B.2.2 Receipt export attributes

The following table presents common receipt export attributes supported by CMM:

Name	Data type	Map type	Description	Example
Basic attributes				
currency	String	CMM No Map	The ISO currency code of the transaction.	USD
amount	Double	CMM No Map	The absolute value of the transaction (in the currency defined in payment_currency).	5000.00
cash_flow_type	String	CMM No Map	The cash flow type of the transaction.	COMMP

Name	Data type	Map type	Description	Example
transaction_type_code	String	CMM No Map	The customer-definable transaction type code. Note: This attribute maps to the Transaction Type Code additional attribute.	123
payment_method	String	CMM Map	The payment method of the transaction.	EFT
txn_date	Date	CMM No Map	The transaction date of the transaction.	20-Oct-2006
value_date	Date	CMM No Map	The value date of the transaction.	20-Oct-2006
description	String	CMM No Map	A relevant description of or other information pertaining to the transaction.	Invoice No. 9876
beneficiary_message	String	CMM No Map	A message to the transaction's beneficiary.	
bank_instruction	String	CMM No Map	Instructions to the transaction's bank.	
Receiving entity attributes				
party_company_identifier	String	CMM No Map	The identification number of the transaction's receiving entity.	
party_second_company_identifier	String	CMM No Map	The secondary identification number of the transaction's receiving entity.	
party_short_name	String	CMM No Map	The short name of the transaction's receiving entity.	Acme USA
party_long_name	String	CMM No Map	The long name of the transaction's receiving entity.	Acme USA Inc.
party_address_line1	String	CMM No Map	The first address line of the transaction's receiving entity.	Suite 100
party_address_line2	String	CMM No Map	The second address line of the transaction's receiving entity.	123 Main St.
party_address_line3	String	CMM No Map	The third address line of the transaction's receiving entity.	
party_address_line4	String	CMM No Map	The fourth address line of the transaction's receiving entity.	
party_city_name	String	CMM No Map	The city of the transaction's receiving entity.	New York
party_state_name	String	CMM No Map	The state or province of the transaction's receiving entity.	NY
party_country_code	String	CMM No Map	The ISO country code of the transaction's receiving entity.	US

Name	Data type	Map type	Description	Example
party_postal_code	String	CMM No Map	The ZIP or postal code of the transaction's receiving entity.	12345
Originating entity attributes				
originator_companyidentifier	String	CMM No Map	The identifier of the transaction's originating entity.	
originator_short_name	String	CMM No Map	The short name of the transaction's originating entity.	Acme Canada
originator_long_name	String	CMM No Map	The long name of the transaction's originating entity.	Acme Canada Inc.
originator_address_line_1	String	CMM No Map	The first address line of the transaction's originating entity.	Suite 100
originator_address_line_2	String	CMM No Map	The second address line of the transaction's originating entity.	123 Main St.
originator_address_line_3	String	CMM No Map	The third address line of the transaction's originating entity.	
originator_address_line_4	String	CMM No Map	The fourth address line of the transaction's originating entity.	
originator_city_name	String	CMM No Map	The city of the transaction's originating entity.	Toronto
originator_state_name	String	CMM No Map	The state or province of the transaction's originating entity.	ON
originator_country_code	String	CMM No Map	The ISO country code of the transaction's originating entity.	CA
originator_postal_code	String	CMM No Map	The ZIP or postal code of the transaction's originating entity.	A1B 2C3
Receiving entity bank attributes				
party_bank_swift_code	String	CMM No Map	The SWIFT code of the transaction's receiving entity bank.	
party_bank_sort_code	String	CMM No Map	The sort code of the transaction's receiving entity bank (as assigned by an agency).	
party_bank_branch_qualifier_code	String	CMM No Map	The branch ID type of the transaction's receiving entity bank. Note: This attribute is equivalent to the SWIFT clearing code and depends on countries and clearing systems.	
party_bank_branch_agency_code	String	CMM No Map	The agency that assigned the branch ID to the transaction's receiving entity bank.	

Name	Data type	Map type	Description	Example
party_bank_long_name	String	CMM No Map	The long name of the transaction's receiving entity bank.	BankOfComm
party_bank_short_name	String	CMM No Map	The short name of the transaction's receiving entity bank.	Bank of Commerce
party_bank_address_1	String	CMM No Map	The first address line of the transaction's receiving entity bank.	Suite 100
party_bank_address_2	String	CMM No Map	The second address line of the transaction's receiving entity bank.	123 Main St.
party_bank_address_3	String	CMM No Map	The third address line of the transaction's receiving entity bank.	
party_bank_address_4	String	CMM No Map	The fourth address line of the transaction's receiving entity bank.	
party_bank_city_name	String	CMM No Map	The city of the transaction's receiving entity bank.	Boston
party_bank_state_name	String	CMM No Map	The state or province of the transaction's receiving entity bank.	MA
party_bank_country_code	String	CMM No Map	The ISO country code of the transaction's receiving entity bank.	US
party_bank_postal_code	String	CMM No Map	The ZIP or postal code of the transaction's receiving entity bank.	12345
Receiving entity bank account attributes				
party_bank_account_number	String	CMM No Map	Thef the transaction's receiving entity bank account.	12345678
party_bank_account_id	String	CMM No Map	The ID of the receiving entity bank account.	AcmeUSBA
party_bank_account_name	String	CMM No Map	The name of the receiving entity bank account.	AcmeUSBA
Counterparty attributes				
cpty_long_name	String	CMM No Map	The short name of the transaction's counterparty.	Smith Company
cpty_short_name	String	CMM No Map	The long name of the transaction's counterparty.	Smith Company Ltd.
cpty_local_address_line_1	String	CMM No Map	The first address line of the transaction's counterparty.	Suite 100
cpty_local_address_line_2	String	CMM No Map	The second address line of the transaction's counterparty.	123 Main St.

Name	Data type	Map type	Description	Example
cpty_local_address_line_3	String	CMM No Map	The third address line of the transaction's counterparty.	
cpty_local_address_line_4	String	CMM No Map	The fourth address line of the transaction's counterparty.	
cpty_city_name	String	CMM No Map	The city of the transaction's counterparty.	Chicago
cpty_state_name	String	CMM No Map	The state or province of the transaction's counterparty.	IL
cpty_country_code	String	CMM No Map	The ISO country code of the transaction's counterparty.	US
cpty_postal_code	String	CMM No Map	The ZIP or postal code of the transaction's counterparty.	12345
cpty_phone_number	String	CMM No Map	The telephone number of the counterparty.	234-567-8901
Counterparty bank attributes				
cpty_bank_swift_code	String	CMM No Map	The SWIFT code of the transaction's counterparty bank.	
cpty_bank_sort_code	String	CMM No Map	The sort code of the transaction's counterparty bank (as assigned by an agency).	
cpty_bank_branch_qualifier_code	String	CMM No Map	The branch ID type of the transaction's counterparty bank. Note: This attribute is equivalent to the SWIFT clearing code and depends on countries and clearing systems.	
cpty_bank_branch_agency_code	String	CMM No Map	The agency that assigned the branch ID to the transaction's counterparty bank.	
cpty_bank_long_name	String	CMM No Map	The name of the transaction's counterparty bank.	First National Bank
cpty_bank_address_1	String	CMM No Map	The first address line of the transaction's counterparty bank.	Suite 100
cpty_bank_address_2	String	CMM No Map	The second address line of the transaction's counterparty bank.	123 Main St.
cpty_bank_address_3	String	CMM No Map	The third address line of the transaction's counterparty bank.	
cpty_bank_address_4	String	CMM No Map	The fourth address line of the transaction's counterparty bank.	

Name	Data type	Map type	Description	Example
cpty_bank_local_address	String	CMM No Map	A concatenation of payee_bank_address_1 to payee_bank_address_4.	Suite 100123 Main St.
cpty_bank_city_name	String	CMM No Map	The city of the transaction's counterparty bank.	Chicago
cpty_bank_state_name	String	CMM No Map	The state or province of the transaction's counterparty bank.	IL
cpty_bank_country_code	String	CMM No Map	The ISO country code of the transaction's counterparty bank.	US
cpty_bank_postal_code	String	CMM No Map	The ZIP or postal code of the transaction's counterparty bank.	12345
Counterparty bank account attributes				
ctpy_bank_account_number	String	CMM No Map	The primary bank account number of the transaction's counterparty bank account. Note: This can be either BBAN or IBAN.	12345678
ctpy_bank_account_name	String	CMM No Map	The name of the counterparty bank account.	AcmeUSBA
ctpy_bank_account_currency_code	String	CMM No Map	The currency of the counterparty bank account.	USD
Intermediary bank attributes				
intermediary_bank_swift_code	String	CMM No Map	The SWIFT code of the transaction's intermediary bank.	
intermediary_bank_sort_code	String	CMM No Map	The sort code of the transaction's intermediary bank (as assigned by an agency).	
intermediary_bank_branch_qualifier_code	String	CMM No Map	The branch ID type of the transaction's intermediary bank. Note: This attribute is equivalent to the SWIFT clearing code and depends on countries and clearing systems.	
intermediary_bank_branch_agency_code	String	CMM No Map	The agency that assigned the branch ID to the transaction's intermediary bank.	
intermediary_bank_short_name	String	CMM No Map	The short name of the transaction's intermediary bank.	TransWorld Bank
intermediary_bank_long_name	String	CMM No Map	The long name of the transaction's intermediary bank.	TransWorld Bank Corp.

Name	Data type	Map type	Description	Example
intermediary_bank_address_1	String	CMM No Map	The first address line of the transaction's intermediary bank.	Suite 100
intermediary_bank_address_2	String	CMM No Map	The second address line of the transaction's intermediary bank.	123 Main St.
intermediary_bank_address_3	String	CMM No Map	The third address line of the transaction's intermediary bank.	
intermediary_bank_address_4	String	CMM No Map	The fourth address line of the transaction's intermediary bank.	
intermediary_bank_city_name	String	CMM No Map	The city of the transaction's intermediary bank.	Atlanta
intermediary_bank_state_name	String	CMM No Map	The state or province of the transaction's intermediary bank.	GA
intermediary_bank_country_code	String	CMM No Map	The ISO country code of the transaction's intermediary bank.	US
intermediary_bank_postal_code	String	CMM No Map	The ZIP or postal code of the transaction's intermediary bank.	12345
Intermediary bank account attributes				
intermediary_bank_account_number	String	CMM No Map	The primary bank account number of the intermediary bank account.	
intermediary_bank_account_name	String	CMM No Map	The name of the intermediary bank account.	
Other attributes				
bank_preauthorization_id	String	CMM No Map	A unique authorization ID for the transaction. Note: This is primarily used for direct debits.	
bank_preauthorization_status	String	CMM No Map	The authorization status for the transaction. Note: This is primarily used for direct debits.	
bank_reference_number	String	CMM No Map	The sending institution's reference number for the transaction.	
cash_record_id	Integer	CMM No Map	The CMM cash record ID for the transaction.	123456
customer_batch_reference_id	String	CMM No Map	The customer batch reference ID of the transaction from the ERP system.	
customer_reference_id	String	CMM No Map	The customer's unique ID for the transaction.	123456

Name	Data type	Map type	Description	Example
customer_txn_reference_id	String	CMM No Map	The customer transaction reference ID from the ERP system.	
direct_debit_contract_number	String	CMM No Map	The contract number of the direct debit transaction.	
interface_type	String	Static No Map	The interface type of the transaction.	payment
lcr_contract_number	String	CMM No Map	The contract number of the letter of credit transaction.	
reconciliation_id	String	CMM No Map	The reconciliation ID of the truncation.	
repetitive_code	String	CMM No Map	A bank-defined code that uniquely identifies the static attributes of the transaction.	

B.2.3 Credit advice export attributes

The following table presents common credit advice export attributes supported by CMM:

Name	Data type	Map type	Description	Example
Basic attributes				
currency	String	CMMNo Map	The **ISO** currency code of the transaction.	USD
amount	Double	CMMNo Map	The absolute value of the transaction (in the currency defined in payment_currency).	5000.00
settlement_currency	String	CMMNo Map	The settlement ISO currency code of the transaction.	USD
settlement_amount	Double	CMMNo Map	The absolute value of the transaction (in the currency defined in settlement_currency).	5000.00
txn_date	Date	CMMNo Map	The transaction date of the transaction.	20-Oct-2006
value_date	Date	CMMNo Map	The value date of the transaction.	20-Oct-2006
message_issue_date	Date	CMMNo Map	The creation date of the transaction.	20-Oct-2006
Entity attributes				
paror_short_name	String	CMMNo Map	The short name of the transaction's entity.	Acme USA
paror_id	String	CMMNo Map	The ID of the transaction's entity.	AcmeUS
Counterparty attributes				

Name	Data type	Map type	Description	Example
payee_short_name	String	CMMNo Map	The short name of the transaction's counterparty.	Smith Company
payee_id	String	CMMNo Map	The ID of the transaction's counterparty.	SmithCo
Other attributes				
cash_record_id	Integer	CMMNo Map	The CMM cash record ID for the transaction.	123456
customer_reference_id	String	CMMNo Map	The customer's unique ID for the transaction.	123456
number_of_remittance_details	Integer	CMMNo Map	The number of remittance details for the transaction.	50

B.2.4 Remittance detail export attributes

The following table presents common remittance detail export attributes supported by CMM:

Name	Data type	Map type	Description	Example
remittance_invoice_number	String	CMMNo Map	The invoice number of the remittance detail.	
remittance_date	String	CMMNo Map	The invoice date of the remittance detail.	
remittance_amount_payable	String	CMMNo Map	The amount payable as stated on the invoice.	
remittance_amount_credit_note	String	CMMNo Map	The amount of the credit note.	
remittance_amount_discounted	String	CMMNo Map	The amount of discount.	
remittance_amount_remitted	String	CMMNo Map	The amount actually paid (payable-credit-discount).	
remittance_currency_code	String	CMMNo Map	The **ISO** currency code of the remittance detail. Note: If you do not provide this attribute, CMM uses the parent transaction's currency code instead.	
remittance_detail_message_type	String	Spec map	The message type of the remittance detail. Note: The value of this attribute is an internal code only and is not mapped to an external code. Therefore, use this attribute with care.	
remittance_retail_reference_type_code	String	CMMNo Map	The reference type code of the remittance detail.	

Name	Data type	Map type	Description	Example
remittance_detail_reference	String	CMMNo Map	The reference of the remittance detail.	
remittance_detail_id	Integer	CMMNo Map	A unique, sequential, numeric ID assigned to the remittance detail.	
remittance_detail_source_reference_id	Integer	CMMNo Map	The cash record ID of the remittance detail's transaction.	
remittance_detail_group_id	Integer	CMMNo Map	The group ID of the remittance detail.	

B.2.5 Bank statement export attributes

The following table presents common bank statement export attributes supported by CMM:

Name	Data type	Map type	Description	Example
Basic attributes				
statement_statement_number	String		The number of the bank statement.	
statement_start_date	Date		The starting date of the bank statement.	
statement_end_date	Date		The ending date of the bank statement.	
statement_is_internal	Boolean		A flag that indicates whether the bank statement is internal.	
statement_is_closable	Boolean		A flag that indicates whether the bank statement can be closed.	
statement_is_external_stmt_period	Boolean		A flag that indicates whether the bank statement is external (in other words, the bank statement's period is `External`).	
statement_is_valid_transaction	Boolean		A flag that indicates whether the bank statement is valid.	
Bank account attributes				
bankaccount_bank_sort_code	String		The sort code of the bank account's bank.	
bankaccount_number	String		The bank account number of the bank account.	
bankaccount_type_id	String		The type of the bank account.	
bankaccount_status_code	String		The status of the bank account. Acceptable values: • Open • Closed • Ceased to exist • Incomplete.	

Name	Data type	Map type	Description	Example
bankaccount_name	String		The name of the bank account.	
bankaccount_address1	String		The first address line of the bank account.	
bankaccount_address2	String		The second address line of the bank account.	
bankaccount_address3	String		The third address line of the bank account.	
bankaccount_address4	String		The fourth address line of the bank account.	
bankaccount_city	String		The city of the bank account.	
bankaccount_state	String		The state or province of the bank account.	
bankaccount_country_code	String		The **ISO** country code of the bank account.	
bankaccount_postal_code	String		The ZIP or postal code of the bank account.	
bankaccount_phone_number	String		The telephone number of the bank account.	
bankaccount_fax_number	String		The fax number of the bank account.	
bankaccount_email	String		The e-mail address of the bank account.	
bankaccount_currency_code	String		The ISO currency code of the bank account.	
Entity attributes				
entity_swift_code			The SWIFT code of the entity.	
entity_aba_code			The ABA code of the entity.	
entity_party_status			The status of the entity.	
entity_parent_party_id			The ID of the entity's parent.	
entity_short_name			The short name of the entity.	
entity_long_name			The long name of the entity.	
entity_address1			The first address line of the entity.	
entity_address2			The second address line of the entity.	
entity_address3			The third address line of the entity.	
entity_address4			The fourth address line of the entity.	
entity_city			The city of the entity.	
entity_state			The state or province of the entity.	
entity_country_code			The ISO country code of the entity.	

Name	Data type	Map type	Description	Example
entity_region_id			The region of the entity.	
entity_postal_code			The postal code of the entity.	
entity_phone_number			The telephone number of the entity.	
entity_fax_number			The fax number of the entity.	
entity_email			The e-mail address of the entity.	
entity_functional_currency_code			The ISO currency code of the entity.	
entity_country_of_incorporation			The country of incorporation of the entity.	
Bank attributes				
bank_swift_code			The SWIFT code of the bank.	
bank_aba_code			The ABA code of the bank.	
bank_party_status			The status of the bank.	
bank_parent_party_id			The ID of the bank's parent.	
bank_short_name			The short name of the bank.	
bank_long_name			The long name of the bank.	
bank_address1			The first address line of the bank.	
bank_address2			The second address line of the bank.	
bank_address3			The third address line of the bank.	
bank_address4			The fourth address line of the bank.	
bank_city			The city of the bank.	
bank_state			The state or province of the bank.	
bank_country_code			The ISO country code of the bank.	
bank_region_id			The region of the bank.	
bank_postal_code			The postal code of the bank.	
bank_phone_number			The telephone number of the bank.	
bank_fax_number			The fax number of the bank.	
bank_email			The e-mail address of the bank.	
bank_functional_currency_code			The ISO currency code of the bank.	
bank_country_of_incorporation			The country of incorporation of the bank.	
Bank balance attributes				
balance_currency_code	String	CMM No Map	The ISO currency code of the bank balance.	USD

Name	Data type	Map type	Description	Example
balance_actual_amount	Double	CMM No Map	The actual value of the balance (in the currency defined in payment_currency).	5000.00
balance_fcast_amount	Double		The forecast value of the balance (in the currency defined in payment_currency).	
balance_date	Date	CMM No Map	The date of the bank balance.	
balance_posting_date	Date		The posting date of the bank balance.	
balance_bank_acct_name	String	CMM Map	The name of the bank balance's bank account.	
balance_bank_acct_number	String	CMM Map	The bank account number of the bank balance's bank account.	
balance_external_balance_type	String	CMM Map	The external type of the bank balance.	
balance_txn_sub_type_id			The transaction subtype of the bank balance.	
Bank transaction attributes				
txn_currency_code	String	CMM No Map	The ISO currency code of the bank transaction.	USD
txn_amount	Double	CMM No Map	The absolute value of the bank transaction (in the currency defined in currency_code).	5000.00
txn_fx_rate	String		The foreign exchange rate of the bank transaction.	
txn_type_code	String	CMM Map	The type of the bank transaction. Acceptable values: • P for payment • R for receipt.	P
txn_bank_txn_status_code	String		The status of the bank transaction.	
txn_change_history	String		The change history of the bank transaction.	
txn_funds_type_code	String	CMM Map	The cash flow type of the bank transaction. Note: This must be a valid cash flow type ID.	COMMP
txn_external_txn_sub_type	String	CMM Map	The counterparty file code of the bank transaction. Note: This must be a valid counterparty file code.	1234
txn_txn_date	Date	CMM No Map	The transaction date of the bank transaction.	20-Oct-2006
txn_value_date	Date	CMM No Map	The value date of the bank transaction.	20-Oct-2006

Name	Data type	Map type	Description	Example
txn_description	String	CMM No Map	A relevant description of or other information pertaining to the bank transaction.	Invoice No. 9876
txn_bank_acct_number	String	CMM Map	The bank account number of the bank transaction's entity bank account.	12345678
txn_bank_acct_name	String	CMM Map	The name of the bank transaction's entity bank account	AcmeUSBA
txn_originating_bank_id	String	CMM No Map	The ID of the bank transaction's entity bank.	BankOfComm
txn_cpty_bank_account_number	String	CMM No Map	The bank account number of the bank transaction's counterparty bank account.	87654321
txn_customer_reference_id	String	CMM No Map	The originating system's unique ID for the bank transaction. Note: If you do not specify a value in this attribute, CMM cannot automatically reconcile the bank transaction.	123456
txn_customer_account_number	String		The customer account number of the bank transaction.	
txn_customer_reference_number	String		The customer reference number of the bank transaction.	
txn_bank_reference_number	String	CMM No Map	The bank's unique ID for the bank transaction.	654321
txn_bank_instructions	String		Instructions to the bank transaction's bank.	
txn_cpty_message	String		A message to the bank transaction's counterparty.	
txn_cheque_number	String	CMM No Map	The check number of the bank transaction.	561866166
txn_file_type	String		The file type of the bank transaction.	
txn_file_sub_type	String		The file subtype of the bank transaction.	
txn_is_internal	Integer		A flag that indicates whether the bank transaction is internal. Acceptable values: • 0 for no • 1 for yes.	
txn_reconciled_ind	Integer		A flag that indicates whether the bank transaction is reconciled. Acceptable values: • 0 for no • 1 for yes.	
txn_aggregate_ind	Integer		A flag that indicates whether the bank transaction is an aggregate.	
txn_statement_number	String		The number of the bank transaction's bank statement.	

Name	Data type	Map type	Description	Example
txn_statement_sequence_number	String		The sequence number of the bank transaction's bank statement.	
txn_financial_cash_flow_types	String	CMM No Map	Financial cash flow types coming from TRM	Interest; Discount Premium; Issuer Fee
txn_external_references	String	CMM No Map	TRM transaction numbers	3412;3413

B.2.6 Company general ledger export attributes

The following table presents common company general ledger export attributes supported by CMM:

Name	Data type	Map type	Description	Example
accounting_period_number	Integer	CMM No Map	The number of the transaction's accounting period.	64691
bank_account_number	String	CMM No Map	The bank account number of the transaction's entity bank account.	643-632327
base_currency	String	CMM No Map	The base **ISO** currency code of the transaction.	USD
base_currency_amount	Double	CMM No Map	The value of the transaction (in the currency defined in base_currency).	4367.98
cpty_id	String	CMM No Map	The ID of the transaction's counterparty.	SmithCo
customer_reference_id	String	CMM No Map	The customer reference ID of the transaction.	
description	String	CMM No Map	A description of the transaction.	
gl_account_code	String	CMM No Map	The company general ledger account to which the transaction is mapped.	T123
gl_group_id	Integer	CMM No Map	The general ledger group ID of the transaction.	10
instrument_category	String	CMM No Map	The instrument category of the transaction.	
instrument_type_code	String	CMM No Map	The instrument type of the transaction.	
intercomapny_txn	String	CMM No Map	A toggle indicating if the transaction is intercompany or not.	

Name	Data type	Map type	Description	Example
originating_system_code	String	CMM No Map	The originating system of the transaction. Acceptable values: • AP for AP imported transaction • AR for AR import imported transaction • DD for direct debit imported transaction • PMTPROC for payment processing (routing) created transaction • Manual for manually entered transaction • Man-OneOff for manually entered one-off counterparty transaction • Man-Template for template manually entered transaction.	
party_id	String	CMM No Map	The ID of the transaction's entity.	AcmeUS
set_of_books_name	Strong	CMM No Map	The set of books of the transaction. Note: This is specified as additional attribute of the entity.	
source_reference_id	Strong	CMM No Map	The original ID of the transaction.	
treasure_gl_id	Integer	CMM No Map	The ID of the treasury general ledger entry.	
txn_amount	Double	CMM No Map	The value of the transaction (in the currency defined in txn_currency).	4566.45
txn_currency	String	CMM No Map	The ISO currency code of the transaction.	USD
txn_date	Date	CMM No Map	The actual date of the transaction.	20-Oct-2006
txn_direction	double	CMM No Map	The direction of the transaction. Acceptable values: • C for credit • D for debit.	C
value_date	Date	CMM No Map	The value date of the transaction.	20-Oct-2006

B.2.7 Interchange export attributes

The following table presents common interchange export attributes supported by CMM:

Name	Data type	Map type	Description	Example
Basic attributes				
interchange_sender_id	String	CMM No Map	ID code published by the sender for other parties to use as the receiver ID.	

Name	Data type	Map type	Description	Example
sender_id_qualifier_code	String	CMM No Map	The system/method of the code structure used to designate the sender or receiver ID element being qualified.	
interchange_recipient_id	String	CMM No Map	ID code published by the receiver of data.	
recipient_id_qualifier_code	String	CMM No Map	The system/method of the code structure used to designate the sender or receiver ID element being qualified.	
functional_group_sender_id	String	CMM No Map	ID code published by the sender of the data at the group level. Note: This attribute is used in the X12 format.	
functional_group_receiver_id	String	CMM No Map	ID code published by the receiver of the data at the group level. Note: This attribute is used in the X12 format.	
Runtime-specific attributes				
import_export_id	Integer	CMM No Map	A unique sequential ID assigned to every interface with CMM. Note: The value of this attribute is derived at runtime and provided by the controller.	

Appendix C Handlers

CMM includes several handlers that you can use in XML template-based formats.

Handlers are tools that interact with your XML files and the application. They can retrieve or update data, and perform conditional handling or operations. They are created and maintained by Wall Street Systems as part of the CMM `DefaultData` folder.

Handlers are meant to be as generic as possible so that they can support a number of functions rather than very specific conditions or processing requirements.

Handlers cannot be moved outside the CMM `DefaultData` folder and cannot be changed by any organization outside of Wall Street Systems.

This section lists the most common handlers supported by CMM. For documentation on all handlers supported by CMM, navigate to the following files under `[CMM Server]/cmm/xml_handler_spec/handlers` using your browser:

```
add_to_buffer.html
attribute_defined_element.html
build_mail_message.html
build_txn_groups.html
charset.html
component_transactions.html
condition_test_element.html
configure_attribute_container.html
context_defined_element.html
context_value_iterator.html
create_node.html
create_object.html
create_tree.html
echo.html
find_parent_node.html
find_root_node.html
get_system_date.html
if.html
include.html
increment_counter.html
insert_chars.html
iterate_db_result_set.html
log_invalid_transaction.html
```

map_value.html

mathFunction.html

pad_handler.html

parse_delimiter.html

parse_fixed_width.html

parse_range_width.html

parse_xml_tree.html

remittance_details_list.html

remove_chars.html

remove_context_variable.html

replace_string.html

reset_counter.html

save_value_to_context.html

send_mail_message.html

set_context_variable.html

set_node_attribute.html

set_value.html

sql_query_call.html

static_data_element.html

store_object.html

substring.html

traverse_tree.html

truncate_handler.html

txn_aggregation.html

txn_groups_list.html

txn_list.html

use_attribute_container.html

use_buffer_input.html

use_node_attribute.html

use_node_value.html

use_output_device.html.

Note: In the above URL, `[CMM Server]` is the location of your CMM server (for example, `http://treasury.acme-co.com`).

C.1 add_to_buffer

This handler reads the data from the current input device and adds them to the buffer in a context with the supplied buffer name and value. If a value is not provided, the handler retrieves the value from the input device and adds it to the buffer in a context.

C.1.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`buffer_name`	String	The name of the buffer in a context.	Yes	
`value`	String	The value of the data in a context.	No	

C.1.2 Examples

The following is an example application of this handler:

```
<add_to_buffer buffer_name="current_node" value="segment"/>
```

This example reads the data from the current input device and adds them to the buffer with `current_node` as the value of `buffer_name` and `segment` as the value of `value`.

C.2 attribute_defined_element

This handler translates and formats the attribute ID into the corresponding data value.

C.2.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`value`	String	The ID of the attribute.	Yes	
`format`	String	The format pattern for the attribute.	No	
`datatype`	String	The data type of the attribute.	No	
`start_index`	Integer	The starting index of the attribute.	No	
`end_index`	Integer	The ending index of the attribute.	No	
`currency_attribute`	String	The currency code to format the attribute.	No	
`thousands_delimiter`	Char	The delimiter for thousands.	No	
`decimal_delimiter`	Char	The delimiter for decimals.	No	
`decimal_delimiter_required`	Boolean	A boolean value that determines whether the decimal delimiter is required or not.	No	`false`

Attribute	Data type	Description	Required	Default value
decimal_digits	String	The number of decimal digits. Note: implied indicates this value is derived based on the currency code.	No	
side	String	The side of the attribute to be manipulated. Acceptable values: • PAD_LEFT • PAD_RIGHT.	No	PAD_LEFT
length	Integer	The maximum length of the attribute.	No	0
pad_char	Char	A character that is used for padding.	No	blank
escape_characters_counted	Boolean	A boolean value that determines whether the escape characters should be counted.	No	false
splitter_group_length	Integer	The splitter group length.	No	
splitter_delimiter	String	The splitter delimiter.	No	
zero_decimals_eliminated	Boolean	A boolean value that determines whether to eliminate the zero decimals.	No	false

C.2.2 Examples

The following is an example application of this handler:

```
<attribute_defined_element value="customer_reference_id" length="16" pad_char=" "
side="PAD_RIGHT"/>
```

This example formats the value of customer_reference_id by padding a blank to the right if its length is less than 16 characters.

C.3 build_mail_message

Creates a mail message. Mail message is stored in the current context.

C.3.1 Parameters

See the example below.

C.3.2 Examples

```
<build_mail_message mail_message_id="message1">
        <message_subject>
            Deal Position Report Completed
        </message_subject>
        <message_priority value="important"/>
        <message_text content_type="text/plain">
          Scheduled Deal Position report generation task has completed. See attached
```

```
report
        </message_text>
        <message_attachments>
            <attachment
physical_path="${_runtime_}\Development\CmmInstallationData\DealPosition.html"/>
        </message_attachments>
    </build_mail_message>
```

C.4 build_txn_groups

This handler groups transactions based on grouping criteria.

Note: You need to use the appropriate transaction attributes, such as payment and receipt attributes.

C.4.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
grouping_id	String	The ID of the transaction grouping.	No	
max_group_size	Integer	The maximum group size by which to split all the transaction lists into child lists.	No	

The following are this handler's parameters that are specified as child elements:

- group_by

  Grouping criteria.

  This child element contains the following attribute:

Attribute	Data type	Description	Required	Default value
attribute_id	String	A grouping criterion on the transaction list.	Yes	

C.4.2 Examples

The following is an example application of this handler:

```
<build_txn_groups>
   <group_by attribute_id="party_bank_account_id"/>
   <group_by attribute_id="value_date"/>
   <group_by attribute_id="currency"/>
</build_txn_groups>
```

This example groups the transactions based on the entity bank account ID, the value date, and the currency code of the transaction.

C.5 charset

This handler forces CMM to transform data to the specified character set.

C.5.1 Parameters

The following are this handler's parameters that are specified as child elements:

- `valid`

  A valid character in the character set. This child element is only required if other child elements are not specified.

  This child element contains the following attribute:

Attribute	Data type	Description	Required	Default value
`char`	Char	A character being specified as a valid character in the set.	Yes	

- `mapped`

  A mapping of one character to another. This child element is only required if other child elements are not specified.

  This child element contains the following attributes:

Attribute	Data type	Description	Required	Default value
`from`	Char	A character being mapped to another character.	Yes	
`to`	Char	A character being mapped to another character.	Yes	

- `default`

  A default character in the character set. This child element is only required if other child elements are not specified.

  This child element contains the following attribute:

Attribute	Data type	Description	Required	Default value
`value`	Char	A character being defined as a default character.	Yes	

C.5.2 Examples

The following is an example application of this handler:

```
<charset>
   <valid char="A"/>
   <valid char="B"/>
   <valid char="C"/>
   <valid char="D"/>
   <mapped from="@" to="?"/>
   <mapped from=";" to="?"/>
   °
</charset>
```

C.6 component_transactions

This handler iterates through the component transactions of the current transaction (assuming the current transaction is an aggregate transaction).

C.6.1 Parameters

This handler does not have parameters.

C.6.2 Examples

The following is an example application of this handler:

```
<if condition="${is_aggregate_transaction}" operator="equals" value="YES">
   <component_transactions>
      <include filename="interfaces.exports.clieop03.pmt.transactionRecord.xml"/>
   </component_transactions>
</if>
```

This example populates the component transaction information if the current transaction is an aggregate.

C.7 condition_test_element

This handler allows you to test a condition on the attribute value.

C.7.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`condition_type`	String	The type of the condition test.	Yes	

The following are this handler's parameters that are specified as child elements:

- `attribute_id`

  The ID of the attribute. This child element can be specified more than once.

  This child element contains the following attribute:

Attribute	Data type	Description	Required	Default value
`value`	String	the ID of the attribute.	Yes	

- `parameter`

  The value of the parameter. This child element can be specified more than once and is required when `list_parameter` is not available.

This child element contains the following attributes:

Attribute	Data type	Description	Required	Default value
`id`	String	The ID of the parameter.	Yes	
`value`	String	The value of the parameter.	Yes	

- `list_parameter`

  The list of parameters. This child element can be specified more than once and is required when `parameter` is not available.

  This child element contains the following attributes:

Attribute	Data type	Description	Required	Default value
`id`	String	The ID of the parameter.	Yes	
`list_item.value`	Element	The value of the `list_item` element of the parameter.	Yes	

- `result_handler`

  The status code of the result handler. This element can be specified more than once.

  This child element contains the following attribute:

Attribute	Data type	Description	Required	Default value
`status_code`	String	The status code of the result handler.	Yes	

C.7.2 Component handlers

The following are this handler's component handlers that are specified as child elements:

- `string_length_validation`

  A component handler that allows you to validate the length of a string. Possible status codes:

  - `WITHIN_BOUNDS`
  - `LESS_THAN_MIN`
  - `GREATER_THAN_MAX`
  - `INVALID_TEST_PARAMETERS`
  - `VALUE_IS_NULL`.

This component handler contains the following attributes:

Attribute	Data type	Description	Required	Default value
attribute_id	String	The value taken from the first element of attribute_id in the input attribute list.	Yes	
min_length	Integer	The minimum length of the string that is specified in a parameter element.	Yes	
max_length	Integer	The maximum length of the string that is specified in a parameter element.	Yes	

- equals

  A component handler that allows you to test the equality of two values. Possible status codes:

  - EQUAL
  - NOT_EQUAL.

  This component handler contains the following attributes:

Attribute	Data type	Description	Required	Default value
attribute_id	String	The first value taken from the first element of attribute_id in the input attribute list.	Yes	
static_string	String	The second value taken from the first element of parameter or list_parameter if available; otherwise, the second element of the attribute element.	Yes	

- exists

  A component handler that allows you to test whether the value does exist or not. Possible status codes:

  - EXISTS
  - DOES_NOT_EXIST.

  This component handler contains the following attribute:

Attribute	Data type	Description	Required	Default value
attribute_id	String	The value taken from the first element of attribute_id in the input attribute list.	Yes	

- attribute_value_in_domain

A component handler that allows you to test whether the value does exist in the given domain. Possible status codes:

- NO_DOMAIN_DEFINED
- NO_VALUE_DEFINED
- VALUE_IN_DOMAIN
- VALUE_NOT_IN_DOMAIN.

This component handler contains the following attribute:

Attribute	Data type	Description	Required	Default value
attribute_id	String	The value taken from the first element of attribute_id in the input attribute list.	Yes	
domain	Element	The domain values specified in the list_item child elements being taken from the list_parameter element if available.	Yes	

- string_length_in_domain

  A component handler that allows you to test whether the length of the value does exist in the given domain. Possible status codes:

  - NO_DOMAIN_DEFINED
  - NO_VALUE_DEFINED
  - VALUE_IN_DOMAIN
  - VALUE_NOT_IN_DOMAIN.

  This component handler contains the following attributes:

Attribute	Data type	Description	Required	Default value
attribute_id	String	The value taken from the first element of attribute_id in the input attribute list.	Yes	
domain	Element	The domain values specified in the list_item child elements being taken from the list_parameter element if available.	Yes	

- number_range

  A component handler that allows you to test whether the numeric value is in the certain range. Possible status codes:

  - WITHIN_BOUNDS
  - VALUE_IS_NULL
  - INVALID_TEST_PARAMETERS
  - LESS_THAN_MIN
  - GREATER_THAN_MAX.

This component handler contains the following attribute:

Attribute	Data type	Description	Required	Default value
attribute_id	String	The value taken from the first element of attribute_id in the input attribute list.	Yes	
minimum	Integer	The minimum number that is specified in a parameter element.	Yes	
maximum	Integer	The maximum number that is specified in a parameter element.	Yes	

- is_numeric_validation

  A component handler that allows you to test whether the value is a number. Possible status codes:

  - VALUE_IS_NULL
  - IS_NUMERIC
  - IS_NOT_NUMERIC.

  This component handler contains the following attribute:

Attribute	Data type	Description	Required	Default value
attribute_id	String	The value taken from the first element of attribute_id in the input attribute list.	Yes	

- number_length_validation

  A component handler that allows you to validate the length of the value. Possible status code:

  - WITHIN_BOUNDS
  - LESS_THAN_MIN
  - GREATER_THAN_MAX
  - INVALID_TEST_PARAMETERS
  - VALUE_IS_NULL.

This component handler contains the following attributes:

Attribute	Data type	Description	Required	Default value
attribute_id	String	The value taken from the first element of attribute_id in the input attribute list.	Yes	
min_length	Integer	The minimum length of the string that is specified in a parameter element.	Yes	
max_length	Integer	The maximum length of the string that is specified in a parameter element.	Yes	

- regex

  A component handler that allows you to evaluate the value based on the given regular expression. This is a very powerful handler that you do almost any validations with the appropriate regular expression. Possible status codes:

  - INVALID_PATTERN
  - VALUE_IS_NULL
  - MATCHES
  - DOES_NOT_MATCH.

  This component handler contains the following attributes:

Attribute	Data type	Description	Required	Default value
attribute_id	String	The value taken from the first element of attribute_id in the input attribute list.	Yes	
attribute_id	String	The value taken from the first element of attribute_id in the input attribute list.	Yes	

- switch

  A component handler that allows you to test whether the value is met the specified condition before switching to that block. Possible status code: Anything that you are expected from the attribute value.

  This component handler contains the following attributes:

Attribute	Data type	Description	Required	Default value
attribute_id	String	The value taken from the first element of attribute_id in the input attribute list.	Yes	

C.7.3 Examples

The following is an example application of this handler that uses the string_length_validation component handler:

```
<condition_test_element condition_type="string_length_validation">
```

```
   <attribute_id value="payee_bank_account_number"/>
   <parameter id="min_length" value="1"/>
   <parameter id="max_length" value="34"/>
   <result_handler status_code="LESS_THAN_MIN">
      <log_invalid_transaction attribute_id="payee_bank_account_number"
      user_message="Payee Bank Account Number must be an alphanumeric string length
      range between 1 to 34."/>
   </result_handler>
   <result_handler status_code="GREATER_THAN_MAX">
      <log_invalid_transaction attribute_id="payee_bank_account_number"
      user_message="Payee Bank Account Number must be an alphanumeric string length
      range between 1 to 34."/>
   </result_handler>
</condition_test_element>
```

Note: An example using the `number_length_validation` component handler would be similar in structure.

The following is an example application of this handler that uses the `equals` component handler:

```
<condition_test_element condition_type="equals">
   <attribute_id value="is_valid_transaction"/>
   <parameter id="static_string" value="YES"/>
   <result_handler status_code="EQUAL">
      <include
      filename="interfaces.exports.citibank.payext.ceemea.transactionFormat.xml"/>
      <increment_counter counter_id="transaction_set_count"/>
      <flag_payment_as_exported/>
   </result_handler>
</condition_test_element>
```

The following is an example application of this handler that uses the `exists` component handler:

```
<condition_test_element condition_type="exists">
   <attribute_id value="payor_bank_swift_code"/>
   <result_handler status_code="DOES_NOT_EXIST">
      <log_invalid_transaction user_message="Payor bank SWIFT code is not
      provided,which is mandatory for all SWIFT payments."/>
   </result_handler>
   <result_handler status_code="EXISTS">
      <condition_test_element condition_type="string_length_in_domain">
         <attribute_id value="payor_bank_swift_code"/>
         <list_parameter id="domain">
            <list_item value="8"/>
            <list_item value="11"/>
         </list_parameter>
         <result_handler status_code="VALUE_NOT_IN_DOMAIN">
            <log_invalid_transaction attribute_id ="payor_bank_swift_code"
            user_message="Invalid payor bank SWIFT code, it must be either 8 or 11
            characters long."/>
         </result_handler>
      </condition_test_element>
   </result_handler>
</condition_test_element>
```

The following is an example application of this handler that uses the `attribute_value_in_domain` component handler:

```
<condition_test_element condition_type="attribute_value_in_domain">
   <attribute_id value="payment_primary_delivery_channel"/>
   <list_parameter id="domain">
      <list_item value="ACH"/>
      <list_item value="WT"/>
      <list_item value="BOOK_TRANSFER"/>
```

```
      <list_item value="CHQ"/>
   </list_parameter>
   <result_handler status_code="VALUE_NOT_IN_DOMAIN">
      <log_invalid_transaction attribute_id ="payment_primary_delivery_channel"
      user_message="invalid payment type, it is not supported by this interface
      currently"/>
   </result_handler>
   <result_handler status_code="VALUE_IN_DOMAIN">
      °
   </result_handler>
</condition_test_element>
```

The following is an example application of this handler that uses the `string_length_in_domain` component handler:

```
<condition_test_element condition_type="string_length_in_domain">
   <attribute_id value="payor_bank_swift_code"/>
   <list_parameter id="domain">
      <list_item value="8"/>
      <list_item value="11"/>
   </list_parameter>
   <result_handler status_code="VALUE_NOT_IN_DOMAIN">
      <log_invalid_transaction attribute_id ="payor_bank_swift_code"
      user_message="Invalid payor bank SWIFT code, it must be either 8 or 11
      characters long."/>
   </result_handler>
</condition_test_element>
```

The following is an example application of this handler that uses the `number_range` component handler.

```
<condition_test_element condition_type="number_range">
   <attribute_id value="payment_amount"/>
   <parameter id="minimum" value="1"/>
   <parameter id="maximum" value="12500"/>
   <result_handler status_code="LESS_THAN_MIN">
      <log_invalid_transaction attribute_id="payment_amount" user_message="Payment
      Amount must be a number whose range is between 1 and 12500."/>
   </result_handler>
   <result_handler status_code="GREATER_THAN_MAX">
      <log_invalid_transaction attribute_id="payment_amount" user_message="Payment
      Amount must be a number whose range is between 1 and 12500."/>
   </result_handler>
</condition_test_element>
```

The following is an example application of this handler that uses the `is_numeric_validation` component handler:

```
<condition_test_element condition_type="is_numeric_validation">
   <attribute_id value="payment_amount"/>
   <result_handler status_code ="IS_NOT_NUMERIC">
      <log_invalid_transaction attribute_id ="payment_amount" user_message="Payment
      Amount should be a numeric value"/>
   </result_handler>
</condition_test_element>
```

The following is an example application of this handler that uses the `regex` handler:

```
<condition_test_element condition_type="regex">
   <attribute_id value="customer_reference_id"/>
   <parameter id="pattern" value="[/].*"/>
   <result_handler status_code ="MATCHES">
      <log_invalid_transaction attribute_id
      ="customer_reference_id"user_message="Customer reference ID must not start
      with a slash '/' "/>
```

```
   </result_handler>
</condition_test_element>
```

The following is an example application of this handler that uses the `switch` handler.

```
<condition_test_element condition_type="switch">
   <attribute_id value="format_specification"/>
   <result_handler status_code="TREMA_LETTER_OF_CREDIT">
      <include filename="interfaces.exports.trema.receipt.lcrspec.xml"/>
   </result_handler>
   <result_handler status_code="CFONB">
      <include filename="interfaces.exports.cfonb.letters_of_credit.lcr_spec.xml"/>
   </result_handler>
</condition_test_element>
```

C.8 configure_attribute_container

This handler configures an attribute container in a context by setting the container ID, the attribute definition provider context ID, and the context ID.

C.8.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`container_id`	String	The ID of the container in a context.	No	
`adp_context_id`	String	The attribute definition provider ID in a context.	No	
`context_id`	String	The ID of a context.	Yes	
`filter_null_values`	Boolean	A boolean value that determines whether the container should filter null values when setting values.	No	`true`

C.8.2 Examples

The following is an example application of this handler:

```
<configure_attribute_container context_id="bank_balance"/>
```

This example configures an attribute container in a context with `bank_balance` as the value of `context_id`.

C.9 context_defined_element

This handler translates and formats the attribute ID to the corresponding data value.

C.9.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`value`	String	The ID of the attribute.	Yes	
`format`	String	The format pattern for the attribute.	No	
`datatype`	String	The data type of the attribute.	No	
`start_index`	Integer	The starting index of the attribute.	No	
`end_index`	Integer	The ending index of the attribute.	No	
`currency_attribute`	String	The currency code to format the attribute.	No	
`thousands_delimiter`	Char	The delimiter for thousands.	No	
`decimal_delimiter`	Char	The delimiter for decimals.	No	
`decimal_delimiter_required`	Boolean	A boolean value that determines whether the decimal delimiter is required.	No	`false`
`decimal_digits`	String	The number of decimal digits. Note: `implied` indicates this value is derived based on the currency code.	No	
`side`	String	The side on which the attribute should be manipulated. Acceptable values: • `PAD_LEFT` • `PAD_RIGHT`.	No	`PAD_LEFT`
`length`	Integer	The maximum length of the attribute.	No	`0`
`pad_char`	Char	A character that is used for padding.	No	`blank`
`escape_characters_counted`	Boolean	A boolean that determines whether the escape characters should be counted.	No	`false`
`splitter_group_length`	Integer	The splitter group length.	No	
`splitter_delimiter`	String	The splitter delimiter.	No	
`zero_decimals_eliminated`	Boolean	A boolean value that determines whether to eliminate the zero decimals.	No	`false`

C.9.2 Examples

The following is an example application of this handler:

```
<attribute_defined_element value="customer_reference_id" length="16" pad_char=" "
side="PAD_RIGHT"/>
```

This example formats the value of `customer_reference_id` by padding a blank to the right if its length is less than 16 characters.

C.10 context_value_iterator

This handler stores the next value from the supplied iterator in the context with the supplied key and a list ID if one exists.

C.10.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`key`	String	The value of the key in a context.	Yes	
`list_id`	String	The ID of the list in a context.	No	
`value`	String	The value of the value in a context.	No	

C.10.2 Examples

The following is an example application of this handler:

```
<context_value_iterator key="attribute_name">
   <value value="BATCHID"/>
   <value value="ORIGINATINGSYSTEMID"/>
</context_value_iterator>
```

This example iterates the list in a context with `attribute_name` as the value of `key` and set values in the context.

C.11 create_node

This handler creates a node in a tree, or a child node under a parent node, with a node name.

C.11.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`node_name`	String	The name of the node in a tree.	Yes	

C.11.2 Examples

The following is an example application of this handler:

```
   <create_node node_name="segment"/>
```

This example creates a node in a tree with `segment` as `node_name`.

C.12 create_object

This handler creates an object with a context ID and type. Usually, it is used before the `store_object` handler.

C.12.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`context_id`	String	The ID of the object in the context.	Yes	
`type`	String	The type of the object. Note: `file_attribute`, `bankbalance`, `banktxn`, and `bankmessage` imply the file attribute, bank balance, bank transaction, and bank message objects respectively.	Yes	

C.12.2 Examples

The following is an example application of this handler:

```
<create_object type ="file_attribute" context_id ="file_level_ob"/>
```

This example creates an object with `file_attribute` as the value of `type` and `file_level_ob` as the value of `context_id`.

C.13 create_tree

This handler creates a tree with a node name and tree ID.

C.13.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`node_name`	String	The name of the node in a tree.	Yes	
`tree_id`	String	The ID of the tree.	Yes	

C.13.2 Examples

The following is an example application of this handler:

```
<create_tree tree_id="bai_tree" node_name="file" >
```

This example creates a tree with `bai_tree` as the value of `tree_id` and `file` as the value of `node_name`.

C.14 echo

This handler logs a message to either the standard output or the log file. This handler is very useful for debugging XML-template-based formats.

C.14.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`message`	String	The message to be logged.	Yes	
`verbose`	Boolean	The verbose mode for the message logging. Note: The output logs can be found in …\CmmVirtualDirectory\logs\auros under the user message log file.	No	`false`
`terse`	Boolean	The terse mode for the message logging. Note: The output logs can be found in …\CmmVirtualDirectory\logs\auros under the user message log file.	No	`false`
`warning`	Boolean	The warning mode for the message logging. Note: The output logs can be found in …\CmmVirtualDirectory\logs\auros under the user message log file.	No	`false`
`error`	Boolean	The error mode for the message logging. Note: The output logs can be found in …\CmmVirtualDirectory\logs\auros under the user message log file.	No	`false`
`systemout`	Boolean	The systemout mode for the message logging. Note: The output logs can be found in …\CmmRuntime\var\jvm_stdout.yyyy_MM_dd_hh_mm_ss.log.	No	`false`

C.14.2 Examples

The following is an example application of this handler:

```
<echo message="payee_long_name=${payee_long_name}" systemout="true"/>
<if condition="${payee_long_name}" operator="equals" value="">
   <log_invalid_transaction attribute_id ="payee_long_name" user_message="payee long
   name is not provided."/>
</if>
```

This example echoes the value of `payee_long_name` to the standard output before performing the condition test.

C.15 find_parent_node

This handler finds a parent node in a tree with a parent node name. Usually, it is used in conjunction with the `create_node` handler.

C.15.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
parent_node_name	String	The name of a parent node in a tree.	Yes	

C.15.2 Examples

The following is an example application of this handler:

```
<find_parent_node parent_node_name="file"/>
<create_node node_name="segment"/>
°
```

This example finds the parent node in a tree with `file` as the value of `parent_node_name` and creates a node with `segment` as the value of `node_name` so that the `segment` node becomes the child node of the `file` node.

C.16 find_root_node

This handler finds the root node in a tree. Usually, it is used before a tree is traversed.

C.16.1 Parameters

This handler does not have parameters.

C.16.2 Examples

The following is an example application of this handler:

```
<find_root_node/>
<use_attribute_container container_id="BankBalanceContainer">
   <traverse_tree>
      °
   </traverse_tree>
</use_attribute_container>
```

This example finds the root node in a tree.

C.17 get_system_date

This handler retrieves the system date in a specified format.

C.17.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
variable_name	String	The name of the context variable.	Yes	

Attribute	Data type	Description	Required	Default value
format	String	The format pattern of the system date. Note: See the Sun website for instructions on how to specify this format pattern.	No	

C.17.2 Examples

The following is an example application of this handler:

```
<get_system_date variable_name="my_formatted_date" format="dd-MMM-yyyy"/>
```

This example gets the system date in the format pattern `dd-MMM-yyyy` storing it in the `my_formatted_date` context variable.

C.18 if

This handler tests a condition before executing a block of scripts if the condition is satisfied.

C.18.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default Value
condition	String	The value being evaluated against the value. Note: If the condition holds a boolean value, the value can be omitted.	Yes [1]	
value	String	The value being evaluated against the value of the condition.	No	
is_defined	String	The identifier of the attribute. Note: Use `operator="notequals"` to negate the condition.	Yes [2]	
operator	String	The operator for the condition test. Acceptable values: • equals • notequals • lessthan • greaterthan.	No	equals

Table notes:

1. Only if `is_defined` is not available.
2. Only if `condition` is not available.

C.18.2 Examples

The following is an example application of this handler:

```
<if is_defined="cheque_number">
   <attribute_defined_element value="cheque_number" end_index="30"/>
</if>
```

This example truncate `cheque_number` to a maximum of 30 characters (if `cheque_number` is defined).

The following is another example application of this handler:

```
<payment_list>
   <if condition="${is_valid_transaction}" operator="equals" value="YES">
      °
   </if>
</payment_list>
```

or

```
<payment_list>
   <set_context_variable variable_name="__is_valid" value="false"/>
   <if condition="${is_valid_transaction}" operator="equals" value="YES">
      <set_context_variable variable_name="__is_valid" value="true"/>
   </if>
   <if condition="${__is_valid}">
      °
   </if>
</payment_list>
```

This example populates the payment message if and only if the payment is a valid transaction.

C.19 include

This handler references other scripts within a script.

Note: Do not use this handler to reference scripts with child nodes.

C.19.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data Type	Description	Required	Default Value
`target_node`	String	The ID of the target node.	No	
`filename`	String	The absolute path of the referenced script.	Yes [1]	
`filename_attribute_id`	String	The ID of the `filename` attribute.	Yes [2]	
`priority_filename_list`	String	The absolute path of the script to invoke as a last resort. Note: This is very useful for the export framework when dealing with a variety of country specifications. The `filename` attribute can specify the path to the country specification and the `priority_filename_list` attribute can specify the path to the generic specification in case the country specification does not exist.	No	

Table notes:

1. Only if the `filename_attribute_id` attribute is not available.
2. Only if the `filename` attribute is not available.

C.19.2 Examples

The following is an example application of this handler:

```
<include filename="interfaces.exports.swift.generic.character_set.xml"/>
```

This example includes the script in the specified path of `filename`.

The following is another example application of this handler:

```
<include
filename="interfaces.exports.boa.ansi.820.${country_spec}.TransactionSetFormat.xml"
priority_filename_list="interfaces.exports.boa.ansi.820.${countries_generic}.Transac
tionSetFormat.xml"/>
```

This example includes the `TransactionSetFormat.xml` script if the path of `filename` exists; otherwise, it loads the script from the path of `priority_filename_list`.

C.20 increment_counter

This handler increments the counter variable by one.

C.20.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`counter_id`	String	The ID of the counter variable.	Yes	

C.20.2 Examples

The following is an example application of this handler:

```
<increment_counter value="num_of_payments"/>
```

This example increments the counter variable, `num_of_payments`, by 1.

C.21 insert_chars

This handler inserts special characters in the attribute value.

Warning: The modified value is stored in the `result_attribute_id` if it is provided; otherwise, the modified value replaces the original value.

C.21.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`attribute_id`	String	The ID of the attribute.	Yes	
`result_attribute_id`	String	The ID of the result attribute.	No	

Attribute	Data type	Description	Required	Default value
insert_indexes	Integer	A list of indexes where the special string is inserted. Note: Each index is separated by a comma.	No	
insert_string	String	The special string to be inserted into the attribute value.	No	

C.21.2 Examples

The following is an example application of this handler:

```
<insert_chars attribute_id="my_var" insert_indexes="3,9" insert_string="test"/>
```

This example inserts the special string, `test`, at the specified indexes of the value of the `my_var` attribute and stores the modified value to `my_var`. The special string is only inserted if the index is less than the length of the attribute value.

The following is another example application of this handler:

```
<insert_chars attribute_id="my_var" result_attribute_id="his_var"
insert_indexes="3,9" insert_string="test"/>
```

This example inserts the special string, `test`, at the specified indexes of the value of the `my_var` attribute and stores the modified value to `his_var`. The special string is only inserted if the index is less than the length of the attribute value.

C.22 iterate_db_result_set

This handler iterates the SQL query result set and stores the field value of every row to the context keyed by the column name. The action handler specified performs actions on those context variables (individual field values). This handler must be used in conjunction with the `sql_query_call` handler.

C.22.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
result_set_id	String	The context variable name for the result set stored in context when the `sql_query_call` handler is called.	Yes	

C.22.2 Examples

The following is an example application of this handler:

```
<sql_query_call query="select BALANCETYPEDESC, DATETIMEMODIFIED from balancetypes
where TXNSUBTYPEID in (3,6)" result_set_id="balance_types_id"/>
<iterate_db_result_set result_set_id ="balance_types_id">
   <echo message="current balance type desc is: ${BALANCETYPEDESC}"
   systemout="true"/>
   <echo message="current date time modified is: ${DATETIMEMODIFIED}"
   systemout="true"/>
```

```
</iterate_db_result_set>
```

This example iterates the result set of a select query and prints out each individual field value. This example expect to return two rows and two columns; you should see four values printed out.

C.23 jython_script

This handler allows you to call Python scripts from within **XML** handler scripts.

C.23.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`script_location`	String	The file location of the Python script. If you do not provide this parameter, the default location is …\InstallationData\installation\scripts\.	No	
`script_name`	String	The file name of the Python script.	No	

C.23.2 Examples

The following is an example application of this handler:

```
<script_def>
   <jython_script script_location="C:\PythonTest" script_name="testscript.py"/>
</script_def>
```

C.24 log_invalid_transaction

This handler asserts the error message to the user whenever the transaction fails on the validations. For the release process, this error message can be viewed from the release screen.

C.24.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`attribute_id`	String	The ID of the attribute.	No	
`user_message`	String	The error message to be displayed to the user.	No	

C.24.2 Examples

The following is an example application of this handler:

```
<log_invalid_transaction attribute_id="payor_bank_account_number"
user_message="payor bank account number is not provided or invalid, it must be
numeric value with length of 10"/>
```

This example asserts the error message to the user when the payor bank account number does not match the expected length of 10 numeric characters.

The following is another example application of this handler:

```
<log_invalid_payment attribute_id="payor_bank_account_number" user_message="payor
bank account number is not provided or invalid, it must be numeric value with length
of 10"/>
```

This example produces the same results as the one above. However, the `log_invalid_payment` handler is deprecated. Use the `log_invalid_transaction` handler instead.

C.25 map_value

This handler maps the values from the context to the attribute container if the source ID is provided; otherwise, the value read from the input device is mapped. If the target ID cannot be found, the default value is used.

C.25.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`source_id`	String	The ID of the source in a context or in the input stream.	Yes	
`target_id`	String	The ID of the target in a context or in the input stream.	Yes	
`default_value`	String	The default value for the mapping.	No	

The following are this handler's parameters that are specified as child elements:

- `map`
- `mapping`

  This child element contains the following attributes:

Attribute	Data type	Description	Required	Default value
`source_value`	String	The value of the source from a context or the input stream.	Yes	
`target_value`	String	The value of the target from a context or the input stream.	Yes	

C.25.2 Examples

The following is an example application of this handler:

```
<map_value source_id="report_id" target_id="file_type" default_value="P">
   <map>
      <mapping source_value="1" target_value="P"/>
      <mapping source_value="2" target_value="R"/>
   </map>
</map_value>
```

This example map the values with `report_id` as the value of `source_id` in a context or in the input device, `file_type` as the value of `target_id` in a context or in the input device, and `P` as the value of `default_value`.

C.26 mathFunction

This handler creates a new attribute that holds a given math function value of a given attribute.

C.26.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`new_attribute_id`	String	The ID of the new attribute that holds a given math function value of a given attribute.	Yes	
`attribute_id`	String	The ID of the attribute.	Yes	
`function`	String	The ID of the math function.	Yes	

C.26.2 Examples

Suppose the `payment_amount` value is 99.99:

```
<mathfunction new_attribute_id="new_payment_amount" attribute_id="payment_amount"
function="int"/>
```

The `new_payment_amount` value is 99.00.

```
<mathfunction new_attribute_id="new_payment_amount" attribute_id="payment_amount"
function="floor"/>
```

The `new_payment_amount` value is 99.00.

```
<mathfunction new_attribute_id="new_payment_amount" attribute_id="payment_amount"
function="ceil"/>
```

The `new_payment_amount` value is 100.00.

```
<mathfunction new_attribute_id="new_payment_amount" attribute_id="payment_amount"
function="abs"/>
```

The `new_payment_amount` value is 99.00.

```
<mathfunction new_attribute_id="new_payment_amount" attribute_id="payment_amount"
function="round"/>
```

The `new_payment_amount` value is 100.00.

C.27 pad_handler

This handler allows you to manipulate an attribute by padding the characters to the left or the right of its value.

Warning: The padded value is stored in the `result_attribute_id` if it is provided; otherwise, the padded value replaces the original value.

C.27.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default Value
`attribute_id`	String	The ID of the source attribute.	Yes	
`result_attribute_id`	String	The ID of the result attribute.	No	
`length`	Integer	The specified length of the attribute value.	Yes	
`side`	String	The side on which the attribute value should be padded. Acceptable values: • `PAD_LEFT` • `PAD_RIGHT`.	No	`PAD_LEFT`
`pad_char`	Char	A character that is padded to the attribute value if the length of the attribute value is less than the specified length.	No	

C.27.2 Examples

The following is an example application of this handler:

```
<pad_handler attribute_id="customer_reference_id" length="16" pad_char=" "
side="PAD_RIGHT"/>
```

This example formats the value of `customer_reference_id` by padding a blank to the right if its length is less than 16 characters and stores the padded value to the `customer_reference_id` attribute.

The following is another example application of this handler:

```
<pad_handler attribute_id="customer_reference_id" result_attribute_id="cus_ref_id"
length="16" pad_char=" " side="PAD_RIGHT"/>
```

This example formats the value of `customer_reference_id` by padding a blank to the right if its length is less than 16 characters and stores the padded value to the `cus_ref_id` attribute.

C.28 parse_delimiter

This handler parses the data from the input device when the node pattern is encountered. The parsed data can be stored in a tree or in context for subsequent processes. If the string of the pattern is empty, the default string of the pattern is used except for those used above.

C.28.1 Parameters

The following are this handler's parameters that are specified as child elements:

- `pattern`

This child element contains the following attributes:

Attribute	Data type	Description	Required	Default value
`string`	String	The value of the pattern in a tree.	Yes	

C.28.2 Examples

The following is an example application of this handler:

```
<parse_delimiter>
   <pattern string="/&#13;&#10;">
      <set_context_variable variable_name="is_start_of_segment" value="true"/>
   </pattern>
   <pattern string="/&#13;">
      <set_context_variable variable_name="is_start_of_segment" value="true"/>
   </pattern>
   <pattern string=",">
      <find_parent_node parent_node_name="segment"/>
      <create_node node_name="element"/>
   </pattern>
   <pattern string="">
      <if is_defined="is_start_of_segment">
         <find_parent_node parent_node_name="file"/>
         <create_node node_name="segment"/>
         <remove_context_variable variable_name="is_start_of_segment"/>
      </if>
      <set_node_attribute attribute_name="value"/>
   </pattern>
</parse_delimiter>
```

In this example:

- The first pattern is encountered in the input device and its value is a string `/
` (slash carriage return and line feed), so the context variable is set with `is_start_of_segment` as `variable_name` and `true` as `value`.
- The second pattern is encountered and its value is a string `/
` (slash carriage return), and the context variable is also set with `is_start_of_segment` as `variable_name` and `true` as `value`.
- The third pattern is encountered and its value is a string ",", the handler finds the parent node `segment` and creates a child node `element` under it.
- The fourth pattern is encountered and its value is a string "" (an empty string) meaning that except for the above patterns, the default pattern is an empty string, so the handler first checks that if a variable `is_start_of_segment` is defined or not then tries to find the parent node `file` and creates a child node `segment` under it, set the node attribute with the `variable_name` `value` and ends the `parse_delimiter`.

C.29 parse_fixed_width

This handler parses the data from the input device when the attribute width is encountered. The parsed data can be stored in a tree or context for subsequent processes. If the width and length are both defined, the handler verifies the length of the data to decide if the parsing continues. If the width is defined but the length is not defined, the parsing continues anyways.

C.29.1 Parameters

The following are this handler's parameters that are specified as child elements:

- `width`

  This child element contains the following attributes:

Attribute	Data type	Description	Required	Default value
`length`	String	The value of the width of the data in the input device.	No	

C.29.2 Examples

The following is an example application of this handler:

```
<parse_fixed_width>
   <width length="2">
      <verify_input_stream value="12"/>
   </width>
</parse_fixed_width>
```

In this example, the node `width` is encountered in the input device and the value of `length` is `2`, and the `value 12` is verified in the input stream of the context with that in the tree.

The following is another example application of this handler:

```
<parse_fixed_width>
   <width>
      <verify_input_stream value="bank_reference"/>
   </width>
</parse_fixed_width>
```

In this example, the node `width` is encountered in the input device and no value of the length specified, and the `value bank_reference` is verified in the input stream of the context with that in the tree.

C.30 parse_range_width

This handler parses data from the input stream based on the width range and data type defined. The actual data parsed is within the range defined but has to match the data type defined. This is useful in parsing SWIFT files, which have range fields and specific data types defined.

C.30.1 Parameters

The following are this handler's parameters that are specified as child elements:

- `width`

  This child element contains the following attributes:

Attribute	Data type	Description	Required	Default value
`min`	integer	The minimum width of the data to be read.	No	`1`
`max`	integer	The maximum width of the data to be read.	No	`Maximum Integer`

Attribute	Data type	Description	Required	Default value
`type`	String	The characters type of the data to be read. Acceptable values: • `alpha` • `numeric` • `alpha_numeric` • `decimal` • `fixed_values` • `any`.	Yes	
`mandatory`	boolean	A boolean value that determines if this field is mandatory or optional.	No	`true`
`valid_values`	String	The valid values.	Yes [1]	

Table notes:

1. Only if the `type` attribute is set to `fixed_values`.

C.30.2 Examples

The following is an example application of this handler:

```
<parse_range_width>
   <width min="3" max="3" type="numeric">
      <save_value_to_context key="account_num">
   </width>
   <width min="2" max="2" type="numeric" mandatory="false">
      <save_value_to_context key="branch_num">
   </width>
   <width max="5" type="alpha">
      <save_value_to_context key="transaction_reference">
   </width>
   <width max="15" type="decimal">
      <save_value_to_context key="transaction_amount">
   </width>
   <width min="2" max="3" type="fixed_values" valid_values="AB,BA,CDC,HK,XYZ,PL">
      <save_value_to_context key="transaction_type">
   </width>
   <width type="any"/>
      <save_value_to_context key="transaction_description">
   </width>
</parse_range_width>
```

C.31 parse_xml_tree

This handler parses an XML file and stores the DOM tree parsed to the context for subsequent tree traversing. If the schema file is provided, the XML file is first validated by the schema; otherwise, only the DOM tree is generated.

C.31.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
schema_file	String	The relative path of the XSD schema file to be used to validate the XML stream.	No	
tree_id	String	An ID to be used to track the tree in the context.	No	dom_tree_id

C.31.2 Examples

The following is an example application of this handler:

```
<parse_xml tree_id="my_tree"
schema_file="com.trema.cmmext.message.schema.tremabankaccountstatement.xml"/>
<include filename="interfaces.imports.demo.demo_traverse_dom_tree.xml"/>
```

In this example:

- The first element validates the XML file with the schema specified, parses the XML file into a DOM tree, and store the DOM tree into the context with a tree ID of `my_tree`.
- The second element traverse the tree stored in context and maps the node attributes and values to the corresponding business objects subsequently based on the mapping logic defined in the `demo_traverse_dom_tree.xml` file.

C.32 remittance_details_list

This handler iterates through the remittance details list for the population of a payment message.

C.32.1 Parameters

This handler does not have parameters.

C.32.2 Examples

The following is an example application of this handler:

```
<remittance_details_list>
    °
</remittance_details_list>
```

This example populates the remittance details for the payment message.

C.33 remove_chars

This handler removes special characters from an attribute's value.

Warning: The modified value is stored in `result_attribute_id` if it is provided; otherwise, the modified value replaces the original value.

C.33.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
attribute_id	String	The ID of the attribute.	Yes	
result_attribute_id	String	The ID of the result attribute.	No	
remove_chars	String	Special characters to be removed from the attribute value.	No	

C.33.2 Examples

The following is an example application of this handler:

```
<remove_chars attribute_id="my_var" remove_chars="-"/>
```

This example removes the hyphen from the value of the my_var attribute and stores the modified value in the my_var attribute.

The following is an example application of this handler:

```
<remove_chars attribute_id="my_var" result_attribute_id="his_var" remove_chars="-"/>
```

This example removes the hyphen from the value of the my_var attribute and stores the modified value in the his_var attribute.

C.34 remove_context_variable

This handler allows you to remove previously declared context variables.

C.34.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
variable_name	String	The name of the context variable.	Yes	
remove_all	Boolean	A boolean value that determines whether all the context variables should be removed.	No	

C.34.2 Examples

The following is an example application of this handler:

```
<remove_context_variable variable_name ="is_uk_domestic"/>
```

This example removes the context variable with the name is_uk_domestic.

C.35 replace_string

This handler allows you to replace all occurrences of an old string component with a new string component for an attribute value.

Warning: The modified value is stored in the **result_attribute_id** if it is provided; otherwise, the modified value replaces the original value.

C.35.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`attribute_id`	String	The ID of the source attribute.	Yes	
`result_attribute_id`	String	The ID of the result attribute.	No	
`old_string`	String	The string component to be replaced.	Yes	
`new_string`	String	The string to replace all occurrence of `old_string`.	Yes	

C.35.2 Examples

The following is an example application of this handler:

```
<replace_string attribute_id="my_var" old_string="abc" new_string="xyz"/>
```

This example replaces all occurrences of `abc` in the `my_var` variable with `xyz` and stores the modified value to the `my_var` attribute.

The following is an example application of this handler:

```
<replace_string attribute_id="my_var" result_attribute_id="his_var" old_string="," new_string="."/>
```

This example replaces all occurrences of `,` in the `my_var` variable with `.` and stores the modified value to the `his_var` attribute.

C.36 reset_counter

This handler resets the counter variable to its original value (0).

C.36.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`counter_id`	String	The ID of the counter variable.	Yes	

C.36.2 Examples

The following is an example application of this handler:

```
<reset_counter counter_id="num_of_payments"/>
```

This example resets the `num_of_payments` counter variable to 0.

C.37 save_value_to_context

This handler attempts to retrieve the value from the current node with the node attribute if it is provided. If there is no value found, the handler then attempts to retrieve it from the current input device and saves it to the context with a key. The context variable can be treated as a list; in this case, a list is stored in the context with the supplied key and the value from the input device is appended to the list. Usually, this handler is used in conjunction with the `use_node_attribute` handler.

C.37.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`key`	String	The value of the key in a context.	Yes	
`node_attribute`	String	The value of the attribute of the current node.	No	
`is_list`	boolean	A boolean value that determines whether it is set in a context.	No	

C.37.2 Examples

The following is an example application of this handler:

```
<use_node_attribute attribute_id="FileID">
   <save_value_to_context key="fileid"/>
</use_node_attribute>
```

This example retrieves a value with the attribute name `FileID`, which is the attribute ID of the `use_node_attribute` handler in the current node, and saves it to the context with `fileid` as the value of the key.

The following is an example application of this handler:

```
<action>
   <save_value_to_context key="message_reference_id" node_attribute ="value"/>
</action>
```

This example saves the value to the context with `value` as the value of `node_attribute` and `message_reference_id` as the value of `key`.

C.38 send_mail_message

Send a mail message thats been built and stored in the user context.

Mail is sent to the SMTP server configured through Configuration Parameters (on page 2 of the interface).

C.38.1 Parameters

See the example below.

C.38.2 Examples

```
<send_mail_message mail_message_id="message1">
    <message_addresses>
        <from_address value="support@company.com"/>
        <to_addresses>
            <address value="dmossman@company.com"/>
        </to_addresses>
        <cc_addresses>
        </cc_addresses>
        <bcc_addresses>
        </bcc_addresses>
        <reply_to_addresses>
            <address value="person@company.com"/>
        </reply_to_addresses>
    </message_addresses>
</send_mail_message>
```

C.39 set_context_variable

This handler declares a variable within the context. Once declared, it can be referenced afterwards. It is strongly recommended to remove the context variable when it is not used.

C.39.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`variable_name`	String	The name of the context variable.	Yes	
`value`	String	The value of the context variable.	Yes	
`datatype`	String	The data type of the context variable. Acceptable values: • `string` • `date` • `double` • `integer`.	No	
`format`	String	The format pattern of the context variable.	No	

C.39.2 Examples

The following is an example application of this handler:

```
<set_context_variable variable_name="is_uk_domestic" value="F"/>
```

```
<condition_test_element condition_type="equals">
   <attribute_id value="payor_bank_country_code"/>
   <parameter id="static_string" value="GB"/>
   <result_handler status_code="EQUAL">
      <set_context_variable variable_name ="is_uk_domestic" value="T"/>
   </result_handler>
</condition_test_element>
```

This example declares a context variable, `is_uk_domestic`, whose value is `T` if the payor bank country code is in the United Kingdom or `F` if it is not.

C.40 set_node_attribute

This handler sets an attribute in the current node in a tree with an attribute name when the append attribute exists and is true and when there is a value attribute provided set the value with it. If the value is not provided, this handler sets the value by getting it from the current input device.

C.40.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`attribute_name`	String	The name of the attribute on the current node in a tree.	Yes	
`append`	boolean	A boolean value that determines if the value of the attribute needs to be appended.	No	
`value`	String	The value of the value on the current node in a tree.	No	

C.40.2 Examples

The following is an example application of this handler:

```
<root>
   <create_tree tree_id="tree" node_name="a"/>
   <create_node node_name="b"/>
   <set_node_attribute attribute_name="z" value="Z"/>
   <set_node_attribute attribute_name="y" value="Y"/>
   <set_node_attribute attribute_name="x" value="X"/>
   <set_node_attribute attribute_name="x" value="eks" append="true"/>
   <set_node_attribute attribute_name="w" value="W"/>
   <set_node_attribute attribute_name="w" value="doubleyou"/>
   <create_node node_name="c"/>
   <set_node_attribute attribute_name="v"/>
   <find_parent_node parent_node_name="a"/>
   <create_node node_name="d"/>
</root>
```

This example supports three scenarios:

- The attribute is set to the current node in a tree with an `attribute_name` and `value`.
- The attribute is set to the current node in a tree with an `attribute_name` and `value` when `append` is `true`
- The attribute is set to the current node in a tree with only an `attribute_name` provided.

C.41 set_value

This handler sets the value of the current object with a value if it is provided; otherwise, this handler retrieves the value form the current input device and sets it into the current object with a value ID, data type, and format if the value and format exist. If the value is specified and the data type is not specified, the data type is defaulted to string.

C.41.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`value_id`	String	The attribute ID of the current object. Note: This must be a valid attribute ID of the current object.	Yes	
`value`	String	The value to be set to the attribute ID of the current object.	No	
`datatype`	String	The data type of the value to be set to the attribute IDof the current object. Valid data types are: • `integer` • `double` • `date` • `boolean`. Note: If the data type value is specified, it has to match the data type required by the attribute ID.Mismatching will cause runtime exceptions. If the data type value is not specified, `String` will be the default data type.	No	`String`
`format`	String	The format of the value to be set to the current object. If the value of the `datatype` attribute is `date`, the value you provide for this attribute must conform to the Java simple date format pattern (see the Sun website).	No	

C.41.2 Examples

The following is an example application of this handler:

```
<build_transaction_object>
   <set_value value_id="bank_acct_number" value="5168516"/>
   <set_value value_id="currency_code" value="GBP"/>
```

```
   <NAryFunction arg1="${transaction_amount}" arg2="100" operation="/"
   result_variable_name="txn_amount_with_decimal"/>
   <set_value value_id="amount" value="516.61" datatype="double"/>
   <set_value value_id="originating_bank_id"/>
   <set_value value_id="type_code" value="B"/>
   <set_value value_id="value_date" value="20050718" datatype="date"
   format="yyyyMMdd"/>
</build_transaction_object>
```

This example supports four scenarios:

- Set the value into the object with a `value_id` and `value`.
- Set the value into the object with a `value_id`, `value`, and `datatype`.
- Set the value into the object with only a `value_id`.
- Set the value into the object with a `value_id`, `value`, `datatype`, and `format`.

C.42 sql_query_call

This handler calls a stored procedure or SQL query statement and stores the query result set in the context as a context variable for other handlers to process. The nesting parameter is only required if the call is a stored procedure call.

C.42.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`db_config_name`	String	The database configuration file name without `.xml`. This file should be stored in `<CMMHOME>\InstallationData\installation\database` and should be based on `config.xml` in the same directory.	No	`config`
`stored_procedure_name`	String	The stored procedure name of the underlying database.	Yes, but only if the SQL call is a stored procedure call.	
`query`	String	The complete SQL query statement to query the underlying database.	Yes, but only if the SQL call is a query call.	
`result_set_id`	String	The context variable name for the result set returned from the query or stored procedure. Wallstreet recommends you specify this only if you expect to return multiple rows and you want to process all the rows. The individual field value of the first row is always stored in the context keyed by the column name regardless of the `resut_set_id` attribute, which is for multiple rows.	No	

The following are this handler's parameters that are specified as child elements:

- `stored_procedure_name`

  If this is provided, you have to specify the stored procedure parameters in the nested parameter elements in the order of appearance in the stored procedure call.

  This child element contains the following attributes:

Attribute	Data type	Description	Required	Default value
`name`	String	The parameter name.	Yes	
`value`	String	The parameter value.	Yes	
`data_type`	String	The parameter data type. Acceptable values: • `integer` • `double` • `date` • `string`.	No	`string`
`param_type`	String	The parameter type that indicates whether the parameter is an input, output, or return parameter. Acceptable values: • `input` • `output` • `return`. `return` must appear in the first parameter element.	Yes	

C.42.2 Examples

The following is an example application of this handler:

```
<sql_query_call db_config_name="TRM_config", query="select
BALANCETYPEDESC,DATETIMEMODIFIED from balancetypes where TXNSUBTYPEID in (3,6)"
result_set_id="balance_types_id"/>
```

This example calls the select query to query the TRM database and stores the result set into context.

The following is another example application of this handler:

```
<set_context_variable variable_name="block_size" value="100"/>
<sql_query_call stored_procedure_name="Block_ID">
   <parameter name="tablename" value="INTERCHANGE" param_type="input"/>
   <parameter name="columnname" value="INTERCHANGEID" param_type="input"/>
   <parameter name="block" value="${block_size}" data_type="integer"
   param_type="input"/>
   <parameter name="sequenceid" data_type="integer" param_type="output"/>
</sql_query_call>
```

This examples calls the select query to query the TRM database and stores the result set into context.

C.43 static_data_element

This handler writes static text in the value attribute to the output stream. For the release process, static text is written to the bank file.

C.43.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`value`	String	The static data element.	Yes	

C.43.2 Examples

The following is an example application of this handler:

```
<static_data_element value="UNZ"/>
```

This example write the value `UNZ` to the output stream.

C.44 store_object

This handler stores an object in a data container in a context with a context ID.

C.44.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
`context_id`	String	The ID of the object in a context.	Yes	
`remove_after_store`	boolean	A boolean value that determines if the object is removed from the context after store.	No	`true` (To avoid duplicated object storing)

C.44.2 Examples

The following is an example application of this handler:

```
<store_object context_id="bank_balance"/>
```

This example store an object into a data container in a context with `bank_balance` as the value of `context_id`.

C.45 substring

This handler creates a new attribute that holds a substring of a given attribute.

C.45.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data type	Description	Required	Default value
substring_attribute_id	String	The ID of the substring of the given attribute.	No	
attribute_id	String	The ID of the attribute.	Yes	
start_index	Integer	The starting index of the attribute.	No	1
end_index	Integer	The ending index of the attribute.	No	length of value

C.45.2 Examples

Suppose payor_bank_account_number value is 123456789:

```
<substring substring_attribute_id="payor_bank_account_number_x1"
attribute_id="payor_bank_account_number" start_index="1" end_index="1"/>
```

The payor_bank_account_number_x1 value is 1.

```
<substring substring_attribute_id="payor_bank_account_number_x2"
attribute_id="payor_bank_account_number" start_index="2" end_index="2"/>
```

The payor_bank_account_number_x2 value is 2.

```
<substring substring_attribute_id="payor_bank_account_number_x13"
attribute_id="payor_bank_account_number" end_index="3"/>
```

The payor_bank_account_number_x13 value is 123.

```
<substring substring_attribute_id="payor_bank_account_number_x49"
attribute_id="payor_bank_account_number" start_index="4"/>
```

The payor_bank_account_number_x49 value is 456789.

```
<substring substring_attribute_id="payor_bank_account_number_x"
attribute_id="payor_bank_account_number"/>
```

The payor_bank_account_number_x value is 123456789.

C.46 traverse_tree

This handler traverses a tree to map values and store data in objects when a node is encountered. If the value of the domain_match attribute is true, the handler attempts to match the element defined in a tree with the one defined here at any position of the sequence; otherwise, the handler attempts to match the element defined in a tree with the one defined here at the exact position of the sequence. If the value of the repeating attribute is true, the handler repeats the action when the element defined in a tree matches the one defined here; otherwise, it does not repeat the action. If there is an attribute with a name and value under a node with a name, the handler attempts to match the attribute name and value in a tree with the one defined here and takes action.

C.46.1 Parameters

The following are this handler's parameters that are specified as child elements:

- node

This child element contains the following attributes:

Attribute	Data type	Description	Required	Default value
`name`	String	The name of the node in a tree.	Yes	
`domain_match`	boolean	A boolean value that determines if the elements in a tree match those defined here.	No	
`repeating`	boolean	A boolean value that determines if the action needs to be repeated when the elements in a tree match the ones defined here.	No	

- `attribute`

  This child element contains the following attributes:

Attribute	Data type	Description	Required	Default value
`name`	String	The name of the attribute of the node in a tree.	Yes	
`value`	String	The value of the attribute of the node in a tree.	Yes	

- `action`

C.46.2 Examples

The following is an example application of this handler:

```
<traverse_tree>
   <node name="file" domain_match="true">
      <action>
         <set_context_variable variable_name="is_file_attribute_stored"
         value="false"/>
      </action>
      <node name="segment">
         <attribute name="value" value="01"/>
         <action>
            <create_object type="file_attribute" context_id="file_level_ob"/>
         </action>
         <node name="element" repeating="true">
            <action>
               <include
               filename="interfaces.imports.demo.bai.fileheadermapping.xml"/>
            </action>
         </node>
      </node>
   </node>
</traverse_tree>
```

This example traverses a tree. When the first node `file` is encountered with the value of `domain_match` set to true, the `traverse_tree` handler tries to match the element in the tree with the one defined here and takes the `set_context_variable` action with `is_file_attribute_stored` as the value of `variable_name` and `false` as the value of `value`. When the second node `segment` is

encountered, the handler tries to match the element with `value` as the attribute name and `01` as the value of the attribute value in the tree with the one defined here and takes the `create_object` action with `file_attribute` as the type of the object and `file_level_ob` as the value of `context_id`. When the third node `element` is encountered with the value of `repeating` set to `true`, the handler tries to match the element in the tree with the one defined here and takes the include action with `interfaces.imports.demo.bai.fileheadermapping.xml` as the value of `filename`.

C.47 truncate_handler

This handler updates the attribute by truncating its value either to the left or the right.

Warning: The modified value will be stored in the `result_attribute_id` if it is provided; otherwise, the modified value will replace the original value.

C.47.1 Parameters

The following are this handler's parameters that are specified as attributes:

Attribute	Data Type	Description	Required	Default Value
`attribute_id`	String	The ID of the attribute.	Yes	
`result_attribute_id`	String	The ID of the result attribute.	No	
`length`	Integer	The specified length of the attribute value.	Yes	
`side`	String	The side on which the attribute value should be kept. Acceptable values: • `KEEP_LEFT` • `KEEP_RIGHT`.	No	`KEEP_LEFT`

C.47.2 Examples

The following is an example application of this handler:

```
<truncate_handler attribute_id="customer_reference_id" length="16"
side="KEEP_LEFT"/>
```

This example formats the value of `customer_reference_id` by truncating the right if its length exceeds 16 characters and stores the truncated value to the `customer_reference_id` attribute.

The following is another example application of this handler:

```
<truncate_handler attribute_id="customer_reference_id"
result_attribute_id="cus_ref_id" length="16" side="KEEP_LEFT"/>
```

This example formats the value of `customer_reference_id` by truncating the right if its length exceeds 16 characters and stores the truncated value to the `cus_ref_id` attribute.

C.48 txn_aggregation

This handler groups transactions based on the given criteria defined in the `aggregation_key` child element.

C.48.1 Parameters

The following table presents this handler's parameters specified as attributes:

Attribute	Data type	Description	Required	Default value
`type`	String	The ID of the transaction aggregation.	Yes	
`assign_default_attributes`	Boolean	A boolean value that determines whether to assign the default attributes.	No	`true`
`max_size`	Integer	A number that determines whether to assign the default attributes.	No	`0` (All in a single list)

The following are this handler's parameters that are specified as child elements:

- `aggregation_key`

  A component handler that allows you to group the transactions based on the given transaction criteria.

- `attribute`

  The transaction attribute to be grouped with.

  This child element contains the following attribute:

Attribute	Data type	Description	Required	Default value
`id`	String	The ID of the attribute. Note: Refer to the appropriate transaction attribute documentation for this value.	Yes	

- `attribute`

  The transaction attribute to be grouped with.

  This child element contains the following attributes:

Attribute	Data type	Description	Required	Default value
`id`	String	The ID of the attribute. Note: Refer to the appropriate transaction attribute documentation for this value.	Yes	

- `accumulate_attributes`

A component handler that accumulates the transaction attributes together.

- `attribute`

  Optional. The transaction attribute to be grouped with.

  This child element contains the following attributes:

Attribute	Data type	Description	Required	Default value
`id`	String	The ID of the attribute.	Yes	
`operation`	String	The type of operation to be performed on the specified attributes. Acceptable values: • `append` • `sum` • `assign`.	Yes	
`delimeter`	Char	A delimiter to be used in the concatenation of the transaction attributes.	Yes [1]	
`max_length`	Integer	The maximum length of the output string.	Yes [1]	

Table notes:

1. Only if the `operation` attribute is set to `append`.

C.48.2 Examples

The following is an example application of this handler:

```
<txn_aggregation type="AP Batch">
   <aggregation_key>
      <attribute id="payor_bank_account_id"/>
      <attribute id="value_date"/>
      <attribute id="payment_currency"/>
      <attribute id="payment_primary_delivery_channel"/>
      <attribute id="domestic_crossborder_status"/>
      <attribute id="transaction_type_code"/>
      <attribute id="source_batch_id"/>
   </aggregation_key>
   <accumulate_attributes>
      <!-- do nothing for now -->
   </accumulate_attributes>
</txn_aggregation>
```

This example presents a sample payment aggregator for payments coming from the same accounts payable batch. This could also be used for other transaction types like direct debits and letters of credit.

C.49 txn_groups_list

This handler iterates through transactions that have been grouped by the `build_txn_groups` handler.

C.49.1 Parameters

The following table presents this handler's parameters specified as attributes:

Attribute	Data type	Description	Required	Default value
grouping_id	String	The ID of the transaction grouping.	No	

C.49.2 Examples

The following is an example application of this handler:

```
<txn_groups_list>
   <if condition="${is_valid_transaction}" operator="equals" value="YES">
      <include filename="interfaces.exports.cfonb.dd.DD_Record.xml"/>
   </if>
</txn_groups_list>
```

This example iterates through the groups of transaction lists for the population of the direct debit export message.

C.50 txn_list

This handler iterates through a list of transactions for the population of the transaction message.

Note: You need to use the appropriate transaction attributes, such as payment and receipt attributes.

C.50.1 Parameters

This handler does not have parameters.

C.50.2 Examples

The following is an example application of this handler:

```
<txn_list>
   <if condition="${is_valid_transaction}" operator="equals" value="YES">
      °
   </if>
</txn_list>
```

This example iterates through the transaction list to populate the export message.

The following is another example application of this handler:

```
<txn_list>
   <include filename="interfaces.exports.cfonb.dd.DD_Record06.xml"/>
</txn_list>
```

This example presents another way to populate the direct debits message.

C.51 use_attribute_container

This handler uses an attribute container with a `container_id` to set values in the objects in the container or store some objects in the container.

C.51.1 Parameters

The following table presents this handler's parameters specified as attributes:

Attribute	Data type	Description	Required	Default value
`container_id`	String	The ID of the container.	Yes	

C.51.2 Examples

The following is an example application of this handler:

```
<use_attribute_container container_id="BankBalanceContainer">
   <traverse_tree>
      °
   </traverse_tree>
</use_attribute_container>
```

This example uses the attribute container with `BankBalanceContainer` as the value of `container_id` to set values in the objects in the container or store some objects in the container.

C.52 use_buffer_input

This handler uses data in the buffer in the current input device with a `buffer_name` to set the data and put it in the input device.

C.52.1 Parameters

The following table presents this handler's parameters specified as attributes:

Attribute	Data type	Description	Required	Default value
`buffer_name`	String	The name of the buffer.	Yes	

C.52.2 Examples

The following is an example application of this handler:

```
<use_buffer_input buffer_name="current_node">
   <set_node_attribute attribute_name="value"/>
</use_buffer_input>
```

This example uses the data with the `buffer_name current_node` in the buffer in the current input device to set the attribute of the node with `value` as `attribute_name` in the input device.

C.53 use_node_attribute

This handler uses the value of the `attribute_id` of the current node in a tree as the attribute name to set or save values.

C.53.1 Parameters

The following table presents this handler's parameters specified as attributes:

Attribute	Data type	Description	Required	Default value
`attribute_id`	String	The ID of the attribute for a node in a tree.	Yes	

C.53.2 Examples

The following is an example application of this handler:

```
<use_node_attribute attribute_id="FileID">
   <save_value_to_context key="fileid"/>
</use_node_attribute>
```

This example retrieves the value with the attribute name `FileID`, which is the attribute ID of the `use_node_attribute` handler in the current node, and saves it to the context with `fileid` as the value of `key`.

The following is an example application of this handler:

```
<use_node_attribute attribute_id="value">
   <set_value value_id="actual_amount" datatype="double"/>
</use_node_attribute>
```

This example retrieves the value with the attribute name `value`, which is the attribute ID of the `use_node_attribute` handler in the current node, and sets it to the context with `actual_amount` as the value of `value_id` and `double` as the value of `datatype`.

C.54 use_node_value

This handler uses the node value of the current node in a tree as the source value for subsequent processes.

C.54.1 Parameters

This handler does not have parameters.

C.54.2 Examples

The following is an example application of this handler:

```
<use_node_value>
   <save_value_to_context key="fileid"/>
</use_node_value>
```

This example retrieves the value from the current node and saves it to the context with `fileid` as the variable name.

The following is another example application of this handler:

```
<use_node_value>
```

```
   <set_value value_id="actual_amount" datatype="double"/>
</use_node_value>
```

This example retrieves the value from the current node and sets it to the current object with `actual_amount` as `value_id` and `double` as `datatype`.

C.55 use_output_device

This handler uses an output device, such as a file or a database, with a `device_id` to do something like storing objects in the output device.

C.55.1 Parameters

The following table presents this handler's parameters specified as attributes:

Attribute	Data type	Description	Required	Default value
`device_id`	String	The ID of the device for output.	Yes	

C.55.2 Examples

The following is an example application of this handler:

```
<use_output_device device_id="BankBalanceOutput">
   <store_object context_id="bb"/>
</use_output_device>
```

This example uses the output device with `BankBalanceOutput` as the value of the `device_id` here to store an object with `bb` as the value of `context_id` into the output device.

Appendix D SSL certificate generation and configuration

E-business relies on the exchange of information between business partners over a network. As information travels from the source to the destination, there is a risk of it being stolen or modified. In web services transactions using SOAP, information is passed between the service invoker and service provider as plain XML. Therefore, any person who intercepts the messages can read the information exchanged.

CMM has implemented its own web services security model to secure web services applications using secure socket layer (SSL) over HTTPS. SSL allows web browsers and web servers to communicate over a secured connection. Therefore, the information being sent is encrypted by one side, transmitted, and then decrypted by the other side before processing. This is a two-way process, meaning that both the server and the browser encrypt all traffic before sending information. Such secured authentication guarantees that the service is accessible for anyone with a verified identity.

The remainder of this appendix provide the guidelines for setting up web services security in CMM for both internal and external types. Specifically, the remainder of this appendix documents the steps required to set up a new secure website for listening to incoming requests over HTTPS and to generate SSL certificates.

Note: This appendix uses OpenSSL to generate the internal SSL certificates and even the external certificates. However, you can use other tools that work for your organization.

D.1 Assumptions

This appendix assumes the following:

- Your organization has successfully installed CMM.
- Your organization has successfully installed the following software:

Category	Requirement
Operating system	• Microsoft Windows 2000 Server • Microsoft Windows Server 2003
Web server	• Microsoft IIS 5.0 • Microsoft IIS 6.0 • Apache Tomcat 5.x
CMM application server	• BEA WebLogic 9.2 • Apache Tomcat 5.x
CMM database server	• Oracle • Microsoft SQL Server
SSL-KeyTool	• Version 2.4 or later (provided by Wallstreet)

- To generate SSL certificates in an operating system other than Windows, OpenSSL is installed and accessible from the command line. Usually, OpenSSL is automatically included with Linux.

D.2 Generating SSL certificates

To establish a trusted network among business partners or within the entities of an organization, the certificates being exchanged must be signed by the same trusted certificate authority (CA).

D.2.1 Generating SSL certificates

To generate SSL certificates:

1. Obtain the latest version of SSL-KeyTool from Wallstreet.
2. Extract SSL-KeyTool to a temporary folder (for example, `c:\temp\SSL-KeyTool\`).
3. Navigate to the temporary folder you created in step 2.
4. Open the `build.properties` file in a text editor.
5. Locate the following section:

```
global.country=CA
global.state=Alberta
global.location=Calgary
global.organization=Trema Laboratories Inc.
global.organization.unit=
global.emailaddress=
global.common.name=wsdemo.corp.trema.com
global.domain.name=trema.com
global.storepass=trema.ca
global.ca.keystore=cacerts.jks
global.cert.validity=9999
```

6. For each property in this section (except `global.ca.keystore`), enter an appropriate value.

For the `global.common.name property`, enter the fully qualified domain on which IIS is running. For example, if CMM's URL is `https://wsdemo.trema.com/cmm`, enter `wsdemo.corp.trema.com`.

7. Locate the following section:

```
cert.auth.id=myCA
cert.auth.name=My Certificate Authority
cert.auth.country=
cert.auth.state=
cert.auth.location=
cert.auth.organization=
cert.auth.organization.unit=Certificate Authorities
cert.auth.emailaddress=
cert.auth.common.name=
cert.auth.storepass=
cert.auth.cert.validity=
cert.auth.outpass=trema.ca
```

8. For each property in this section, do one of the following:

- Enter an appropriate value.
- Leave the value blank to use the corresponding global property's value instead.

The `cert.auth` properties are only applicable if you are generating a self-signed CA certificate.

The value of the `cert.auth.organization.unit` property is used for additional validation of the client certificate when accessing the web services.

9. Locate the following section:

```
client.id=
client.name=
client.country=
client.state=
client.location=
client.organization=
client.organization.unit=
client.emailaddress=
client.common.name=
client.storepass=
client.auth.cert.validity=
```

10. For each property in this section, do one of the following:

- Enter an appropriate value.
- Leave the value blank to use the corresponding global property's value instead.

Leave as many values blank as possible.

The information provided in this section is used to generate the certificate request to send to the CA for signing.

11. Locate the following section:

```
#openssl.cmd=<absolute path of openssl command>
```

```
#openssl.conf=<absolute path of the openssl configuration>
```

12. If you want to specify a custom path for the OpenSSL executable, remove the number signs (`#`) and enter appropriate values for the properties in this section.
13. Save and close the file.
14. Open a command prompt and navigate to the temporary folder you created in step 2.
15. Initialize the environment variables:
    - If you are using Windows, enter `setup`.
    - If you are using Linux, enter `. setup.sh`.
16. Enter `ant` to view usage information.
17. Do one of the following:
    - To generate a client SSL certificate with a self-signed CA:
      a. Generate a CA certificate by entering `ant create-CA`. SSL-KeyTool creates the following files:

Component	Location
CA private key	...\demoCA\private\<cert.auth.id.>key.pem
Self-signed CA certificate to sign the client certificate	...\demoCA\<cert.auth.id>.cert
Public CA certificate in DER format to be imported into the client trust store when using IIS as the Web server	...\demoCA\newcerts\<cert.auth.id>.der
CA certificate including both private and public keys in PKCS #12 format Note: Do not give this CA certificate to others as it can be used to sign the client certificate.	...\demoCA\<cert.auth.id>.p12
JKS keystore with a public CA certificate to ship to a Java client	...\demoCA\cacerts.jks

      b. Generate a self-signed X509 client certificate by entering "ant create-client-certificate". SSL-KeyTool creates the following files:

Component	Location
Client private key	...\demoCA\private\<client.id.>key.pem
Client certificate request to be signed by the CA	...\demoCA\<client.id>req.pem
Client certificate when the CA signs the client certificate request	...\demoCA\newcerts\<client.id>.cer
Client certificate including both private and public keys in PKCS #12 format to be imported into the client browser	...\demoCA\<client.id>.p12

To establish a secured connection with other parties being signed by the CA, a client certificate (for example, `<client.id>.p12`) and public CA certificate (for example, `<cert.auth.id>.der` or `cacerts.jks`) are required.

– To generate a client SSL certificate with a third-party CA:

a. Generate an X509 client certificate request by entering `ant create-client-certificate-request`. SSL-KeyTool creates the following files:

Component	Location
Client private key	…\demoCA\private\<client.id.>key.pem
Client certificate request to be signed by the third-party CA	…\demoCA\<client.id>req.pem

b. Send the client certificate request to the third-party CA for signing.

Once the client certificate request is signed by the third-party CA, a signed certificate is sent back. Place the signed certificate in …\demoCA\newcerts\<client.id>.cer for exporting the client certificate in PKCS #12 format.

c. Export the client certificate in PKCS #12 format by entering `ant export-client-certificate`. SSL-KeyTool creates the following files:

Component	Location
Client certificate including both private and public keys in PKCS #12 format to be imported into the client browser	…\demoCA\<client.id>.p12

To establish a secured connection with other parties being signed by this third-party CA, a client certificate (for example, `<client.id>.p12`) and a public CA certificate in either DER or JKS format are required.

D.3 Creating a secure website using IIS

To minimize interference with the existing CMM website, you need to create a new website for listening to incoming requests over HTTPS. To bypass this secure website, clients must go through client authentication by exchanging SSL certificates between the client and the server; only SSL certificates that are signed by the same CA can pass.

You can only create multiple websites in a single installation of IIS in server versions of Windows (Window 2000 Server, Windows Server 2003, and so on).

Note: As you complete the procedures in this section, you will need to restart IIS several times; therefore, Wallstreet recommends you shut down the CMM application server.

D.3.1 Assigning an additional IP address

To add a new website for HTTPS, you need to add a new IP address to the current local area connection. Ideally, a DNS name must be assigned to this IP address so that it can be accessible anywhere on the network.

D.3.1.1 Assigning an additional IP address

To assign an additional IP address:

1. Select **Start - Settings - Control Panel**.
2. In the Control Panel window, double-click **Network and Dial-up Connections**.
3. In the Network and Dialup Connections window, right-click the appropriate **Local Area Connection** and select **Properties** in the resulting popup menu.
4. In the Local Area Connection Properties dialog, click **Properties**.
5. In the Internet Protocol (TCP/IP) Properties dialog, click **Advanced**....
6. In the IP addresses section of the Advanced TCP/IP Settings dialog, click **Add**... to assign an additional IP address to the connection.

   In most organizations, the IT department is responsible for assigning the IP address.

   Wallstreet recommends that you assign a DNS name to the IP address so that it is accessible across the network. For example, the IP address `172.24.80.53` in the above image is assigned to the DNS name `wsdemo.corp.tream.com`.

D.3.2 Creating a secure website

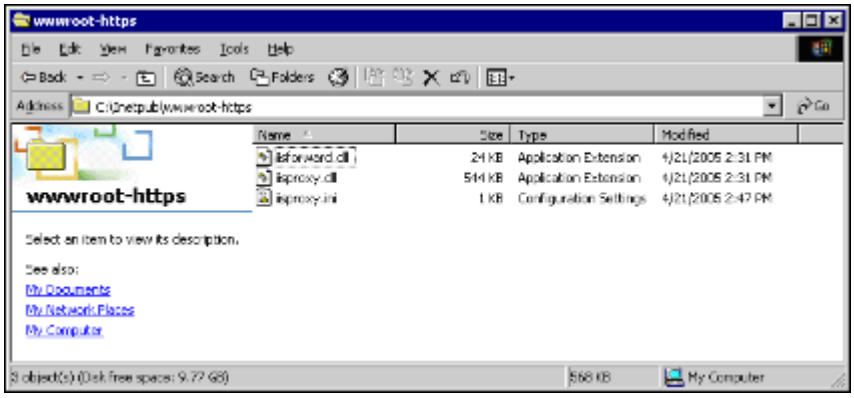
This section documents the required steps to create a new website for HTTPS in IIS.

D.3.2.1 Creating a secure website

To create a secure website:

1. Select **Start - Settings - Control Panel**.
2. In the Control Panel window, double-click **Administrative Tools**.
3. In the Administrative Tools window, double-click **Internet Services Manager**.
4. In the Internet Information Services window, right-click the computer's icon in the left frame of the window and select **New - Web Site** in the resulting popup menu.
5. In the Web Site Description panel of the website creation wizard, enter a description of the website (for example, `HTTPS Website`) in the **Description** field.
6. Click **Next >**.
7. In the IP Address and Port Settings panel of the website creation wizard, select the IP address to use for the website (for example, `172.24.80.53`) in the **Enter the IP address to use for this Web site** list.
8. Click **Next >**.
9. In the Web Site Home Directory panel of the website creation wizard, enter the path to the website's home folder (for example, `C:\Inetpub\wwwroot-https`) in the **Path** field.

The `wwwroot-https` folder in the above example is a copy of the `wwwroot` folder, but it only contains the BEA WebLogic IIS plug-in scripts:



The `iisproxy.ini` file in the `wwwroot-https` folder is very similar to the one in the `wwwroot` folder:

```
WebLogicHost=localhost
WebLogicPort=4001
WlForwardPath=/cmm
HungServerRecoverSecs=86400
Idempotent=OFF
Debug=ALL
```

10. Click **Next >**.

11. In the Web Site Access Permissions panel of the website creation wizard, click **Next >**.

12. In the final panel of the website creation wizard, click **Finish**.

D.3.3 Configuring the secure website

This section documents the required steps to configure the BEA WebLogic IIS plug-ins for the secure website you created in *D.3.2 Creating a secure website* on page 424.

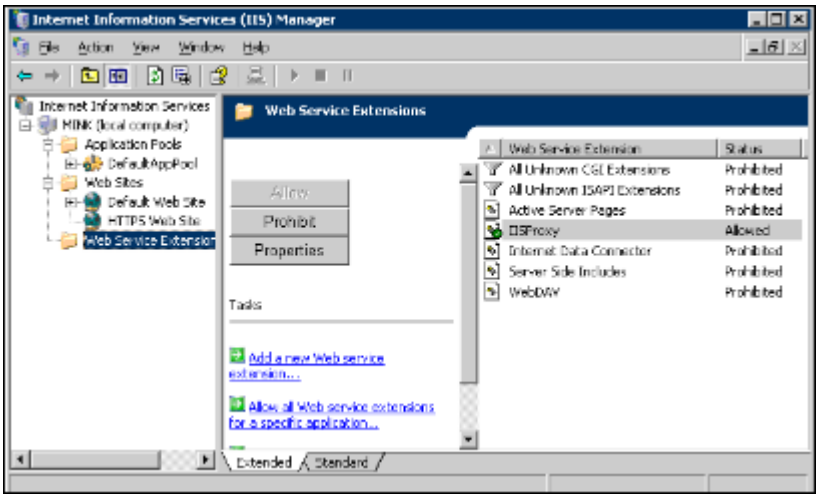
D.3.3.1 Configuring the secure website

To configure the secure website:

1. Open the Internet Information Services window (if it is not already opened).

For instructions on opening the Internet Information Services window, see *D.3.2 Creating a secure website* on page 424.

2. Right-click the secure website's icon in the left frame of the window and select **Properties** in the resulting popup menu.
3. In the [Secure website name] Properties dialog, click the **Home Directory** tab to open it.
4. Do one of the following:
   - If you are using IIS 5.0, ensure the **Application Protection** list is set to `High (Isolated)`.
   - If you are using IIS on Windows Server 2003, ensure the **Application name** field and **Application pool** list contain values.
5. Click **Configuration**....
6. In the Add/Edit Application Extension Mapping dialog, create an extension application mapping, ensuring the **Check that file exists** checkbox is not selected.
7. Click **OK**.
8. In the [Secure website name] Properties dialog, click the **ISAPI Filters** tab to open it.
9. Click **Add**....
10. In the Filter Properties dialog, enter `IISForward` in the **Filter Name** field.
11. Enter the path to secure website's `iisforward.dll` file (for example, `C:\Inetpub\wwwroot-https\iisforward.dll`) in the **Executable** field.
12. Click **OK**.
13. In the Add/Edit Application Extension Mapping dialog, click **Apply**.
14. Click **OK**.
15. Do the following:
    - If you are using Windows 2000 Server, restart IIS.
    - If you are using Windows Server 2003:
      a. Click **Web Service Extensions** in the left frame of the window.



      b. In the Web Service Extensions panel that displays in the right frame, click **Add new Web service extension**....
      c. In the New Web Service Extension dialog, enter `IISProxy_https` in the **Extension name** field.
      d. Add a path to the secure website's `iisproxy.dll` file (for example,

C:\Inetpub\https-wwwroot\iisproxy.dll) in the **Required files** list.

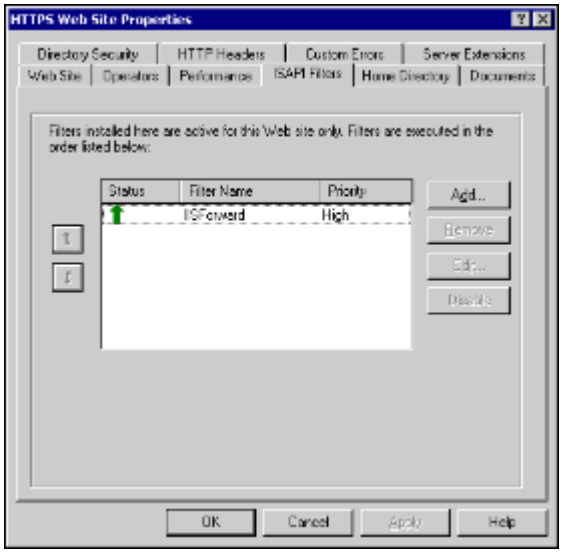
e. Select the **Set extension status to Allowed** checkbox.

f. Click **OK**.

g. Restart IIS.

16. Right-click the secure website's icon in the left frame of the window and select **Properties** in the resulting popup menu.

17. In the [Secure website name] Properties dialog, click the **ISAPI Filters** tab to open it.



A green, upwards pointing arrow displays in the **Status** column of the IISForward ISAP filter.

If **Unknown** displays in the **Priority** column of the IISForward ISAP filter, navigate to the secure website (for example, `http://wsdemo.corp.trema.com`) in your browser. This causes IIS to load the ISAPI filter.

D.3.4 Creating a virtual folder for content

This section documents the required steps to create a virtual folder in which to store CMM web content. The alias and path of this virtual folder should be similar to that of the default website (for example, `conten`).

Note: This section assumes that you have already set up the BEA WebLogic IIS plug-in on HTTP for the default website and that CMM has already been deployed to the BEA WebLogic server.

D.3.4.1 Creating a virtual folder for content

To create a virtual folder for content:

1. Open the Internet Information Services window (if it is not already opened).

For instructions on opening the Internet Information Services window, see *D.3.2 Creating a secure website* on page 424.

2. Right-click the computer's icon in the left frame of the window and select **New - Virtual Directory** in the resulting popup menu.
3. In the first panel of the virtual folder creation wizard, click **Next >**.
4. In the Virtual Directory Alias panel of the virtual folder creation wizard, enter an appropriate alias for the virtual folder (for example, `content`) in the **Alias** field.
5. Click **Next >**.
6. In the Web Site Content Directory panel of the virtual folder creation wizard, enter the path of the virtual folder (for example, `C:\lhogD1\VirtualDirectory`) in the **Directory** field.
7. Click **Next >**.
8. In the Access Permissions panel of the virtual folder creation wizard, click **Next >**.
9. In the final panel of the virtual folder creation wizard, click **Finish**.
10. Update the virtual folder path in the `nvp.xml` file:

```
<name name="ID_VirtualDirectory">
   <description>
      The virtual directory name for the path specified in
      ID_VirtualDirectoryPath
   </description>
   <value>content</value>
</name>
```

For detailed instruction on modifying the `nvp.xml` file, see the *WebSuite System Administration Guide*.

D.3.5 Configuring two-way SSL

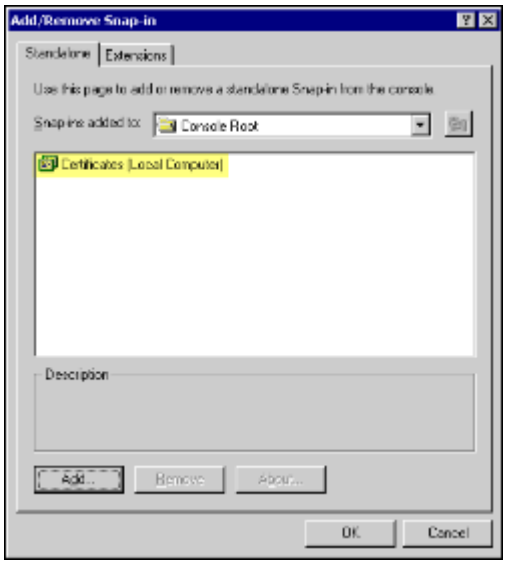
This section documents the required steps to configure two-way SSL on the IIS secure website using the test certificates you generated with the SSL-KeyTool.

D.3.5.1 Importing server and CA certificates

To import server and CA certificates:

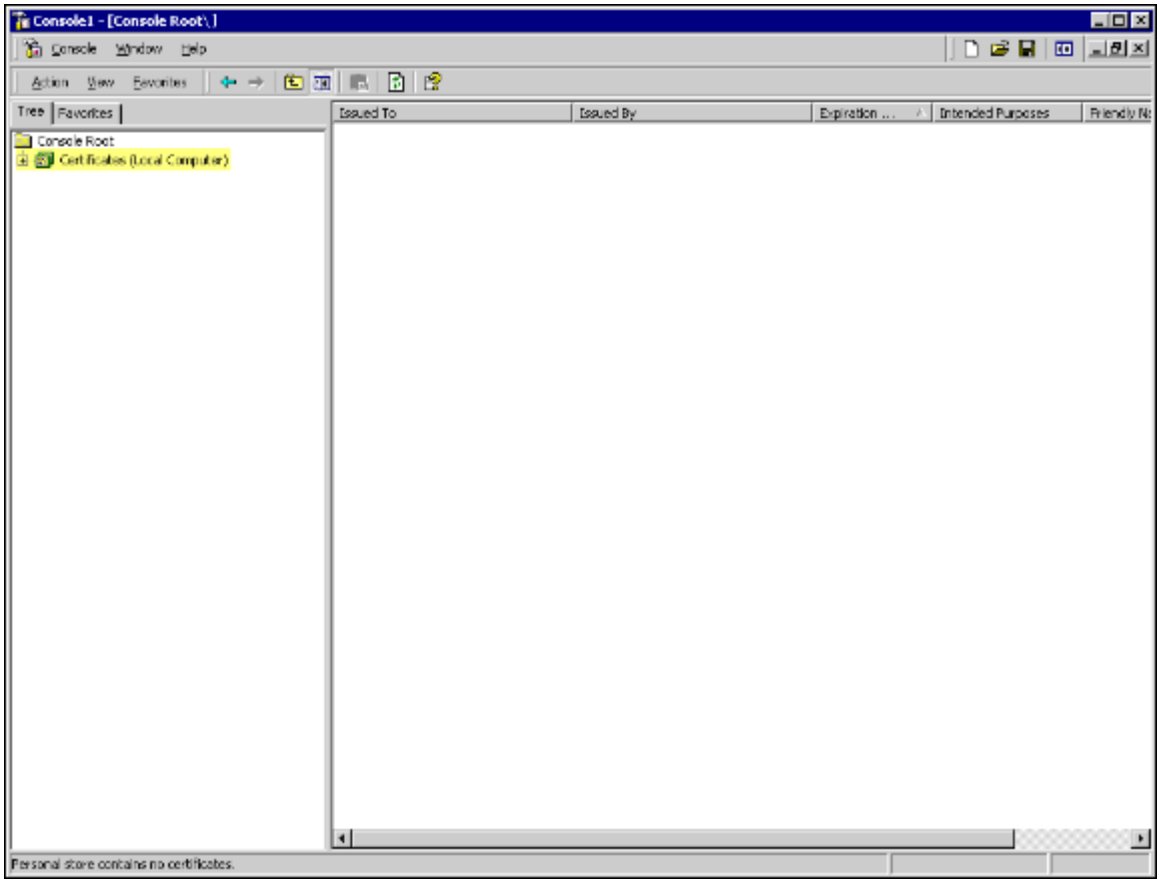
1. Select **Start - Run**....
2. In the **Run** dialog, enter `mmc` in the **Open** field.
3. Click **OK**.
4. In the Console1 window, select **Console - Add/Remove Snap-in**....
5. In the Add/Remove Snap-in dialog, click **Add**....
6. In the Add Standalone Snap-in dialog, select `Certificates` in the **Available Standalone Snap-ins** list.
7. Click **Add**.
8. In the Certificates snap-in panel of the certificates snapin wizard, select the **Computer account** option button.
9. Click **Next >**.
10. In the Select Computer panel of the certificates snapin wizard, click **Finish**.
11. In the Add Standalone Snap-in dialog, click **Close**.

A new snap-in, **Certificates (Local Computer)**, appears in the Add Standalone Snap-in dialog:



12. In the Add Standalone Snap-in dialog, click **OK**.

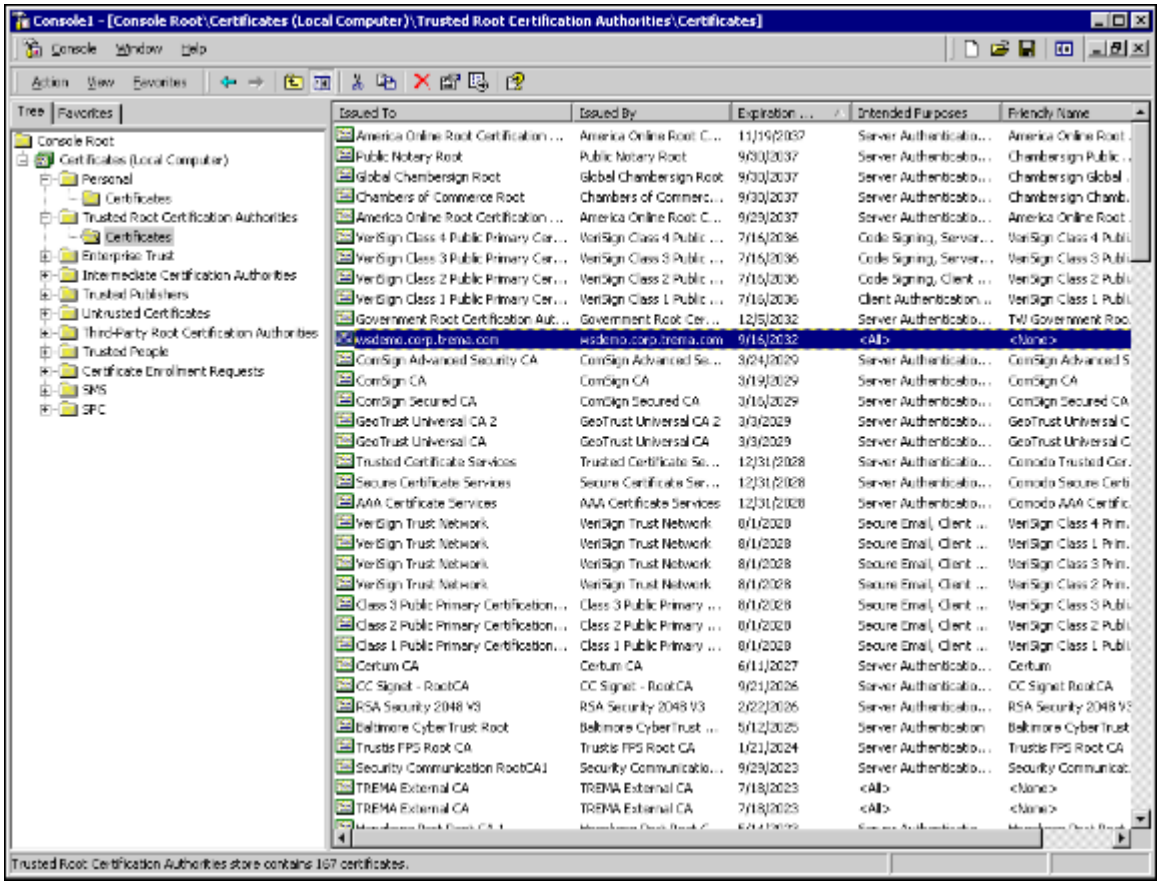
The **Console Root** folder in the left frame of the Console1 window contains a new child folder, **Certificates (Local Computer)**:



13. In the left frame, navigate to and right-click **Console Root\Certificates (Local Computer)\Personal\Certificates**. In the resulting popup menu, select **Import**....
14. In the first panel of the certificates import wizard, click **Next >**.
15. In the File to Import panel of the certificates import wizard, enter the path to the server certificate (for example, `C:\lhogD1\keystrokes\demoCA\server.p12`) in the **File name** field.
16. Click **Next >**.
17. In the Password panel of the certificates import wizard, enter the private key for the server certificate in the **Password** field.

    The default private key for the test certificate is `trema.ca`.
18. Click **Next >**.
19. In the Certificate Store panel of the certificates import wizard, select the **Place all certificates in the following folder** option button.
20. Click **Next >**.
21. In the Completing the Certificate Import Wizard panel of the certificates import wizard, click **Finish**.
22. In the Certificate Import Wizard dialog, click **OK**.

    The imported certificate displays in the right frame of the Console1 window:



23. In the left frame, navigate to and right-click **Console Root\Certificates (Local Computer)\ Trusted Root Certification Authorities\Certificates**. In the resulting popup menu, select **Import**....

24. In the first panel of the certificates import wizard, click **Next >**.

25. In the File to Import panel of the certificates import wizard, enter the patch to the server certificate in the **File name** field.

26. Click **Next >**.

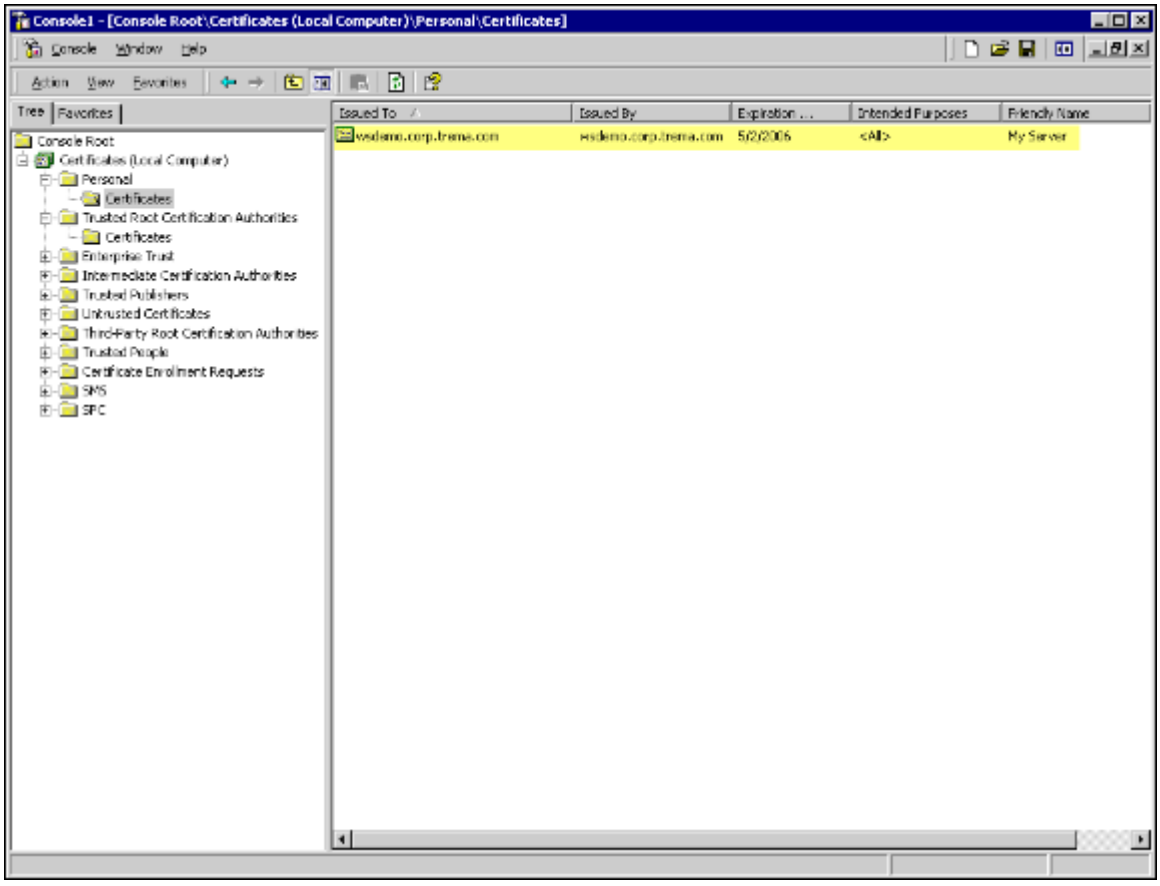
27. In the Certificate Store panel of the certificates import wizard, select the **Place all certificates in the following folder** option button.

28. Click **Next >**.

29. In the Completing the Certificate Import Wizard panel of the certificates import wizard, click **Finish**.

30. In the Certificate Import Wizard dialog, click **OK**.

    The imported certificate displays in the right frame of the Console1 window:



D.3.5.2 Assigning the server certificate

To assign the server certificate:

1. Open the Internet Information Services window (if it is not already opened).

For instructions on opening the Internet Information Services window, see *D.3.2 Creating a secure website* on page 424.

2. Right-click the secure website's icon in the left frame of the window and select **Properties** in the resulting popup menu.
3. In the [Secure website name] Properties dialog, click the **Directory Security** tab to open it.
4. In the Secure communications section of the [Secure website name] Properties dialog, click **Server Certificate**….
5. In the first panel of the web server certificate wizard, click **Next >**.
6. In the Server Certificate panel of the web server certificate wizard, select the **Assign an existing certificate** option button.
7. Click **Next >**.
8. In the Available Certificates panel of the web server certificate wizard, select the server certificate.
9. Click **Next >**.
10. In the Certificate Summary panel of the web server certificate wizard, click **Next >**.
11. In the Completing the Web Server Certificate Wizard panel of the web server certificate wizard, click **Finish**.
12. In the Secure communications section of the [Secure website name] Properties dialog, click **Edit**….
13. In the Secure Communications dialog, select the **Require secure channel (SSL)** checkbox.
14. Select the **Require client certificates** option button.
15. Click **OK**.
16. In the [Secure website name] Properties dialog, click **Apply**.
17. Click **OK**.

D.3.6 Configuring one-way SSL

This section documents the required steps to enable the IIS web server to proxy the client certificate to the BEA WebLogic server once the client has been successfully authenticated.

D.3.6.1 Configuring one-way SSL

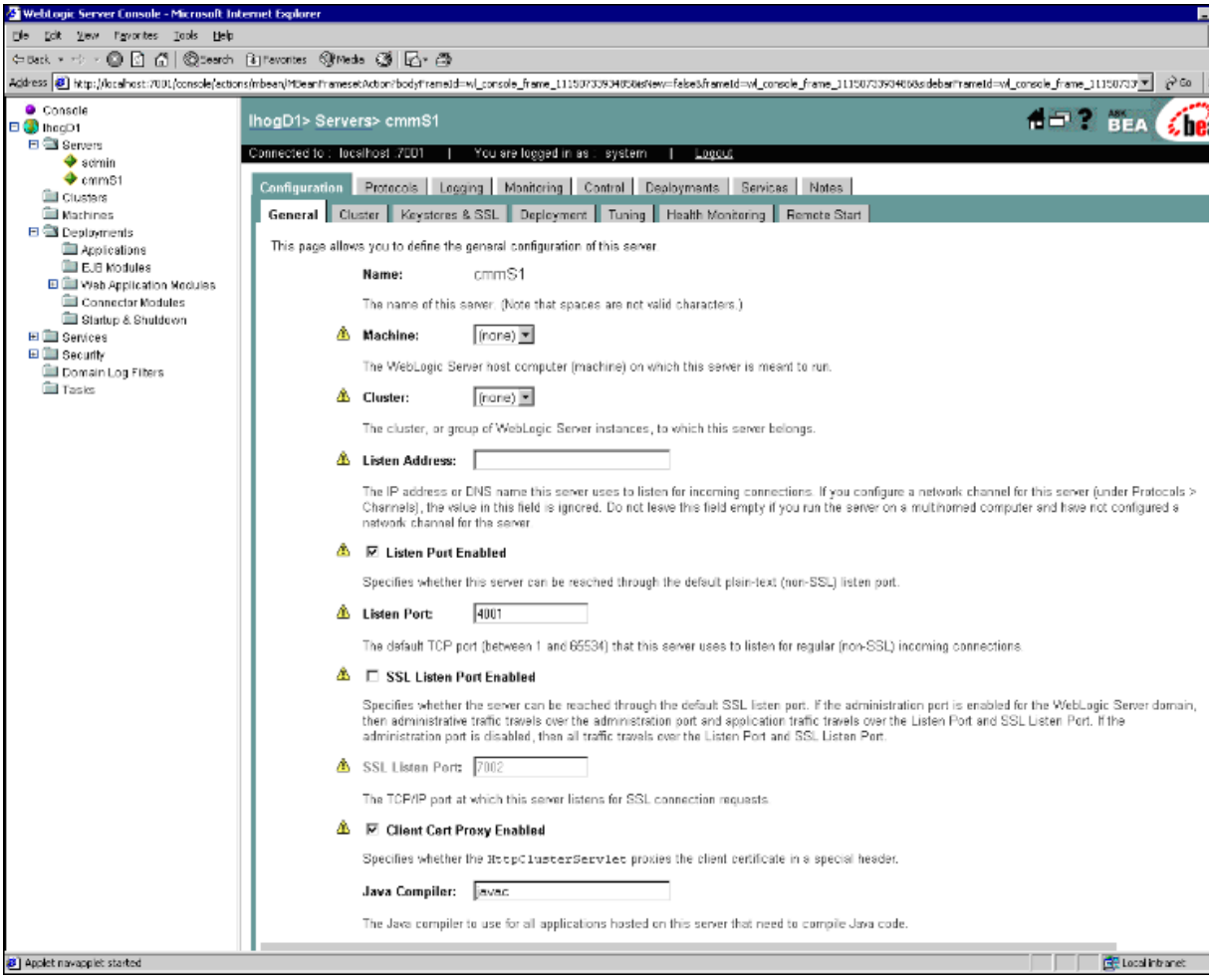
To configure one-way SSL:

1. Open the `iisproxy.ini` file in a text editor.
2. Edit the file to forward requests to the BEA WebLogic server using HTTP:

```
WebLogicHost=localhost
WebLogicPort=4001
WlForwardPath=/cmm
HungServerRecoverSecs=86400
Idempotent=OFF
Debug=ALL
```

In a production environment, you usually turn off the debug mode by removing `Debug=ALL`. (When you include `Debug=ALL`, the debugging information is written to the `/tmp/wlproxy.log` file in Unix systems and the `C:\tmp\wlproxy.log` file in Windows systems.)

3. Save and close the file.
4. Restart IIS.
5. Log into the BEA WebLogic Admin Console (available at `http://localhost:7001/console`) and enable the client certificate proxy.



For detailed instructions, see the BEA WebLogic documentation.

6. Restart BEA WebLogic.

D.4 Creating a secure website using Tomcat

This section documents the required steps to create a new SSL port in Tomcat for listening to the incoming requests to CMM web services.

Note: Adding a new SSL port is relatively simple in Tomcat compared to IIS. There is no need to set up a new Tomcat home. All you need to do is to add a connector to the `server.xml` file.

D.4.1 Creating a secure website using Tomcat

To create a secure website using Tomcat:

1. Navigate to …\tomcat.5.x.xx\conf\.
2. Open the `server.xml` file in a text editor.
3. Add the following connector to the file:

```
<!-- Sample SSL configuration using client authentication -->
<Connector
   port="8543"
   maxThreads="150"
   minSpareThreads="25"
   maxSpareThreads="75"
   enableLookups="false"
   disableUploadTimeout="true"
   acceptCount="100"
   debug="0"
   scheme="https"
   secure="true"
   clientAuth="true"
   sslProtocol="TLS"
   keystoreFile="C:/lhogD1/keystores/demoCA/server.p12"
   keystorePass="trema.ca"
   keystoreType="PKCS12"
   truststoreFile="C:/lhogD1/keystores/demoCA/cacerts.jks"
   truststorePass="trema.ca"
   truststoreType="JKS"
   algorithm="SunX509"/>
```

   Do the following:

   - Replace the SSL port so that it does not conflict with any existing ports.
   - Ensure the `clientAuth` attribute is set to `true`.
   - Replace the appropriate paths, types, and private keys for both the server certificate and CA public certificate.
4. Save and close the file.
5. Restart Tomcat.

D.5 Testing HTTPS configuration

This section documents the required steps to test HTTPS configuration with CMM PKI X509.

D.5.1 Importing client certificates to the web browser

To import client certificates to the web browser:

1. Navigate to the extracted folder of keystores.
2. Double-click …\demoCA\alterna.p12 to import the client certificate into the personal account.

   The default private key for the test certificate is `trema.ca`.

D.5.2 Configuring CMM with PKI X509

Note: The following steps are for a configuration based on the test certificate information. For the steps for a production configuration, contact Wallstreet.

To configure CMM with PKI X509:

1. Add the strong authentication service definition to …\CmmInstallationData\installation\appserver\service_definitions\CredentialsServiceDefinitions.xml:

```
<ServiceDefinition
   interface_classname="alterna.appserver.security.IaCredentialsAuthenticator"
   implementation_classname="alterna.appserver.security.CaSACredentialsAuthentic
   ator"
   name="IaCredentialsAuthenticator"
   registration_policy="append"/>
```

2. Configure the rules to validate the client certificate in …\CmmInstallationData\installation\appserver\authentication\strong_auth\sa_cert_config.xml:

```
<strong_authentication>
   <!-- Attributes that will be used in SA credential authenticator, mapping with
   the name of value element -->
   <mapping login_id="userid">
      <!-- the certificate must be an instance of java.security.cert.Certificate
      -->
      <certificate id="javax.servlet.request.X509Certificate"
      issuer_dn_pattern="(OU=C([a-zA-Z]*) A([a-zA-Z]*),)">
         <!-- Possible IDs: SubjectDN, IssuerDN, SigAlgName, SigAlgOID and
         SerialNumber -->
         <value name="userid" id="SubjectDN"
         classname="alterna.appserver.security.cert.CaX509CertificateStringValue
         Parser" pattern="=(\w*)@" is_required="true" />
         <!-- more value on different attributes of the certificate can be added
         on here -->
      </certificate>
      <!-- more certificates can be added on here -->
   </mapping>
</strong_authentication>
```

   Note:

   - The above setting is based on the issuer and subject of the client test certificate.
   - The `issuer_dn_pattern` attribute of the `certificate` element tells CMM to only accept the issuer having an organization unit (OU) with a first word starting with C and a second word starting with A (for example, Certificate Authorities):

     ```
     E = myCA@trema.com
     CN = wsdemo.corp.trema.com
     OU = Certificate Authorities
     O = Trema Laboratories Inc.
     L = Calgary
     S = Alberta
     C = CA
     ```

   - The `pattern` attribute in the `value` element tells CMM to extract anything after the equal sign (=) and before the at sign (@); this is usually the user ID in the e-mail address:

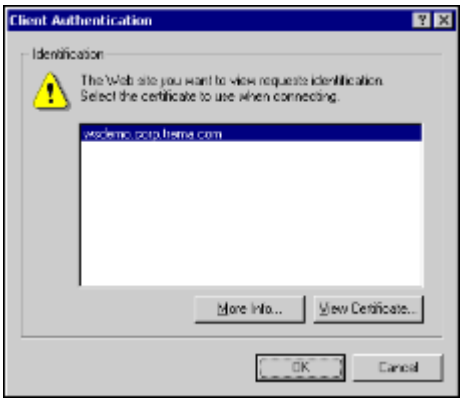
     ```
     E = alterna@trema.com
     CN = wsdemo.corp.trema.com
     OU = Client - Alterna
     O = Trema Laboratories Inc.
     S = Alberta
     ```

```
C = CA
```

3. Restart the cmmS1 service.

D.5.3 Testing CMM with PKI X509

To test CMM with PKI X509, navigate to CMM in a browser (for example, `https://wsdemo.corp.trema.com/cmm`). You may be prompted to select the client certificate for client authentication:



After you select the client certificate:

- If the client authentication was successful, you are automatically logged in to CMM.
- Otherwise, the default login page displays.

Note: This appendix assumes that an alterna user exists in the CMM database. If not, you can manually add one before continuing. Log into CMM using form-based authentication to create a new user.