NonStop SOAP 4.1 User's Manual

HP Part Number: 532118-006
Published: April 2014
Edition: J06.09 and subsequent J-series RVUs and H06.20 and subsequent H-series RVUs.

© Copyright 2010, 2014 Hewlett-Packard Development Company, L.P.
Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

Export of the information contained in this publication may require authorization from the U.S. Department of Commerce.

Microsoft, Windows, and Windows NT are U.S. registered trademarks of Microsoft Corporation.

Intel, Pentium, and Celeron are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Apache Axis2/C is maintained by the Apache Software Foundation and is released under the Apache 2.0 License http://www.apache.org/licenses/LICENSE-2.0.

Motif, OSF/1, UNIX, X/Open, and the "X" device are registered trademarks, and IT DialTone and The Open Group are trademarks of The Open Group in the U.S. and other countries.

Open Software Foundation, OSF, the OSF logo, OSF/1, OSF/Motif, and Motif are trademarks of the Open Software Foundation, Inc. OSF MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THE OSF MATERIAL PROVIDED HEREIN, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. OSF shall not be liable for errors contained herein or for incidental consequential damages in connection with the furnishing, performance, or use of this material.

© 1990, 1991, 1992, 1993 Open Software Foundation, Inc. The OSF documentation and the OSF software to which it relates are derived in part from materials supplied by the following: © 1987, 1988, 1989 Carnegie-Mellon University. © 1989, 1990, 1991 Digital Equipment Corporation. © 1985, 1988, 1989, 1990 Encore Computer Corporation. © 1988 Free Software Foundation, Inc. © 1987, 1988, 1989, 1990, 1991 Hewlett-Packard Company. © 1985, 1987, 1988, 1989, 1990, 1991, 1992 International Business Machines Corporation. © 1988, 1989 Massachusetts Institute of Technology. © 1988, 1989, 1990 Mentat Inc. © 1988 Microsoft Corporation. © 1987, 1988, 1989, 1990, 1991, 1992 SecureWare, Inc. © 1990, 1991 Siemens Nixdorf Informationssysteme AG. © 1986, 1989, 1996, 1997 Sun Microsystems, Inc. © 1989, 1990, 1991 Transarc Corporation.OSF software and documentation are based in part on the Fourth Berkeley Software Distribution under license from The Regents of the University of California. OSF acknowledges the following individuals and institutions for their role in its development: Kenneth C.R.C. Arnold, Gregory S. Couch, Conrad C. Huang, Ed James, Symmetric Computer Systems, Robert Elz. © 1980, 1981, 1982, 1983, 1985, 1986, 1987, 1988, 1989 Regents of the University of California.

Contents

About This Document....18
Supported Release Version Updates (RVUs)....18
Intended Audience....18
New and Changed Information....18
New and Changed Information in 532118–006 Edition....18
New and Changed Information in 532118–005 Edition....18
New and Changed Information in 532118–004 Edition....18
New and Changed Information in 532118–003 Edition....19
New and Changed Information in 532118–002 Edition....20
Document Organization....20
Notation Conventions....21
General Syntax Notation....21
Notation for Messages....23
Notation for Management Programming Interfaces....24
General Syntax Notation....24
Publishing History....26
HP Encourages Your Comments....26
1 Introduction to NonStop SOAP....27
Supported Standards....27
Features of NonStop SOAP....28
Architecture of NonStop SOAP....29
NonStop SOAP Server....30
CGI Interface....31
NonStop SOAP 4 Engine....31
TS/MP Message Receiver....31
XML Message Receiver....31
WS-Security Module....31
Transaction Module....31
NonStop SOAP Tools....31
SoapAdminCL Tool....31
WSDL2C Tool....31
WSDL2PWY Tool....31
NonStop SOAP Administration Utility....32
SOAP Clients....32
Request Processing in NonStop SOAP....32
Compatibility....34
Migration....34
2 Installing NonStop SOAP....36
Prerequisites....36
Setting Up NonStop SOAP on a NonStop System....36
Installing NonStop SOAP....36
Default location installation....36
Non-standard location installation....37
Using PINSTALL Utility in Guardian Environment....37
Using PAX Command in OSS Environment....37
Setting up the Deployment Environment....38
Running the Deployment Script....39
Modifying the local.config file....41
Tuning NonStop SOAP....42
Running NonStop SOAP....42

Accessing the echo service....43
Setting up Multiple NonStop SOAP Deployment Instances in a Single iTP WebServer....43
Deploying a Sample Web Service....44
Deploying the empdb Service....44
Setting up the empdb environment....45
Accessing the empdb service....48
Deploying the adminserver Service....50
Setting up the adminserver Environment....50
Accessing the adminserver Service....51
3 Migrating NonStop SOAP 3 Services to NonStop SOAP 4 or Higher Versions....52
Prerequisites....52
Migrating the NonStop SOAP 3 Services....52
Generating SOAP 4 Service Artifacts and Client Files from NonStop SOAP 3....52
Generating the SDL File from the NonStop SOAP 3 SDR File....53
Modifying the SDL File....54
Generating the HTML Client, WSDL, XML Schema, and services.xml Files to Deploy the Service....54
Migration Considerations....56
Accessing the Migrated Application....56
Migrating NonStop SOAP 3 Transactions....58
Migrating NonStop SOAP 3 User-exits....61
Migrating the NonStop SOAP 3 User-exit Samples....62
Migrating the pre_process() and post_process() user-exits using mod_empdb....62
Implementation of the pre_process() user-exit in NonStop SOAP 3....63
Implementation of the post_process()user-exit in NonStop SOAP 3....63
Migration of the pre_process() and post_process() user-exits in NonStop SOAP 4....64
Migrating the pre_service() and pre_marshal() user-exits in NonStop SOAP 3 to Message Receiver User Functions in NonStop SOAP 4.1....71
Implementation of the pre_service() and pre_marshal() user-exit in NonStop SOAP 3....71
Migration of pre_service() and pre_marshal() user-exits in NonStop SOAP 4.1....72
4 Getting Started with NonStop SOAP 4....77
Deploying a Pathway Application and a NonStop Process as a Web Service 77
Getting the sample application files from sample_services....78
Creating DDL dictionary files....78
Creating SDL files....79
Creating Web Service files using SOAPAdminCL tool....79
Compiling reflector.c....81
Configuring and starting the reflector application....81
Accessing the Web Service deployment....82
Deploying a NonStop process as a Web Service....83
5 NonStop SOAP 4 Service APIs....85
Basic NonStop SOAP 4 Service APIs....85
APIs to Implement the Service Interface with NonStop SOAP 4....85
The axis2_get_instance()Function....85
Creating the service skeleton structure....86
Setting the service skeleton operations structure....86
Returning the service skeleton structure to the service....86
The axis2_remove_instance()Function....86
APIs to Implement the Service Skeleton Structure Interface....87
The init Function....87
The invoke Function....87

The fault Function....88
The free Function....88
APIs to Extract the Input Parameters and Return the Response....89
Reading an AXIOM node using NonStop SOAP 4 APIs....89
The axiom_element_get_qname()Function....89
The axiom_node_get_data_element()Function....90
The axiom_attribute_get_qname()Function....90
The axiom_element_get_attribute()Function....90
Creating an AXIOM node using NonStop SOAP 4 APIs....91
Creating an AXIOM node using AXIOM Node Create APIs....91
Creating an AXIOM node from an XML using AXIOM Document APIs....93
APIs for Logging....95
Creating the log file structure....95
Developing NonStop SOAP 4 Services using Service APIs....95
Defining the XML Request and Response Payload....96
Creating the SDL File for the Service....98
Generating the WSDL File and the services.xml File....99
Generating the Service Skeleton Files....100
Implementing the Business Logic in the Service Skeleton Files....102
Compiling the Service Code and Deploying the Service....102
Testing the Service....103
6 NonStop SOAP 4 Client APIs....105
NonStop SOAP 4 Client APIs....105
APIs to Implement the Client Interface with NonStop SOAP 4....105
The axis2_svc_client_create()Function....106
The axis2_svc_client_free()Function....106
APIs to Generate the Request Node and Consume the Response Node....106
Generating the Request AXIOM node using NonStop SOAP 4 APIs....107
Creating an AXIOM node using AXIOM Node Create APIs....107
Creating an AXIOM node from an XML using AXIOM Document APIs....110
Consuming the Response AXIOM node using NonStop SOAP 4 APIs....111
The axiom_element_get_qname()Function....111
The axiom_node_get_data_element()Function....112
The axiom_attribute_get_qname()Function....112
The axiom_element_get_attribute()Function....112
APIs to Invoke the Web Service....113
The axis2_svc_client_send_receive() Function....113
The axis2_svc_client_send_receive_non_blocking() Function....113
The axis2_svc_client_send_robust() Function....114
The axis2_svc_client_fire_and_forget() Function....114
APIs for Logging....115
Creating the log file structure....115
Logging messages at different log levels....115
The axutil_log_impl_log_warning()Function....116
The axutil_log_impl_log_info() Function....116
The axutil_log_impl_log_user()Function....116
The axutil_log_impl_log_debug()Function....116
The axutil_log_impl_log_trace()Function....116
Releasing the log structure....116
ADB APIs for creating requests....117
The adb_<messagename>_create()Function....117
The adb_<messagename>_free()Function....117
The adb_<complextype>_create_with_values() Function....117
The adb_<messagename>_set_<complextype>() Function....118

The adb_<messagename>_get_<complextype>() Function....118
Developing NonStop SOAP 4 Clients Using Client APIs....119
Generating NonStop SOAP 4 Client Stubs using the WSDL2C tool....119
Implementing Business Logic in Client Stubs....121
Compiling the Client Code and Deploying the Client....121
Compiling the Client Code using the Makefile....121
Compiling the Client Code manually....122
Testing the Client....123
7 Customizing NonStop SOAP 4 Message Processing....124
Overview....124
Modules and Handlers....124
Message Receiver User Functions....125
NonStop SOAP 4 Message Processing Customization....125
Customizing the NonStop SOAP 4 Message Process Using Handlers....126
Request Processing in NonStop SOAP 4....127
Flows....127
Phases....128
Pre-defined phases in inflow....128
User-defined phases in inflow....129
Pre-defined phases in outflow....129
User-defined phases in outflow....130
Handlers....130
Modules....130
Creating a User-Defined Phase....131
Deploying and Attaching a Module....132
Developing a Sample Module for NonStop SOAP 4....132
Running the SoapAdminCL Tool to Generate the Module Handler Stub Files....133
Implementing the Business Logic in the Module Handler Stub Files....134
Implementing the invoke() Method for loggingInHandler....134
Implementing the invoke() Method for loggingOutHandler....134
Engaging the Module Handler at the Service Level....136
Verifying the Module Handler Output....137
Customizing the NonStop SOAP 4 Message Process Using Message Receiver User Functions....137
Configuring NonStop SOAP 4 Message Receiver User Functions....138
Configuring NonStop SOAP 4 Message Receiver User Functions at the service level....138
Configuring NonStop SOAP 4 Message Receiver User Functions at the global level....138
Modifying the Data Buffer Passed to the Service using NonStop SOAP 4 Message Receiver User Functions....139
Setting the pre_service and pre_marshal Message Receiver User Functions....139
Creating an instance of the Message Receiver User Function structure....139
Setting pre_service and pre_marshal function names....140
Implementing the pre_service and pre_marshal Message Receiver User Functions....141
Modifying the Pathway or Process Attributes using NonStop SOAP 4 Message Receiver User Functions....143
The axis2_msg_recv_get_pathmonName()Function....143
The axis2_msg_recv_get_serverclassName()Function....143
The axis2_msg_recv_get_serviceName()Function....144
The axis2_msg_recv_get_operationName()Function....144
The axis2_msg_recv_get_processName()Function....145
The axis2_msg_recv_set_pathmonName()Function....145
The axis2_msg_recv_set_serverclassName()Function....145
The axis2_msg_recv_set_serviceName()Function....146
The axis2_msg_recv_set_operationName()Function....146
The axis2_msg_recv_set_processName()Function....147

Modifying the Message Flow in the Pathway Message Receiver using NonStop SOAP 4 Message Receiver User Functions....147
The axis2_msg_recv_get_skipService()Function....147
The axis2_msg_recv_get_skipMarshal()Function....148
The axis2_msg_recv_set_skipService ( ) Function....148
The axis2_msg_recv_set_skipMarshal()Function....149
Developing Sample Message Receiver User Functions for NonStop SOAP 4....149
Running the SoapAdminCL Tool to Generate the Message Receiver User Functions Stub Files....150
Implementing the Business Logic in the Message Receiver User Functions Stub Files....150
Implementing the pre_service Message Receiver User Function....151
Implementing the pre_marshal Message Receiver User Function....151
Engaging the Message Receiver User Functions at the Service Level....151
Verifying the Message Receiver User Functions Output....152
8 NonStop SOAP 4 Service Description Language....153
SDL File Elements and Attributes....153
SDL Service Types....154
The Pathway Element....154
The PathwayEnvironment Element....154
The Service Element....155
The Operation Element....159
The Process Element....174
The ProcessEnvironment Element....174
The ProcessDetails Element....174
The ServerAPI Element....176
9 NonStop SOAP 4 Configuration Files....177
The itp_axis2.config File....177
Linking iTP WebServer to the NonStop SOAP 4 Deployment....177
Specifying iTP WebServer Filemap for NonStop SOAP 4....177
Defining the Log Levels of the NonStop SOAP 4 Server....178
Defining the Log File Size....180
Defining Separate Log and Trace Files for NonStop SOAP 4 Servers....180
The axis2.xml File....180
Setting the Message Receiver and the Message Exchange Pattern....180
Attaching a Module at the Global Level....181
Attaching Message Receiver User Functions at the Global Level....181
Specifying the Order of Phase Invocation in NonStop SOAP 4 Message Processing....182
The services.xml File....182
Updating the Service Parameters....183
Configuring a service as a Pathway application....183
Configuring a service as a process....183
Configuring a service as a DLL....183
Other Service Parameters....184
Defining Multiple SOAP Response Selection Criteria....186
Controlling TMF Transaction Support....188
Engaging a Module at the Service Level....189
Setting the Operation-Specific MEP....189
Setting the Operation-Specific Message Receiver....190
The module.xml File....190
Specifying the Module Name and Module Class File Name....191
Defining a New Handler and Specifying its Invocation Order....191
Defining the Module-Specific Parameters....192

10 NonStop SOAP Tools....194
The SoapAdminCL Tool....194
Generating NonStop SOAP 4 Files using the SDL File....194
Generating Module and Handler Stubs....199
Generating Message Receiver User Functions Stubs....200
Generating the NonStop SOAP 4 SDL File from the NonStop SOAP 3 SDR File....201
Additional Options....202
The WSDL2PWY Tool....203
Generating NonStop SOAP 4 Client Stubs....204
Generating Service Stubs....205
XML Schema and C Data Types Mapping....206
Migrating to Contract-First Services....208
WSDL2PWY Limitations....210
The WSDL2C Tool....211
Generating NonStop SOAP 4 Client Stubs....211
Generating NonStop SOAP 4 Service Skeleton Files....213
The NonStop SOAP 4 Administration Utility....215
Installing NonStop SOAP 4 Administration Utility....215
11 NonStop SOAP 4 Features....217
DDL Comments....217
Specifying DDL Fields as Optional....218
Exposing DDL Fields as XML Attributes....221
Setting the SoapDDLAttribute in the SDL File....222
Flagging a DDL Field with SOAP DDL Comment Tags....222
Specifying Base64 Encoding....225
Using the SOAP_OCCURS_DEP_ON DDL Comment Tag....226
Hot-Deployment of the NonStop SOAP 4 Server....229
Hot-Update for the Deployed Services....229
Internationalization and Encoding....230
Configuring the request encoding....230
Configuring the response encoding....230
Configuring the server encoding....231
Communicating with a Non-Pathway Process....231
Creating an SDL File to Communicate with a Non-Pathway Process....232
Creating a DDL File with the Request/Response Structures....232
Generating NonStop SOAP 4 Files using the SoapAdminCL Tool....232
Validation Module....233
SOAP_WSDL_NAME DDL Comment....233
Support for Multiple DDL Definitions....234
Check on SOAP Service Deployment....235
Unbounded data elements support....235
12 Transaction Management....236
Transaction Module Configuration....237
SOAP Server Transaction Management....238
SOAP Client Transaction Management....238
Begin a New Transaction....239
Continue a Transaction....241
Commit a Transaction....242
Abort a Transaction....244
Configuring the Level of Transaction Support....245
Transaction Timeouts....246
Configuring Transaction Timeout....247
Configuring Transactions to Abort on Fault....248
Session Management and Transactions....249

Begin a New Session....250
Begin Transaction within a Session....251
Continue a Transaction within a Session....253
Commit a Transaction within a Session....254
Abort a Transaction within a Session....255
End the Session....256
The Cookie File....258
Session Timeout....258
Subsessions....259
The SOAP_COOKIE_DELETION_INTERVAL Parameter....259
13 Using the Contract-First Approach in NonStop SOAP 4....260
NonStop SOAP 4 Tools for Developing Web Services Using the Contract-First Approach....260
WSDL Considerations....260
Using a Pre-defined WSDL File....260
Developing a NonStop SOAP 4 Pathway Web Service Using the WSDL2PWY Tool....262
Deploying a NonStop SOAP 4 Pathway Web Service....269
Accessing a NonStop SOAP 4 Pathway Web Service....269
Developing a NonStop SOAP 4 Non-Pathway Web Service Using the WSDL2C Tool....270
14 WS–Security in NonStop SOAP 4....271
Overview of Encryption and Signing....271
Supported WS–Security Features....273
Securing a NonStop SOAP 4 Service....274
Rampart Specific Assertions....275
Publishing the Security Requirements....277
Configuring the Client to Invoke a Secured Web Service....277
Configuring the Axis2c Client....277
Configuring Non–Axis2c Clients....278
Extensible Modules....278
Sample Programs....279
Functionality of the Sample Client and Server Program....280
Running the Web Services Security Sample Programs....280
WS-Security Scenarios....281
Scenario 1: Timestamp....281
Scenario 2: UsernameToken....281
Scenario 3: Encryption....282
Scenario 4: Signature....282
Scenario 5: Combining TimeStamp, UsernameToken, Encryption, and Signature with Protection order Sign->Encrypt....283
Scenario 6: Combining TimeStamp, UsernameToken, Encryption, and Signature with Protection Order Encrypt->Sign....283
Scenario 7: Symmetric Binding. Encryption using Derived Keys....283
Scenario 8: Symmetric Binding, Signature....283
Scenario 9: Symmetric Binding. Both Encryption and Signature with Protection Order Encrypt->Sign....283
Scenario 10: Symmetric Binding. Both Encryption and Signature with Protection Order Sign->Encrypt....283
Scenario 11: Symmetric Binding. Both Encryption and Signature with Protection Order Encrypt->Signature Encryption....283
Scenario 12: Symmetric Binding. Both Encryption and Signature with Protection Order Sign->Encrypt. Signature Encryption....283
Recommendations....284
A NonStop SOAP 4 Error and Warning Messages....285
NonStop SOAP 4 Error Messages....285

Client API Errors....285
NonStop SOAP 4 Engine Errors....286
NonStop SOAP 4 Utility Errors....290
Pathway Message Receiver Errors....290
CGI Errors....294
Transaction Management Errors....294
SoapAdminCL Errors....296
Pathway Service Definition Errors....296
Memory Allocation Errors....298
SoapAdminCL Error Messages....298
WSDL2PWY and WSDL2C Error Messages....303
NonStop SOAP 4 Warning Messages....304
B Install Files and Folders....308
Verifying the Extracted Product Files....308
Verifying the Deployed Files....312
C SoapAdminCL DDL datatype to XML datatype conversion....314
D NonStop SOAP 4 APIs....316
AXIOM APIs....316
Attributes....316
The axiom_attribute_create()Function....316
The axiom_attribute_free() Function....316
The axiom_attribute_get_qname() Function....317
The axiom_attribute_serialize() Function....317
The axiom_attribute_get_localname() Function....317
The axiom_attribute_get_value()Function....318
The axiom_attribute_get_namespace() Function....318
The axiom_attribute_set_localname() Function....318
The axiom_attribute_set_value() Function....319
The axiom_attribute_set_namespace() Function....319
Comment....320
The axiom_comment_create() Function....320
The axiom_comment_free() Function....320
The axiom_comment_get_value() Function....321
The axiom_comment_set_value() Function....321
Document....321
axiom_document_create....321
axiom_document_free....322
axiom_document_get_root_element....322
axiom_document_set_root_element....322
axiom_document_build_all....323
Element....323
axiom_element_create....323
axiom_element_create_with_qname....324
axiom_element_add_attribute....324
axiom_element_get_attribute....325
axiom_element_get_attribute_value....325
axiom_element_free....325
axiom_element_get_localname....326
axiom_element_set_localname....326
axiom_element_get_namespace....326
axiom_element_set_namespace....327
axiom_element_get_all_attributes....327
axiom_element_get_children....327

axiom_element_get_children_with_qname....328
axiom_element_remove_attribute....328
axiom_element_set_text....329
axiom_element_get_text....329
axiom_element_to_string....330
axiom_element_get_child_elements....330
axiom_element_extract_attributes....330
axiom_element_get_attribute_value_by_name....331
axiom_element_get_localname_str....331
axiom_element_set_localname_str....331
Namespace....332
axiom_namespace_create....332
axiom_namespace_free....332
axiom_namespace_equals....332
axiom_namespace_serialize....333
axiom_namespace_get_uri....333
axiom_namespace_get_prefix....334
axiom_namespace_to_string....334
axiom_namespace_create_str....334
axiom_namespace_set_uri_str....334
axiom_namespace_get_uri_str....335
axiom_namespace_get_prefix_str....335
Node....335
axiom_node_create....335
axiom_node_free_tree....336
....336
axiom_node_insert_sibling_after....336
axiom_node_insert_sibling_before....337
axiom_node_serialize....337
axiom_node_get_parent....338
axiom_node_get_first_child....338
axiom_node_get_first_element....338
axiom_node_get_last_child....339
axiom_node_get_previous_sibling 339
axiom_node_get_next_sibling....339
axiom_node_get_node_type....340
axiom_node_get_data_element....340
....340
axiom_node_to_string....341
Text....341
axiom_text_create....341
axiom_text_free....341
axiom_text_serialize....342
axiom_text_set_value....342
axiom_text_get_value....343
axiom_text_get_text....343
Client API Module....343
client service....343
axis2_svc_client_get_svc 343
axis2_svc_client_set_options 344
axis2_svc_client_get_options 344
axis2_svc_client_engage_module 344
axis2_svc_client_disengage_module 345
axis2_svc_client_add_header 345
axis2_svc_client_remove_all_headers 346

axis2_svc_client_fire_and_forget_with_op_qname....346
axis2_svc_client_fire_and_forget....346
axis2_svc_client_send_receive 347
axis2_svc_client_send_receive_with_op_qname....347
axis2_svc_client_send_receive_non_blocking_with_op_qname....348
axis2_svc_client_get_svc_ctx 348
axis2_svc_client_free 349
axis2_svc_client_create 349
Options....349
axis2_options_get_action....349
axis2_options_get_fault_to....350
axis2_options_get_from....350
axis2_options_get_transport_receiver....350
axis2_options_get_transport_in....351
axis2_options_get_transport_in_protocol....351
axis2_options_get_message_id....351
axis2_options_get_properties....352
axis2_options_get_property....352
axis2_options_get_property....352
axis2_options_get_reply_t....353
axis2_options_get_transport_out....353
axis2_options_get_sender_transport_protocol....353
xis2_options_get_soap_version_uri....354
axis2_options_get_timeout_in_milli_seconds....354
axis2_options_get_parent....354
axis2_options_set_parent....355
axis2_options_set_action....355
axis2_options_set_fault_to....355
axis2_options_set_from....356
axis2_options_set_to....356
axis2_options_set_transport_receiver....356
axis2_options_set_transport_in....357
axis2_options_set_transport_in_protocol....357
axis2_options_set_message_id....358
axis2_options_set_properties....358
axis2_options_set_property....358
axis2_options_set_reply_to....359
axis2_options_set_transport_out....359
axis2_options_set_sender_transport....360
axis2_options_set_soap_version_uri....360
axis2_options_set_timeout_in_milli_seconds....360
axis2_options_set_transport_info....361
axis2_options_get_manage_session....361
axis2_options_set_manage_session....362
axis2_options_set_msg_info_headers....362
axis2_options_get_msg_info_headers....362
axis2_options_get_soap_version....363
axis2_options_set_soap_version....363
axis2_options_get_soap_action....363
axis2_options_set_soap_action....364
axis2_options_free....364
axis2_options_set_http_headers....364
axis2_options_create....365
Context Hierarchy....365
message context....365

The axis2_msg_ctx_get_base()Function....365
The axis2_msg_ctx_get_parent()Function....365
The axis2_msg_ctx_set_parent()Function....366
The axis2_msg_ctx_free()Function....366
The axis2_msg_ctx_get_soap_envelope()Function....366
The axis2_msg_ctx_get_response_soap_envelope()Function....367
The axis2_msg_ctx_get_fault_soap_envelope()Function....367
The axis2_msg_ctx_set_msg_id()Function....367
The axis2_msg_ctx_get_msg_id()Function....368
The axis2_msg_ctx_get_server_side()Function....368
The axis2_msg_ctx_set_soap_envelope()Function....368
The axis2_msg_ctx_set_response_soap_envelope()Function....369
The axis2_msg_ctx_set_fault_soap_envelope()Function....369
The axis2_msg_ctx_set_message_id()Function....370
The axis2_msg_ctx_set_server_side()Function....370
The axis2_msg_ctx_get_msg_info_headers()Function....370
The axis2_msg_ctx_is_keep_alive()Function....371
The axis2_msg_ctx_set_keep_alive()Function....371
The axis2_msg_ctx_get_op_ctx()Function....371
The axis2_msg_ctx_get_svc_ctx_id()Function....372
The axis2_msg_ctx_set_svc_ctx_id()Function....372
The axis2_msg_ctx_get_conf_ctx()Function....373
The axis2_msg_ctx_get_svc_ctx() Function....373
Service Skeleton API....373
AXIS2_SVC_SKELETON_INIT 373
AXIS2_SVC_SKELETON_INIT_WITH_CONF....373
AXIS2_SVC_SKELETON_FREE....374
AXIS2_SVC_SKELETON_INVOKE....374
AXIS2_SVC_SKELETON_ON_FAULT....374
Utilities....374
Utilities For Logging....374
The axutil_log_impl_log_critical()Function....374
The axutil_log_impl_log_error()Function....375
The axutil_log_impl_log_warning() Function....375
The axutil_log_impl_log_info() Function....375
The axutil_log_impl_log_user()Function....376
The axutil_log_impl_log_debug()Function....376
The axutil_log_impl_log_trace()Function....376
The axutil_log_free()Function....377
The axutil_log_create()Function....377
The axutil_log_create_default()Function....377
Utilities For Error Reporting....378
The axutil_error_free() Function....378
The axutil_error_get_message() Function....378
The axutil_error_set_error_message() Function....378
The axutil_error_set_error_number() Function....379
The axutil_error_set_status_code() Function....379
The axutil_error_get_status_code() Function....379
Utilities For String Operations....379
The axutil_strdup()Function....380
The axutil_strndup()Function....380
The axutil_strmemdup()Function....380
The axutil_strcmp()Function....381
The axutil_strncmp()Function....381
The axutil_strlen()Function....381

The axutil_strcasecmp() Function....382
The axutil_strncasecmp() Function....382
The axutil_stracat() Function....382
The axutil_strcat() Function....383
The axutil_strstr() Function....383
The axutil_strcasestr() Function....383
The axutil_strchr() Function....384
The axutil_replace()Function....384
The axutil_strltrim()Function....385
The axutil_strrtrim() Function....385
The axutil_strtrim() Function....385
The axutil_string_replace()Function....386
The axutil_string_substring_starting_at()Function....386
The axutil_string_substring_ending_at()Function....386
The axutil_string_tolower() Function....387
The axutil_string_toupper() Function....387
Utilities For Array List....387
The axutil_array_list_create() Function....387
The axutil_array_list_size()Function....388
The axutil_array_list_is_empty() Function....388
The axutil_array_list_contains() Function....388
The axutil_array_list_add() Function....389
The axutil_array_list_add_at()Function....389
The axutil_array_list_remove() Function....390
The axutil_array_list_free() Function....390
Utilities For Hash Tables....390
The axutil_hash_first()Function....390
The axutil_hash_next()Function....390
The axutil_hash_count()Function....391
The axutil_hash_free()Function....391
The axutil_hash_contains_key()Function....391
The axutil_hash_make()Function....392
The axutil_hash_get()Function....392
The axutil_hash_set()Function....392
qname....393
The axutil_qname_create()Function....393
The axutil_qname_create_from_string()Function....393
The axutil_qname_free()Function....393
The axutil_qname_equals()Function....393
The axutil_qname_clone()Function....394
The axutil_qname_get_uri()Function....394
The axutil_qname_get_prefix()Function....394
The axutil_qname_get_localpart()Function....394
The axutil_qname_to_string()Function....395
parameter....395
The axutil_param_create()Function....395
....395
....395
....396
....396
....396
The axutil_param_set_locked() Function....396
The axutil_param_free()Function....397
The axutil_param_set_attributes()Function....397
....397

The axutil_param_set_value_free()Function.....397
E Transaction Management Model in NonStop SOAP 3 and NonStop SOAP 4..398
Glossary.....399
Index.....405

Figures

1 NonStop SOAP Architecture....30
2 SOAP Clients and their Capabilities....32
3 The SOAP Message Format....33
4 The SOAP Message Format (in XML format)....33
5 NonStop SOAP 4 Request Processing Flow for TS/MP or NonStop process-based Web Services....34
6 NonStop SOAP 4 Installation and Deployment Environment....38
7 Processing Logic in NonStop SOAP 3 and NonStop SOAP 4.1....53
8 Stylesheet Based Formatted Output....70
9 Web Service deployment of NonStop Process/Pathway Application....77
10 Client APIs in NonStop SOAP 4....105
11 Customizing NonStop SOAP 4....124
12 Modified Message Process in NonStop SOAP 4....126
13 Message Process in NonStop SOAP 4....127
14 Files Generated by the `SoapAdminCL` Tool....195
15 Files Generated by the `WSDL2PWY` Tool....203
16 Files Generated by the `WSDL2C` Tool....211
17 Beginning a New Transaction....240
18 Beginning a New Session....250
19 272
20 272
21 273

Tables

1 Transaction Management Parameters in NonStop SOAP 3 and NonStop SOAP 4.1....61
2 User-exits Mapped with Equivalent Functions in NonStop SOAP 4.1....62
3 C89 Compiler Options....122
4 The DDL Comments values....164
5 Conversion Operator Values Table....172
6 Comparison Operator Values Table....173
7 Logging Levels....178
8 Valid AlignmentRule setting for existing C services that use DDL generated header files....184
9 Valid AlignmentRule setting for existing C services that does not use DDL generated header files....184
10 Valid AlignmentRule setting for existing COBOL services that use DDL generated header files....185
11 Valid AlignmentRule setting for existing COBOL services that does not use DDL generated header files....185
12 Displaying `BLANK` element in response XML....185
13 Location of Files Generated by the `SoapAdminCL` Tool....196
14 XML schema to C data type mapping....206
15 XML schema definitions, which deviate from WSDL specification....210
16 Comment Tags....217
17 SDL Values and DDL Comment Tags....225
18 Attributes in the Transaction Header Block....239
19 Transaction Timeout....247
20 Session Header Attributes....249
21 Unsupported WSDL File Elements....261
22 WSDL File Elements with Limited Support....261
23 Rampart Specific Assertions....276

24 WS-Security Sample Programs Directory....279
25 SoapAdminCL conversion table for DDL datatype to XML datatype Conversion:....314
26 Transaction Management Models in NonStop SOAP 3 and NonStop SOAP 4....398

Examples

1 A Sample itp_axis2.config Configuration File....179
2 A Sample services.xml Configuration File....189
3 A Sample module.xml File Structure....190
4 A Sample module.xml Configuration File....192
5 A Sample DDL file with the @SOAP_OPTIONAL Tag....219
6 An XSD Schema for a Sample DDL file with the @SOAP_OPTIONAL Tag 220
7 An XML Message-I for a Sample DDL file with the @SOAP_OPTIONAL Tag....221
8 An XML Message-II for a Sample DDL file with the @SOAP_OPTIONAL Tag....221
9 A Sample XML file with DDL fields mapped to the Corresponding Elements....221
10 A Sample XML file with DDL fields represented as XML attributes....221
11 A Sample SDL File with SoapDDLAttribute....222
12 A Sample DDL File with the @SOAP_ATTRIBUTE and @SOAP_ELEMENT Tag....224
13 An XML Message for a Sample DDL file with the @SOAP_ATTRIBUTE and @SOAP_ELEMENT Tag....224
14 A Sample DDL file with the @SOAP_BASE64 Tag....226
15 An XML Schema for a Sample DDL file with the @SOAP_BASE64 Tag....226
16 XML Message for a DDL file with the @SOAP_BASE64 Tag....226
17 A Sample DDL file with the @SOAP_OCCURS_DEP_ON Tag....227
18 A Sample DDL file with the @SOAP_OCCURS_DEP_ON Tag....228
19 An XML Schema for a Sample DDL file with the @SOAP_OCCURS_DEP_ON Tag....228
20 The SOAP Message Body for a Sample DDL file with the @SOAP_OCCURS_DEP_ON Tag....229
21 Begin a New Transaction....241
22 Continue a Transaction....242
23 Commit a Transaction....243
24 Abort a Transaction....244
25 Begin a New Session....251
26 Begin a New Transaction within a Session....252
27 Begin a New Session and Transaction Combined....253
28 Continue a Transaction....254
29 Commit a Transaction within a Session....255
30 Abort a Transaction within a Session....256
31 End a Session....257
32 Commit a Transaction and End a Session....258

About This Document

This manual describes how to install, configure, and use NonStop SOAP 4.1.

Supported Release Version Updates (RVUs)

This manual supports J06.09 and all subsequent J-series RVUs and H06.20 and all subsequent H-series RVUs until otherwise indicated in a replacement publication.

Intended Audience

This manual is intended for customers who wish to develop Web applications deployed in NonStop SOAP 4.

New and Changed Information

New and Changed Information in 532118–006 Edition

- Removed sample codes by providing references to the samples in the installation directory.
- Updated "Installing NonStop SOAP" (page 36) section to include information about multiple installations.
- Updated the declaration of AXIS2C_HOME locations throughout the manual in relevant sections.
- Modified WSDL2C tool options in the "The `WSDL2C` Tool" (page 211) section.
- Added a note to the "The `ResponseSelection` Element" (page 171) section.

New and Changed Information in 532118–005 Edition

- Added additional SOAPAdminCL `[-w]` option [-w]

New and Changed Information in 532118–004 Edition

- Added a section that explains SOAP_WSDL_NAME DDL Comment (page 233)
- Added a section that explains Support for Multiple DDL Definitions (page 234)
- Added additional `SoapAdminCL` Error Messages (page 298)
- Added a section that explains WS–Security in NonStop SOAP 4 (page 271)
- Added the following sub sections in the WS-Security chapter:
  - Added a section that explains Supported WS–Security Features (page 273)
  - Added a section that explains Securing a NonStop SOAP 4 Service (page 274)
  - Added a section that explains Rampart Specific Assertions (page 275)
  - Added a section that explains Publishing the Security Requirements (page 277)
  - Added a section that explains Configuring the Client to Invoke a Secured Web Service (page 277)
  - Added a section that explains Extensible Modules (page 278)
  - Added a section that explains Sample Programs (page 279)
  - Added a section that explains WS-Security Scenarios (page 281)

- Added a section that explains Recommendations (page 284)
- Added additional SOAPAdminCL `[-w]` option [-w]

New and Changed Information in 532118–003 Edition

- Added the Other Service Parameters (page 184) section containing descriptions for `DDLMapping`, `AlignmentRule` and `responseType` service parameters.
- Updated the examples in the following sections:
  - Defining the XML Request and Response Payload (page 96)
  - Testing the Service (page 103)
  - The `Service` Element (page 155)
  - The `ProcessDetails` Element (page 174)
  - DDL Comments (page 217)
  - Specifying DDL Fields as Optional (page 218)
  - Flagging a DDL Field with SOAP DDL Comment Tags (page 222)
  - Specifying `Base64` Encoding (page 225)
  - Using the `SOAP_OCCURS_DEP_ON` DDL Comment Tag (page 226)
- Modified sections throughout the manual for better readability.
- Added the appendices, "`SoapAdminCL` DDL datatype to XML datatype conversion" (page 314), "NonStop SOAP 4 APIs" (page 316) and "Transaction Management Model in NonStop SOAP 3 and NonStop SOAP 4" (page 398).
- Added the "Defining the Log File Size" (page 180) and "Defining Separate Log and Trace Files for NonStop SOAP 4 Servers" (page 180) sections containing information to configure the log file size and specify separate log files for NonStop SOAP 4 servers.
- Added the "Hot-Update for the Deployed Services" (page 229) section.
- Updated the "Hot-Deployment of the NonStop SOAP 4 Server" (page 229) section
- Added the "Unbounded data elements support" (page 235) appendix explaining the unbounded data elements feature support.
- Added the "Validation Module" (page 233) section explaining the validation module feature.
- Updated the "Internationalization and Encoding" (page 230) section.
- Added the "`WSDL2PWY and WSDL2C` Error Messages" (page 303) section listing the errors returned by `WSDL2PWY` and `WSDL2C` utilities.
- Added the "Migration Considerations" (page 56) section listing the differences between NonStop SOAP 3 and NonStop SOAP 4 for migration.
- Updated the explanation for `defaultResp` attribute in "The `ResponseSelection` Element" (page 171)
- Added the ADB APIs "The `adb_<messagename>_create()` Function" (page 117), "The `adb_<messagename>_free()` Function" (page 117), "The `adb_<complextype>_create_with_values()` Function" (page 117), "The `adb_<messagename>_set_<complextype>()` Function" (page 118) and "The `adb_<messagename>_get_<complextype>()` Function" (page 118)

New and Changed Information in 532118–002 Edition

- Updated information about the procedure for "Installing NonStop SOAP" (page 36)
- Updated one of the prerequisites "Prerequisites" (page 36) to install NonStop SOAP 4 on NonStop system.
- Updated information about "Running the Deployment Script" (page 39) procedure.
- Updated information about the procedure to set up empdb environment on "Setting up the `empdb` environment" (page 45).
- Updated information about the procedure to deploy a Pathway application and a NonStop process as a Web service (page 77)

Document Organization

This document is organized as follows:

Chapter 1: Introduction to NonStop SOAP	This chapter describes supported standards, features, architecture, and components of NonStop SOAP 4.1.
Chapter 2: Installing NonStop SOAP	This chapter describes the procedure to install NonStop SOAP 4 on the NonStop system. It also describes the steps to deploy a sample Web service and the `adminserver` service, which helps you to verify if you have successfully installed NonStop SOAP 4 on the NonStop system.
Chapter 3: Migrating NonStop SOAP 3 Services to NonStop SOAP 4 or Higher Versions	This chapter describes the procedure to migrate the existing NonStop SOAP 3 services, transactions, and user-exits to NonStop SOAP 4.
Chapter 4: Getting Started with NonStop SOAP 4	This chapter describes the procedure to deploy a Web service as a Pathway application and a NonStop process.
Chapter 5: NonStop SOAP 4 Service APIs	This chapter describes the NonStop SOAP 4 application programming interfaces you can use to develop a DLL-based Web service deployed in NonStop SOAP 4.
Chapter 6: NonStop SOAP 4 Client APIs	This chapter describes the NonStop SOAP 4 APIs you can use to develop client applications that consume SOAP Web services.
Chapter 7: Customizing NonStop SOAP 4 Message Processing	This chapter provides information on customizing the default NonStop SOAP 4 message process using Phases, Modules, Handlers, and Message Receiver User Functions.
Chapter 8: NonStop SOAP 4 Service Description Language	This chapter describes the SDL file and its elements:
Chapter 9: NonStop SOAP 4 Configuration Files	This chapter describes the NonStop SOAP 4 configuration files, namely, the `itp_axis2.config` file, the `axis2.xml`file, the `services.xml` file, and the `module.xml` file.
Chapter 10: NonStop SOAP Tools	This chapter describes the `SoapAdminCL` tool, the `WSDL2PWY` tool, and the `WSDL2C` tool that is available with the NonStop SOAP 4 distribution. It also provides and overview of the NonStop SOAP 4 Administration Utility.
Chapter 11: NonStop SOAP 4 Features	This chapter provides information about NonStop SOAP 4 features, namely, DDL comments, hot-deployment of the NonStop SOAP 4 server, internationalization and encoding, and communicating with a non-Pathway process.
Chapter 12: Transaction Management	This chapter discusses transaction management in NonStop SOAP 4.
Chapter 13: Using the Contract-First Approach in NonStop SOAP 4	This chapter

	the contract (WSDL file) is defined first, which states the type of service interface description that the service expects. The service code is generated based on this information.
Chapter 14: WS–Security in NonStop SOAP 4	This chapter provides a brief introduction to WS-Security and describes how WS-Security can be used in NonStop SOAP 4.
Appendix A: NonStop SOAP 4 Error and Warning Messages	This appendix provides cause, effect, and recovery of NonStop SOAP 4 error and warning messages.

Notation Conventions

General Syntax Notation

This list summarizes the notation conventions for syntax presentation in this manual.

UPPERCASE LETTERS

Uppercase letters indicate keywords and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required. For example:

MAXATTACH

Italic Letters

Italic letters, regardless of font, indicate variable items that you supply. Items not enclosed in brackets are required. For example:

file-name

Computer Type

Computer type letters indicate:

- C and Open System Services (OSS) keywords, commands, and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required. For example:

  Use the `cextdecs.h` header file.

- Text displayed by the computer. For example:

  ```
  Last Logon: 14 May 2006, 08:02:23
  ```

- A listing of computer code. For example

  ```
  if (listen(sock, 1) < 0)
  {
  perror("Listen Error");
  exit(-1);
  }
  ```

Bold Text

Bold text in an example indicates user input typed at the terminal. For example:

```
ENTER RUN CODE

?123
CODE RECEIVED:      123.00
```

The user must press the **Return** key after typing the input.

[ ] Brackets

Brackets enclose optional syntax items. For example:

```
TERM [\system-name.]$terminal-name

INT[ERRUPTS]
```

A group of items enclosed in brackets is a list from which you can choose one item or none. The items in the list can be arranged either vertically, with aligned brackets on each side of

the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
FC [ num  ]
   [ -num ]
   [ text ]

K [ X | D ] address
```

{ } Braces

A group of items enclosed in braces is a list from which you are required to choose one item. The items in the list can be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
LISTOPENS PROCESS { $appl-mgr-name }
                  { $process-name  }

ALLOWSU { ON | OFF }
```

| Vertical Line

A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
INSPECT { OFF | ON | SAVEABEND }
```

… Ellipsis

An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed sequence of syntax items any number of times. For example:

```
M address [ , new-value ]…

 - ] {0|1|2|3|4|5|6|7|8|9}…
```

An ellipsis immediately following a single syntax item indicates that you can repeat that syntax item any number of times. For example:

```
"s-char…"
```

Punctuation

Parentheses, commas, semicolons, and other symbols not previously described must be typed as shown. For example:

```
error := NEXTFILENAME ( file-name ) ;

LISTOPENS SU $process-name.#su-name
```

Quotation marks around a symbol, such as a bracket or brace indicate the symbol is a required character that you must type as shown. For example:

```
"[" repetition-constant-list "]"
```

Item Spacing

Spaces shown between items are required unless one of the items is a punctuation symbol, such as a parenthesis or a comma. For example:

```
CALL STEPMOM ( process-id ) ;
```

If there is no space between two items, spaces are not permitted. In this example, no spaces are permitted between the period and any other items:

```
$process-name.#su-name
```

Line Spacing

If the syntax of a command is too long to fit on a single line, each continuation line is indented three spaces and is separated from the preceding line by a blank line. This spacing distinguishes items in a continuation line from items in a vertical list of selections. For example:

```
ALTER [ / OUT file-spec / ] LINE

   [ , attribute-spec ]…
```

!i and !o

In procedure calls, the !i notation follows an input parameter (one that passes data to the called procedure); the !o notation follows an output parameter (one that returns data to the calling program). For example:

```
CALL CHECKRESIZESEGMENT (  segment-id                    !i
                         , error        ) ;              !o
```

!i,o

In procedure calls, the !i,o notation follows an input/output parameter (one that both passes data to the called procedure and returns data to the calling program). For example:

```
error := COMPRESSEDIT ( filenum ) ;                      !i,o
```

!i:i

In procedure calls, the !i:i notation follows an input string parameter that has a corresponding parameter specifying the length of the string in bytes. For example:

```
error := FILENAME_COMPARE_ (  filename1:length           !i:i
                            , filename2:length ) ;       !i:i
```

!o:i

In procedure calls, the !o:i notation follows an output buffer parameter that has a corresponding input parameter specifying the maximum length of the output buffer in bytes. For example:

```
error := FILE_GETINFO_ (  filenum                        !i
                        , [ filename:maxlen ] ) ;        !o:i
```

Notation for Messages

This list summarizes the notation conventions for the presentation of displayed messages in this manual.

Bold Text

Bold text in an example indicates user input typed at the terminal. For example:

```
ENTER RUN CODE

?123
CODE RECEIVED:      123.00
```

The user must press the Return key after typing the input.

Nonitalic Text

Nonitalic letters, numbers, and punctuation indicate text that is displayed or returned exactly as shown. For example:

```
Backup Up.
```

Italic Text

Italic text indicates variable items whose values are displayed or returned. For example:

```
p-register

process-name
```

[ ] Brackets

Brackets enclose items that are sometimes, but not always, displayed. For example:

```
Event number = number [ Subject = first-subject-value ]
```

A group of items enclosed in brackets is a list of all possible items that can be displayed, of which one or none might actually be displayed. The items in the list can be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
proc-name trapped [ in SQL | in SQL file system ]
```

{ } Braces

A group of items enclosed in braces is a list of all possible items that can be displayed, of which one is actually displayed. The items in the list can be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
obj-type obj-name state changed to state, caused by
{ Object | Operator | Service }

process-name State changed from old-objstate to objstate
{ Operator Request. }
{ Unknown.          }
```

| Vertical Line

A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
Transfer status: { OK | Failed }
```

% Percent Sign

A percent sign precedes a number that is not in decimal notation. The % notation precedes an octal number. The %B notation precedes a binary number. The %H notation precedes a hexadecimal number. For example:

```
%005400

%B101111

%H2F

P=%p-register E=%e-register
```

Notation for Management Programming Interfaces

This list summarizes the notation conventions used in the boxed descriptions of programmatic commands, event messages, and error lists in this manual.

UPPERCASE LETTERS

Uppercase letters indicate names from definition files. Type these names exactly as shown. For example:

```
ZCOM-TKN-SUBJ-SERV
```

lowercase letters

Words in lowercase letters are words that are part of the notation, including Data Definition Language (DDL) keywords. For example:

```
token-type
```

!r

The !r notation following a token or field name indicates that the token or field is required. For example:

```
ZCOM-TKN-OBJNAME      token-type ZSPI-TYP-STRING.          !r
```

!o

The !o notation following a token or field name indicates that the token or field is optional. For example:

```
ZSPI-TKN-MANAGER      token-type ZSPI-TYP-FNAME32.         !o
```

General Syntax Notation

This list summarizes the notation conventions for syntax presentation in this manual.

UPPERCASE LETTERS

Uppercase letters indicate keywords and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required. For example:

```
SELECT
```

Italic Letters

Italic letters, regardless of font, indicate variable items that you supply. Items not enclosed in brackets are required. For example:

```
file-name
```

Computer Type

Computer type letters within text indicate case-sensitive keywords and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required. For example:

```
myfile.sh
```

Bold Text

Bold text in an example indicates user input typed at the terminal. For example:

```
ENTER RUN CODE

?123
CODE RECEIVED:      123.00
```

The user must press the Return key after typing the input.

[ ] Brackets

Brackets enclose optional syntax items. For example:

```
DATETIME [start-field TO] end-field
```

A group of items enclosed in brackets is a list from which you can choose one item or none. The items in the list can be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
DROP SCHEMA schema [CASCADE]
                   [RESTRICT]

DROP SCHEMA schema [ CASCADE | RESTRICT ]
```

{ } Braces

Braces enclose required syntax items. For example:

```
FROM { grantee[, grantee]...}
```

A group of items enclosed in braces is a list from which you are required to choose one item. The items in the list can be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
INTERVAL { start-field TO end-field }
         { single-field }

INTERVAL { start-field TO end-field  | single-field }
```

| Vertical Line

A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
{expression | NULL}
```

... Ellipsis

An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed sequence of syntax items any number of times. For example:

```
ATTRIBUTE[S] attribute [, attribute]...

{, sql-expression}...
```

An ellipsis immediately following a single syntax item indicates that you can repeat that syntax item any number of times. For example:

expression-n...

Punctuation

Parentheses, commas, semicolons, and other symbols not previously described must be typed as shown. For example:

```
DAY (datetime-expression)
@script-file
```

Quotation marks around a symbol, such as a bracket or brace indicate the symbol is a required character that you must type as shown. For example:

```
"{" module-name [, module-name]... "}"
```

Item Spacing

Spaces shown between items are required unless one of the items is a punctuation symbol, such as a parenthesis or a comma. For example:

```
DAY (datetime-expression)
DAY(datetime-expression)
```

If there is no space between two items, spaces are not permitted. In this example, no spaces are permitted between the period and any other items:

```
myfile.sh
```

Line Spacing

If the syntax of a command is too long to fit on a single line, each continuation line is indented three spaces and is separated from the preceding line by a blank line. This spacing distinguishes items in a continuation line from items in a vertical list of selections. For example:

```
match-value [NOT] LIKE pattern

   [ESCAPE esc-char-expression]
```

Publishing History

Part Number	Product Version	Publication Date
532118-001	N.A.	February 2010
532118–002	SOAP 4	May 2011
532118–003	SOAP 4	March 2012
532118–004	SOAP 4.1	February 2013
532118–005	SOAP 4.1	April 2013
532118–006	SOAP 4.1	April 2014

HP Encourages Your Comments

HP encourages your comments concerning this document. We are committed to providing documentation that meets your needs. Send any errors found, suggestions for improvement, or compliments to **docsfeedback@hp.com**.

Include the document title, part number, and any comment, error found, or suggestion for improvement you have concerning this document.

1 Introduction to NonStop SOAP

NonStop SOAP is a product that runs on an HP NonStop system and enables a Simple Object Access Protocol (SOAP) client to communicate with the Pathway services and NonStop processes.

NonStop SOAP is based on Apache Axis2/C version 1.5.0. Apache Axis2/C is a Web services engine developed and maintained by the Apache Software Foundation. For information about the Apache Axis2/C, see http://ws.apache.org/axis2/c/.

Along with the features provided by Apache Axis2/C, NonStop SOAP customizes Apache Axis2/C to provide the following additional features:

- Expose NonStop legacy applications as Web services
- Create SOAP protocol-based Web services that run on the NonStop platform
- Create SOAP protocol-based clients that can communicate with SOAP protocol-based services

The NonStop SOAP server implements the SOAP protocol that enables objects running on diverse platforms to communicate with Pathway applications or NonStop processes. It enables you to develop applications where the participating entities can reside on different systems and be implemented using different programming languages. As a result, NonStop SOAP exposes NonStop applications as Web services that can participate in a service-oriented architecture (SOA) environment.

The NonStop SOAP key components are:

- NonStop SOAP 4 server – the SOAP server
- `SoapAdminCL` tool – the SOAP server administration tool
- `WSDL2PWY` tool – code-generation tool for Pathway services
- `WSDL2C` tool – code-generation tool for non-Pathway services
- Client and server application programming interface (API) library – collection of C language API calls that enable you to use NonStop SOAP 4 features.

Supported Standards

The tools provided with NonStop SOAP produce artifacts and code that complies with the following standards:

- XML version 1.0
- SOAP version 1.1 and 1.2
- XML schema 1.0
- JavaScript version 1.3
- HTTP version 1.1
- HTML version 3.2
- WSDL specification 1.1
- SOAP attachments support implemented using handlers

Features of NonStop SOAP

The key features of NonStop SOAP are:

- ## Support for SOAP 1.1 and SOAP 1.2 standards

  NonStop SOAP implements the World Wide Web Consortium (W3C) SOAP 1.1 and SOAP 1.2 standards.

- ## Support for WS standards

  NonStop SOAP implements the following WS standards:

  - WS-Policy
  - WS-Security

  For more information on NonStop SOAP support for WS standards, see "WS–Security in NonStop SOAP 4" (page 271) .

- ## Tools for the service-first development approach

  NonStop SOAP provides the `SoapAdminCL` tool that implements the service-first approach for developing TS/MP applications. For more information on using the `SoapAdminCL` tool, see "NonStop SOAP Tools" (page 194).

- ## Tools for the contract-first development approach

  NonStop SOAP provides the `WSDL2C` and `WSDL2PWY` tools that support the contract-first approach for developing TS/MP and non-TS/MP Web services. For more information on using the `WSDL2C` and `WSDL2PWY` tools, see "NonStop SOAP Tools" (page 194).

- ## Tools for SOAP client development

  NonStop SOAP 4 provides the `WSDL2C` tool that helps generation of SOAP clients in the C programming language. For more information on building SOAP clients, see "NonStop SOAP 4 Client APIs" (page 105) .

- ## Transaction management

  NonStop SOAP 4 provides transaction management support for Pathway applications exposed as Web services using Transaction Management Facility (TMF). For more information on transaction management in NonStop SOAP 4, see "Transaction Management" (page 236).

- ## Support for TS/MP and non-TS/MP applications

  NonStop SOAP 4 enables you to expose the following applications as Web services:

  - Pathway application written in C or COBOL programming language
  - NonStop process-based application written in C or COBOL programming language
  - DLL-based applications written in the C programming language

- ## Powerful modular architecture

  NonStop SOAP has a modular architecture that can be used to modify the default message process of the NonStop SOAP 4 server using pluggable modules. For more information about modules, see "Customizing NonStop SOAP 4 Message Processing" (page 124).

  Message Receiver User Functions (MRUF) can be used to modify the message buffer and Pathway or NonStop process service invocation attributes. For more information about MRUF, see "Customizing NonStop SOAP 4 Message Processing" (page 124).

- ### Hot deployment of services

  New services can be deployed or existing services can be removed from NonStop SOAP without stopping the server.

- ### Dynamic invocation

  A single deployment of NonStop SOAP can be used to expose applications as Web services. Every application is deployed as a Web service under NonStop SOAP.

- ### Advanced caching

  NonStop SOAP provides built-in caching of services it hosts. The caching functionality helps you to access the frequently accessed services faster by caching the request and response structures.

- ### Encoding support

  NonStop SOAP provides native support for more than 1000 character encodings. For more information on supported encodings, see "NonStop SOAP 4 Features" (page 217).

Architecture of NonStop SOAP

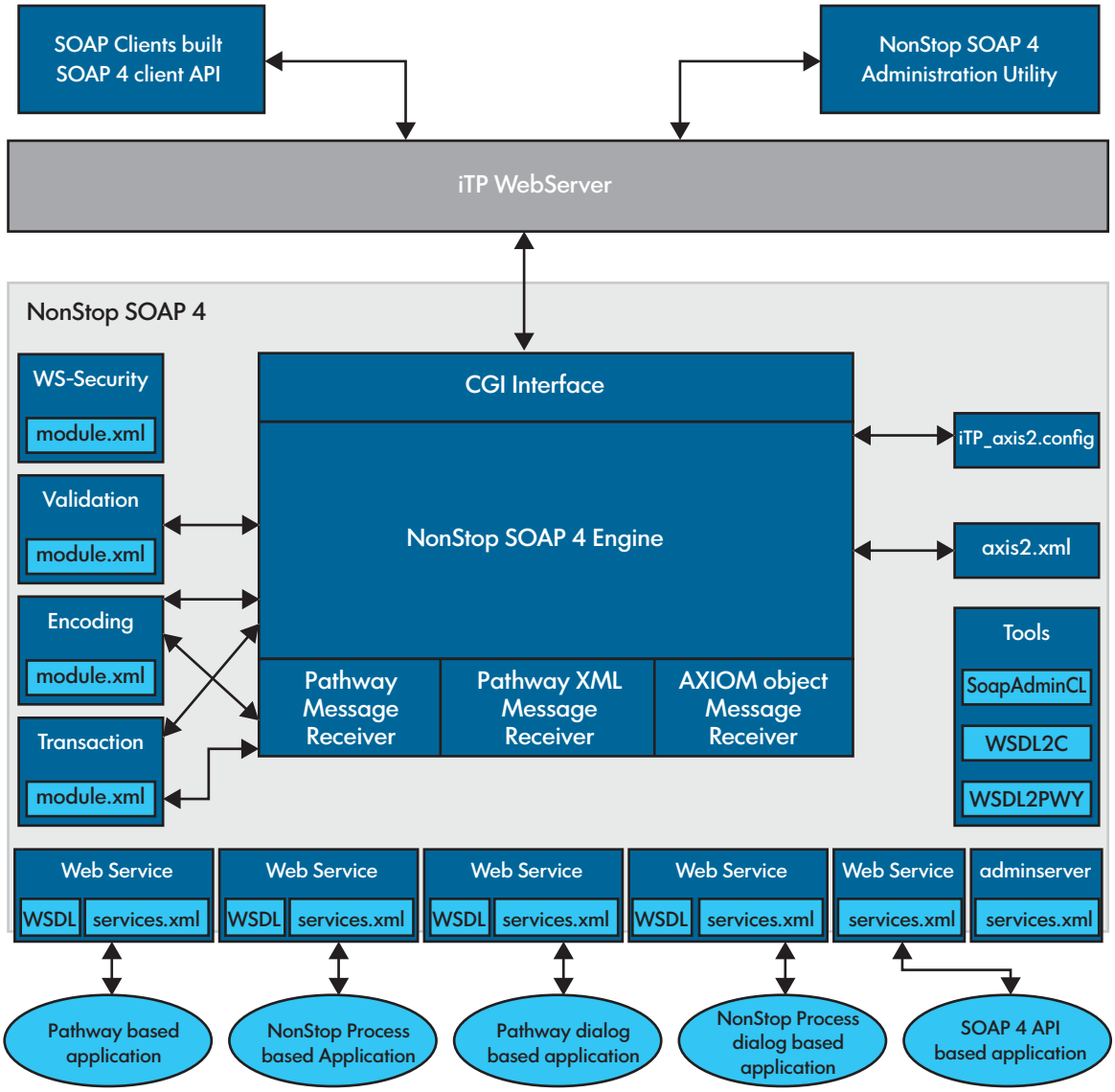
NonStop SOAP server runs under the iTP WebServer and uses it to communicate with SOAP clients using HTTP.

NonStop SOAP has a modular architecture with individual modules to:

- interact with the iTP WebServer
- process SOAP 1.1 and 1.2 messages
- communicate with Pathway applications and NonStop processes
- manage transactions and WS-Security headers

Figure 1 illustrates the NonStop SOAP architecture.

Figure 1 NonStop SOAP Architecture



NonStop SOAP 4 has the following components:

- "NonStop SOAP Server" (page 30)
- "NonStop SOAP Tools" (page 31)
- "NonStop SOAP Administration Utility" (page 32)
- "SOAP Clients" (page 32)

NonStop SOAP Server

NonStop SOAP server has the following components:

- "CGI Interface" (page 31)
- "NonStop SOAP 4 Engine" (page 31)
- "TS/MP Message Receiver" (page 31)
- "XML Message Receiver" (page 31)
- "WS-Security Module" (page 31)
- "Transaction Module" (page 31)

CGI Interface

NonStop SOAP runs as a server class under iTP WebServer. The CGI interface component of NonStop SOAP is an interface between iTP WebServer and the NonStop SOAP engine. The CGI interface component processes the Hypertext Transfer Protocol (HTTP) headers in SOAP requests and adds HTTP headers to the SOAP response.

NonStop SOAP 4 Engine

The NonStop SOAP engine processes the SOAP 1.1 or SOAP 1.2 requests and creates the appropriate SOAP response. The processing of the NonStop SOAP engine can be customized using pluggable modules attached to it.

TS/MP Message Receiver

The TS/MP message receiver is an interface between the NonStop SOAP engine and the TS/MP Web service. The TS/MP message receiver converts the request XML to a buffer that is understood by the underlying TS/MP application or NonStop process. In response processing, the TS/MP message receiver converts the response buffer from a TS/MP application or NonStop process to response XML.

XML Message Receiver

The XML message receiver is an interface between the dynamic-link library (DLL) based Web services and the NonStop SOAP 4 engine. The XML message receiver creates a data structure in the C programming language from the XML input and sends it to the DLL-based Web service.

WS-Security Module

The WS-Security module provides support for processing WS-Security headers in SOAP messages.

Transaction Module

The transaction module implements transaction management in SOAP communication between the client and the Web service.

NonStop SOAP Tools

NonStop SOAP 4 provides the following tools to ease the development of Web services:

- "SoapAdminCL Tool" (page 31)
- "WSDL2C Tool" (page 31)
- "WSDL2PWY Tool" (page 31)

SoapAdminCL Tool

The SoapAdminCL tool is used in the service-first development approach. The SoapAdminCL tool exposes the underlying TS/MP application or NonStop process as a Web service and generates the contract for the same.

WSDL2C Tool

The WSDL2C tool is used in the contract-first development approach to create DLL-based Web services in the C programming language from the WSDL contract.

WSDL2PWY Tool

The WSDL2PWY tool is used in the contract-first development approach to create a C-language TS/MP application-based Web service from the WSDL contract.

NonStop SOAP Administration Utility

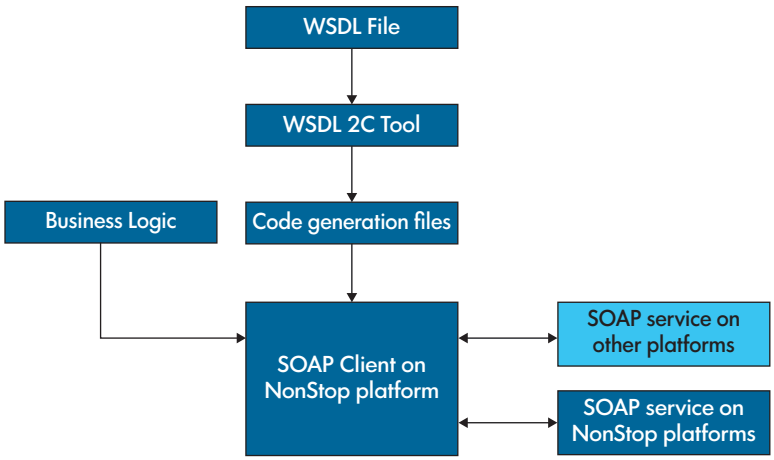
The NonStop SOAP Administration Utility (T0904) is a Java-based GUI tool that provides functionality to create and deploy the service definition language (SDL) file from a remote system such as Microsoft Windows. For more information about the NonStop SOAP 4 Administration Utility, see "NonStop SOAP Tools" (page 194).

SOAP Clients

The `WSDL2C` tool can also be used to generate SOAP clients in the C programming language. The SOAP clients developed using the `WSDL2C` tool can communicate with any SOAP service on any platform. For more information on building SOAP clients using the `WSDL2C` tool, see "NonStop SOAP 4 Client APIs" (page 105).

Figure 2 shows generation of SOAP clients and their capabilities.

Figure 2 SOAP Clients and their Capabilities



The `WSDL2C` tool generates most of the code to create the SOAP client from the WSDL file. You must add business logic to the generated code and build the SOAP client.

The SOAP clients generated using the `WSDL2C` tool can communicate with SOAP services running on NonStop platforms and other platforms.

Request Processing in NonStop SOAP

NonStop SOAP 4 communicates with the client using the SOAP protocol.

A SOAP message is a well formed XML message that includes:

- An XML declaration (optional)
- A SOAP envelope (the root element) made up of the following:
  - SOAP Header (optional)
  - SOAP Body

The SOAP envelope must use the SOAP 1.1 or SOAP 1.2 envelope namespace. A SOAP header is used to transmit data that is not part of the SOAP protocol, but includes transaction-related information or security-related information. The SOAP body element encapsulates the operation name and data in the format specified by the WSDL file for the service.

Figure 3 shows a typical SOAP message sent through HTTP.

NOTE: In Figure 3 and Figure 4, all shades of green represent the XML part of the message and the grey shade represents non-XML part of the message.

Figure 3 The SOAP Message Format

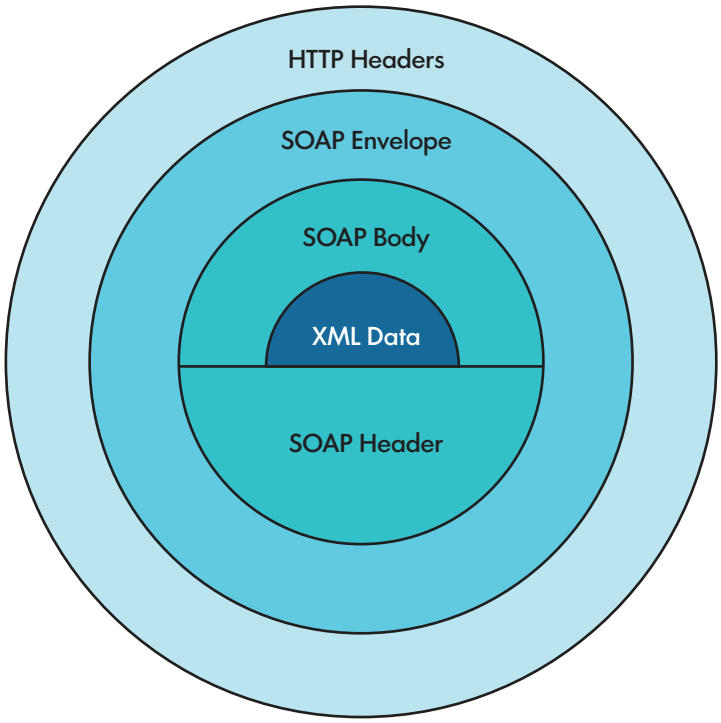


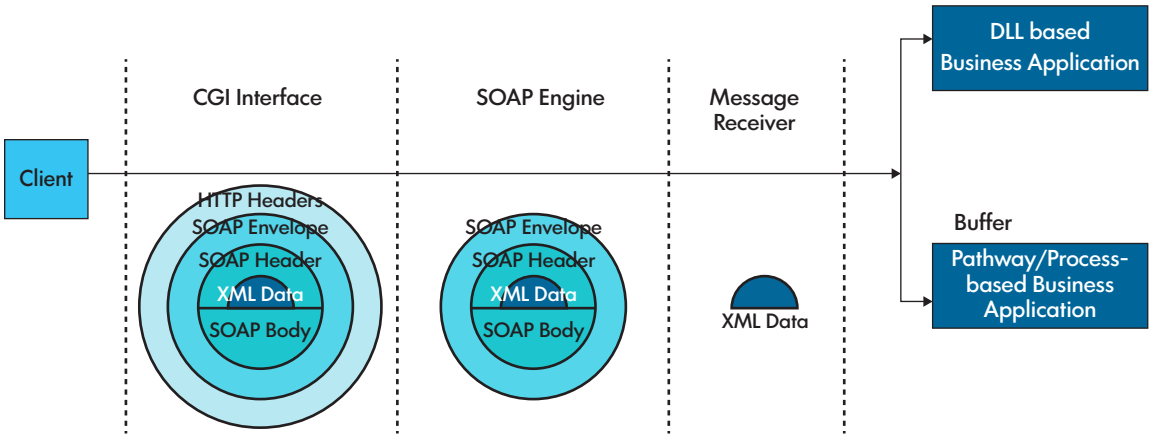
Figure 4 shows a typical SOAP message (in XML format) sent through HTTP.

Figure 4 The SOAP Message Format (in XML format)

```
<?xml version="1.0" ?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:urn1="urn:cpq_tns_reflector">
<soapenv:Header>
   <urn:Session BeginNewTransaction="true" />
 </soapenv:Header>
 <soapenv:Body>
   <urn1:reflector>
     <urn1:input>hello world</urn1:input>
   </urn1:reflector>
 </soapenv:Body>
</soapenv:Envelope>
```

Figure 5 shows the request processing flow for TS/MP based or NonStop process-based Web services.

Figure 5 NonStop SOAP 4 Request Processing Flow for TS/MP or NonStop process-based Web Services



A Web service client sends a request to the NonStop SOAP server using the SOAP protocol through the iTP WebServer. The request message is processed as follows:

1. When this request is received by the NonStop SOAP server, the CGI interface of NonStop SOAP processes the HTTP headers in the request message and forwards the SOAP envelope to the NonStop SOAP engine.
2. The NonStop SOAP engine processes the SOAP envelope, SOAP body, and the SOAP header. The XML request inside the SOAP body is then sent to the message receiver.
3. The message receiver converts the XML request to a format understood by the underlying business application and sends the request data to the business application. NonStop SOAP converts the response received from the business application to a SOAP message and sends this response to the Web service client through the iTP WebServer.

   For example, in a Pathway or process service invocation, the XML request is converted to a character buffer by the Pathway message receiver and is sent to a server class or process respectively. The server class or process sends the response buffer which is then converted into response XML by the Pathway message receiver.

Compatibility

NonStop SOAP 4 is backward compatible with NonStop SOAP 3.

Services developed in NonStop SOAP 4 are compatible with:

- SOAP clients that support SOAP 1.1 and SOAP 1.2 protocol.
- SOAP clients that communicate with NonStop SOAP 3 services.

Clients developed in NonStop SOAP 4 are compatible with:

- SOAP services that support SOAP 1.1 and SOAP 1.2 protocol
- SOAP services developed in NonStop SOAP 3.

Migration

NonStop SOAP 4 provides tools to migrate NonStop SOAP 3 services to NonStop SOAP 4 services. The information in the SDR used by the NonStop SOAP 3 services, can be extracted into NonStop SOAP 4 compatible SDL using the `SoapAdminCL` tool or the NonStop SOAP 4 Administration Utility (T0904). The SDL generated from the SDR is then used to migrate NonStop SOAP 3 service to a NonStop SOAP 4 service.

The `SoapAdminCL` tool or the NonStop SOAP 4 Administration Utility (T0904) can also be used to generate WSDL files which are compatible with the NonStop SOAP 3 service that is being

migrated. This allows the WSDL-based clients to seamlessly communicate with the migrated NonStop SOAP 4 service.

For information on how to migrate NonStop SOAP 3 services to NonStop SOAP 4 services, see "Migrating NonStop SOAP 3 Services to NonStop SOAP 4 or Higher Versions" (page 52).

2 Installing NonStop SOAP

This chapter describes the procedure to install NonStop SOAP on the NonStop system. It also describes the steps to deploy a sample Web service and the `adminserver` service.

The following tasks are described in this chapter:

- "Setting Up NonStop SOAP on a NonStop System" (page 36)
- "Deploying a Sample Web Service" (page 44)
- "Deploying the `adminserver` Service" (page 50)

Prerequisites

Before you start, ensure that the following software is installed on the NonStop system:

- Open System Services (OSS) environment installed on a NonStop system H06.21 or later.
- NonStop iTP WebServer 6.0 or later
- NonStop TS/MP 2.0 or later
- Data Definition Language (DDL) compiler 6.0 or later
- NonStop XML parser

NOTE: Ensure that you have read/write access to the OSS directory.

Setting Up NonStop SOAP on a NonStop System

Setting up NonStop SOAP on a NonStop system involves the following steps:

1. "Installing NonStop SOAP" (page 36)
2. "Setting up the Deployment Environment" (page 38)
3. "Tuning NonStop SOAP " (page 42)
4. "Running NonStop SOAP " (page 42)
5. "Setting up Multiple NonStop SOAP Deployment Instances in a Single iTP WebServer" (page 43)

Installing NonStop SOAP

Install NonStop SOAP on the target system using one of the following methods:

1. Default location installation
   - Using the DSM/SCM Planner Interface
2. Non-standard location installation
   - Using PINSTALL utility in Guardian environment
   - Using PAX command in OSS environment

Default location installation

This is the default method of installation to the target location. You can use the Distributed Systems Management/Software Configuration Manager (DSM/SCM) Planner Interface for this method of installation. Receive the file from disk or tape to a guardian location on the target system and follow the steps as described below:

1. RECEIVE the product SPR files from the disk (distribution subvolume (DSV) locations) or tape.
2. COPY the SPR to a new revision of the software configuration that is being updated.

3. BUILD and APPLY the configuration revision.

   In the DSM/SCM planner interface, if you select **Maintain > Target Maintenance > Modify > MANAGE OSS FILES** option, the files are automatically extracted by DSM/SCM into the default location `/usr/tandem/nssoap/t0865h01_<XXX>`, where `<XXX>` is the SPR ID.

   If you do not select the **MANAGE OSS FILES** option, the `PAX` file is only copied to the `$tsvvol.ZOSSUTL` subvolume, but not extracted to the `/usr/tandem/nssoap/` location. You must then manually extract the `PAX` file using the PINSTALL utility or the `PAX` command as described in the subsequent sections.

4. Run ZPHIRNM to perform the RENAME.

NOTE: The DSM/SCM interface does not support multiple installations. If you install NonStop SOAP 4 using DSM/SCM, the existing installation of NonStop SOAP 4 is replaced by the new installation. However, if you want to have multiple installations on the target system, follow the Non-standard location installation method, using the PINSTALL utility or the `PAX` command.

Non-standard location installation

In this method of installation, you can use either PINSTALL utility or the `PAX` command to extract the `PAX` file to any location.

Using PINSTALL Utility in Guardian Environment

1. Go to the Guardian subvolume `$ISV.ZOSSUTL` where the `PAX` file is located:

   ```
   TACL> VOLUME $ISV.ZOSSUTL
   ```

   where, `ISV` is the installation volume where SUT or OSS Utils are installed.

2. Extract the `T0865PAX` file using the `PINSTALL` command:

   ```
   TACL> PINSTALL -s:<version-specific directory>:<your-install-dir>: -rvf T0865PAX
   ```

   where,

   `<version-specific directory>` is of the form `/usr/tandem/nssoap/t0865h01_<XXX>` (`<XXX>` being the SPR ID). For example, `/usr/tandem/nssoap/t0865h01_AAA`, and

   `<your-install-dir>` is an OSS directory of your choice.

   NOTE:

   - $ISV.ZOSSUTL is the target location specified during export of `PAX` file from DSM/SCM or when installing SUT via DSM/SCM.
   - If the –s option is not given, PINSTALL by default extracts the `PAX` to the *`<version-specific directory>`*.
   - For more information about using `PINSTALL`, see the *Open System Services Management and Operations Guide*.

Using PAX Command in OSS Environment

Go to the folder where `T0865PAX` is located and use the following command:

```
pax -rvf T0865PAX
```

NOTE: The `PAX` command extracts the product files from the `T0865PAX` file and places them in a version specific OSS directory, like `/usr/tandem/nssoap/t0865h01_AAA` where, `AAA` is the SPR ID.

For more details on the files and folder structure, see "Install Files and Folders" (page 308).

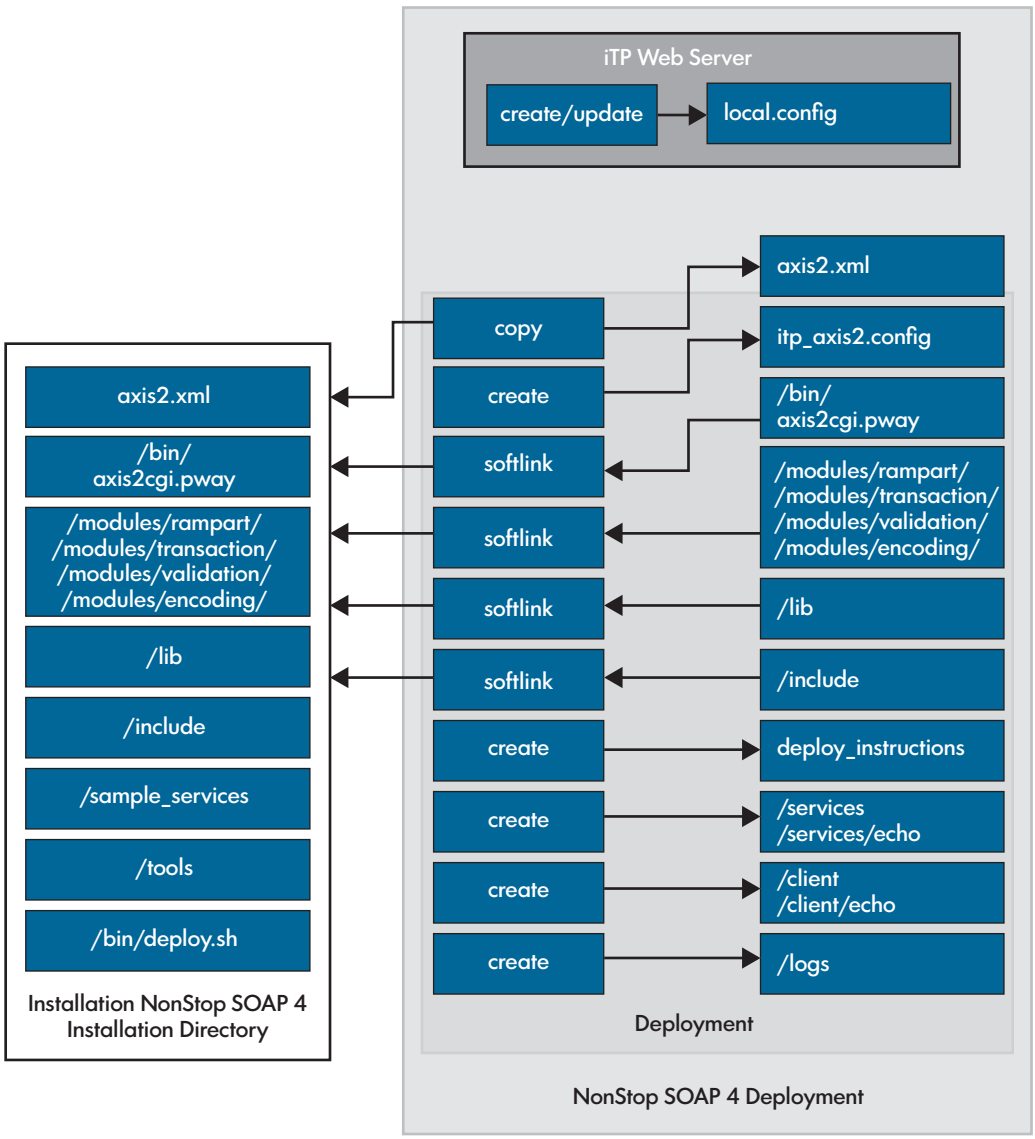
Setting up the Deployment Environment

After NonStop SOAP is installed, you must create at least one NonStop SOAP deployment environment by running the NonStop SOAP deployment script. The deployment script will create a new NonStop SOAP deployment under the iTP WebServer you choose, and install the `echo` sample service and `echo` client. You can deploy multiple instances of NonStop SOAP on the same NonStop system. Multiple instances of NonStop SOAP can be deployed under different iTP WebServers or under the same iTP WebServer.

For more information see "Setting up Multiple NonStop SOAP Deployment Instances in a Single iTP WebServer" (page 43).

Figure 6 shows the NonStop SOAP 4 deployment directory that is created after running the deployment script and its relationship with the NonStop SOAP Installation directory.

Figure 6 NonStop SOAP 4 Installation and Deployment Environment



NOTE: If you reinstall NonStop SOAP, you must redeploy it to create the soft links.

To deploy NonStop SOAP on the iTP WebServer, complete the following steps:

- "Running the Deployment Script" (page 39)
- "Modifying the `local.config` file" (page 41)

Running the Deployment Script

To set up the deployment environment, complete the following steps:

1. Ensure that the iTP WebServer is installed on your NonStop server. For more details, see *iTP WebServer Installation Manual*.
2. Go to `<NonStop SOAP 4 Installation Directory>/bin` and run the `deploy.sh` script.

   For example:

   ```
   OSS> cd /usr/tandem/nssoap/t0865h01/bin
   OSS> ./deploy.sh
   ```

   where,

   `/usr/tandem/nssoap/t0865h01`
   is the NonStop SOAP installation directory.
3. Select a task from the following `deploy.sh` script main menu:
   1. Setup NonStop SOAP deployment environment.
   2. Setup the empdb sample Web services environment.
   3. Setup the adminserver Web services environment.
   4. Quit.
4. Select `"1 Setup NonStop SOAP deployment environment"` in the `deploy.sh` script menu.

   The following messages appear.

   ```
   --- The NonStop SOAP 4 installation location is /usr/tandem/nssoap/t0865h01
   === Enter the directory where NonStop SOAP 4 will be deployed
   :
   ```
5. Enter the path where you want to create the deployment directory.

   For example:

   `/home/usr/my_nssoap`

   NOTE:

   - The deployment directory will be referred as `<NonStop SOAP Deployment Directory>` throughout this chapter.
   - While specifying the deployment directory, **do not** give a location that has the `$` character in it.

   The following message appears.

   ```
   === Enter a string value which will be used to define the URL pattern for accessing the web services
   deployed in this environment (default axis2c)
   :
   ```
6. Enter a string value that will be used to define the `url_pattern` for accessing Web services or

   Press **Enter** to accept the default value, that is, `axis2c`.

   The Web address pattern must be a unique string. It must be unique among all other instances of NonStop SOAP deployment on that iTP WebServer. The following messages appears.

   ```
   --- The URL pattern to access your wsdl files will be:
   http://<your_ip_address>:< your_port>/<url_pattern>/services/< your service>?wsdl
   --- The URL pattern to access your generated HTML files will be:
   http://<your_ip_address>:< your_port>/<url_pattern>/client
   === Enter a string value which will be used to define the Serverclass name for this Soap Server deployment
   under iTP WebServer (default axis2cgi)
   :
   ```

   Here the string value will be a part of the Web address to access Web services deployed in this deployment environment. The Web address pattern is shown in the previous message.

where,

< ip address> : <port>

is the IP address and port of iTP WebServer with which NonStop SOAP 4 is integrated.

`url_pattern`

is the string value entered in this step.

`services`

is the name of the directory in *< NonStop SOAP 4 Deployment Directory >* where NonStop SOAP services are present.

NOTE: Do not rename this directory. This directory must be named as services and placed in *< NonStop SOAP 4 Deployment Directory >*. All services in NonStop SOAP have their individual sub-directories in the services directory.

`< service_name>`

is a sub-directory in the `services` directory that includes the runtime files needed to deploy a service in NonStop SOAP. For example, if you accept the default value (`axis2c`), the Web address to access a service called myservice will be:

`http://127.0.0.1:8088/axis2c/services/myservice`

The following message appears.

```
=== Enter a string value which will be used to define the Serverclass name for
this Soap Server deployment under iTP WebServer (default axis2cgi)
:
```

7. In the following step enter a string value that will be used to define the Soap ServerClass name for current Soap 4 deployment

   or,

   Press `Enter` to accept the default value, that is, `axis2cgi`. The Soap ServerClass name must be a unique string. It must not be similar to any other instance of NonStop SOAP deployment on that iTP WebServer. This is used to create the softlink for *< NonStop SOAP Installation Directory>*`/bin/axis2cgi.pway` file and create entry in *< NonStop SOAP Deployment Directory>*`/itp_axis2.config` file.

   The following message appears:

   ```
    === Enter iTP Webserver's location
   === Press Enter if you do not want to deploy under running web server
   ```

8. In the following step enter the location where the iTP WebServer is installed. If you enter your WebServer's location, the `local.config` file for your WebServer will be modified to start this SOAP 4 deployment. Press ENTER if you do not know the location of the iTP WebServer you plan to use this SOAP 4 deployment.

   For example: `/home/usr/my_webserver`

   NOTE:

   - Enter the absolute path where the iTP WebServer is installed. This location must include `httpd.config` file.
   - While specifying the WebServer's directory, you must not include `$` character in location.
   - If `httpd.config` file does not include the reference to `local.config` file, a reference will be created. If `local.config` file does not exist, then it will be created. The `local.config` will be modified to source in the SOAP server's config file during iTP WebServer startup.

9. The following message appears:

=== Press **ENTER** to deploy NonStop SOAP 4 at the deployment location and install the `echo` sample service.

If web server's location is provided in earlier steps, The following message appears:

```
=== Do you want to restart the iTP Webserver now? (y/n)
```

Provide your option Y/N. If 'Y' is provided, this will restart the iTP Web server under which the current SOAP server is deployed.

The following message appears:

```
=== Press ENTER to return to the main menu
```

10. After NonStop SOAP is deployed and the echo sample service is installed, select a task from the following options:
    - Select option 2 for "Deploying the `empdb` Service" (page 44)
    - Select option 3 for "Deploying the `adminserver` Service" (page 50)
    - Select option 4 to **Quit**

NOTE: For the complete list of folders and files, see "Install Files and Folders" (page 308).

Modifying the `local.config` file

If you have not specified the location of your iTP WebServer deployment during the execution of the SOAP 4 deployment script, then you must manually edit the `local.config` file in your iTP WebServer's conf directory according to the following instructions:

Copy the following lines of code from the `deploy_instructions` file (created in *<NonStop SOAP Deployment Directory>*) and append it to the `local.config` file (in the *<iTP WebServer Deployment Directory>*`/conf` directory):

```
#### Begin (lines to be added to iTP WebServer's local.config file) ####

      if { [file exists <NonStop SOAP 4 deployment directory>/itp_axis2.config] } {
      source <NonStop SOAP  deployment directory>/itp_axis2.config
      }

#### End (lines to be added to iTP WebServer's local.config file) ####
```

For example:

```
#### Begin (lines to be added to iTP WebServer's local.config file) ####

      if { [file exists /home/usr/my_nssoap/itp_axis2.config] } {
      source /home/usr/my_nssoap/itp_axis2.config
      }

#### End (lines to be added to iTP WebServer's local.config file) ####
```

where,

`/home/usr/my_nssoap`
  is the *<NonStop SOAP 4 Deployment Directory>*.

NOTE:

- *<iTP WebServer Deployment Directory>* is the OSS directory where iTP WebServer is deployed using the iTP WebServer setup script.
- If the iTP WebServer is configured to use a TCP/IP process other than `/G/ZTC0`, check the following directive is there in the `itp_axis2.config` file. If it is not there then add it in the file:

```
Server  $Axis2c {
        ...
        MapDefine =TCPIP^PROCESS^NAME  $transport
        ...
}
```

  For additional information on how to verify the iTP WebServer configuration, see the *iTP Secure WebServer System Administrator's Guide*. For information on the `itp_axis2.config` file, see "NonStop SOAP 4 Configuration Files".

NonStop SOAP is now deployed on the iTP WebServer and is set up in the *<NonStop SOAP 4 Deployment Directory>*. The same way NonStop SOAP can be deployed on secure iTP WebServer by updating the `local.config` in the *< Secure iTP WebServer Deployment Directory>*`/conf` directory. You can set up multiple deployment instances of NonStop SOAP on the same NonStop system. For more information on setting up multiple deployment instances, see "Setting up Multiple NonStop SOAP Deployment Instances in a Single iTP WebServer" (page 43).

Tuning NonStop SOAP

The `AXIS2CGI` server class that implements the SOAP 4 server is a single-threaded process and only services one SOAP request at a time. If you want that multiple SOAP requests be serviced concurrently, you must configure the iTP WebServer PATHMON in which SOAP 4.1 is running to have additional static or dynamic copies of the `AXIS2CGI` server class.

For information on how to tune a server class, see the *TS/MP System Management Manual*.

Running NonStop SOAP

After deploying NonStop SOAP on the iTP WebServer, you can run it on a NonStop system.

To run NonStop SOAP on a NonStop system, complete the following steps:

NOTE: You can skip step1, if iTP Webserver is restarted while deploying NonStop SOAP 4.

1. Go to the *<iTP WebServer Deployment Directory>*`/conf` directory and run the `restart` script:

   ```
   OSS> cd <iTP_WebServer_Deployment Directory>/conf
   OSS> ./restart
   ```

   This command restarts the iTP WebServer and NonStop SOAP 4.
2. Verify that iTP WebServer and NonStop SOAP 4 are running:

   ```
   OSS> gtacl -c 'pathcom <PATHMON_name>; status server *'
   ```

   where,

   <PATHMON_name>

   is the name of the PATHMON on which iTP WebServer is running.

   This command returns the following statistics to verify if iTP WebServer and NonStop SOAP 4 are running:

   ```
   SERVER            #RUNNING   ERROR  INFO
   AXIS2CGI             1
   ```

```
GENERIC-CGI          1
HTTPD                1
```

NOTE:

- `AXIS2CGI` represents a server class name.
- A value of `1` for `GENERIC-CGI` or `HTTPD` indicates that iTP WebServer is running.
- A value of 0 for `GENERIC-CGI` or `HTTPD` indicates that iTP WebServer is not running and an error number is displayed in the respective ERROR column. For more information about the error and its resolution, see the *NonStop TS/MP Pathsend and Server Programming Manual*.

After NonStop SOAP 4 is deployed and running, you can access the `echo` sample service. The `echo` sample returns the same value in the response which you send in the request.

Accessing the `echo` service

To access the `echo` service, run the command-line client `echo.exe`. The `echo.exe` file is located in the *<NonStop SOAP 4 Deployment Directory>*`/client/echo` directory.

```
OSS> cd <NonStop SOAP 4 Deployment Directory>/client/echo
OSS> ./echo.exe http://<ip address>:<port>/<url_pattern>/services/echo
```

The `echo.exe` file sends requests to the NonStop SOAP 4 server and displays the following output:

```
No CLIENT HOME specified. Using default AXIS2C_HOME=<NonStop SOAP 4 Deployment Directory>

Using endpoint : http://www.nonstopsoap.com/axis2c/services/echo

Sending OM : <ns1:echoString xmlns:ns1="http://ws.apache.org/axis2/services/echo
"><text>Hello</text></ns1:echoString>

Received OM : <ns1:echoString xmlns:ns1="http://ws.apache.org/axis2/c/samples"><
text>Hello</text></ns1:echoString>

echo client invoke SUCCESSFUL!
```

The echo sample generates `echo.log` and `echo.log.trc` log files in the *<NonStop SOAP 4 Deployment Directory>*`/logs` directory.

The `echo.log` file contains log messages from NonStop SOAP libraries. The log levels are set in `itp_axis2.config` file. For information on setting the log levels, see "Defining the Log Levels of the NonStop SOAP 4 Server" (page 178).

The `echo.file.trc` file contains the message processing details. The log contains both the entry and exit points for functions in the order in which they are run. The messages in `echo.file.trc` file are used for debugging the echo sample.

Setting up Multiple NonStop SOAP Deployment Instances in a Single iTP WebServer

To set up multiple deployment instances of NonStop SOAP to run in a single instance of iTP WebServer, complete the following steps:

1. Create more than one NonStop SOAP deployment directories using `deploy.sh` script."Setting up the Deployment Environment" (page 38):
   - *<NonStop SOAP Deployment Directory 1>*
   - *<NonStop SOAP Deployment Directory 2>*

     For multiple directories, you can create directories, such as *NonStop SOAP Deployment Directory 3, 4.....*

NOTE: Ensure the following while creating the directories:

- When running the `deploy.sh` script, the Soap ServerClass name must be a unique string. It must not be similar to any other instance of NonStop SOAP deployment on that iTP WebServer.
- If you want to manually edit the `local.config` file, update the file for both the NonStop SOAP deployment directories, by using the instructions mentioned in "Setting up the Deployment Environment" (page 38).
- When running the `deploy.sh` script, the Web address pattern must be a unique string. It must not be similar to any other instance of NonStop SOAP deployment on that iTP WebServer.

2. The Soap ServerClass name is used to create the softlink for `< Installation_Dir>/bin/axis2cgi.pway` file as follows.

   ```
   <NonStop SOAP 4 Deployment Directory 2>/bin/< $Soap ServerClass>.pway
   ```

3. The `deploy.sh` script also uses Soap ServerClass name to make the entry in `< NonStop SOAP 4 Deployment Directory>/itp_axis2.config` file as follows.

   ```
   set Axis2c $AXIS2_DEPLOYMENT_ROOT/bin/< $Soap ServerClass>.pway
   ```

NOTE: You can skip step 4 if iTP Webserver is restarted while deploying the NonStop SOAP 4.

4. Go to the `< iTP WebServer Deployment Directory>/conf` directory and run the restart script using the following command:

   ```
   OSS> cd < iTP_WebServer_Deployment Directory>/conf
   OSS> ./restart
   ```

   This command restarts the iTP WebServer and the NonStop SOAP system with more than one deployment instance of NonStop SOAP.

For accessing the Web services deployed on multiple NonStop SOAP Deployment instances in a Single iTP WebServer by using different `url_pattern`.

NOTE: Ensure that the `url_pattern` is different for all deployment instances.

Deploying a Sample Web Service

The NonStop SOAP distribution includes the following sample services: `empdb`, `echo`, `reflector`, and `math`. This section describes the steps to deploy the `empdb` service.

NOTE: The `deploy.sh` script enables you to set up only the `empdb`, and the `echo` services.

Deploying the `empdb` Service

The `empdb` service enables you to:

- Add an employee and the employee details, such as Employee number, Firstname, Middlename, Lastname, Registration number, and Branch number.
- Search for an employee using Employee number.
- Delete an employee using Employee number.

Deploying the `empdb` sample service involves the following steps:

1. "Setting up the `empdb` environment" (page 45)
2. "Accessing the `empdb` service" (page 48)

Setting up the empdb environment

To set up the empdb service environment, complete the following steps:

1. Go to <*NonStop SOAP 4 Installation Directory*>/bin and run the deploy.sh script.

   For example:

   ```
   OSS> cd /usr/tandem/nssoap/t0865h01/bin
   OSS>./deploy.sh
   ```

   where,

   /usr/tandem/nssoap/t0865h01 is the NonStop SOAP 4 installation directory.

2. The following messages appear showing the installation location being used:

   ```
   Setting up NonStop SOAP 4 empdb sample web services
   --- The NonStop SOAP 4 installation location is /usr/tandem/nssoap/t0865h01
   === Enter a NonStop SOAP 4 deployment directory path
   :
   ```

   NOTE:

   1. If you did not exit the deploy.sh script after "Setting up the Deployment Environment" (page 38), you will be prompted with default deployment path. Press **Enter** to accept default path or enter the different deployment path.
   2. If you try to deploy empdb service in the location where the empdb service is already deployed then you will be prompted to redeploy it or not as in following message:

      ```
      The empdb service already deployed.
      Do you want to redeploy it? y/n
      Note: This will overwrite the existing deployment and will shutdown and restart the existing Pathmon:
      ```

3. Enter the path of the <*NonStop SOAP 4 Deployment Directory*> to get the following message.

   ```
    Enter a Pathmon name for the sample Employee Serverclass (e.g. $pmon)
   :
   ```

   NOTE: If you specify a PATHMON name which already exists, you will be prompted with options as in the following message:

   ```
   The Pathmon name you have entered already exists.  Please select a task from the following options:
   1. Give different PATHMON name
   2. Proceed with the given PATHMON name
   Note: This will shutdown and restart the existing PATHMON
   3. Quit
   Enter Your Choice:
   ```

   where,

   Option 1 allows you to specify a different PATHMON name.

   Option 2 proceeds with the same PATHMON name, you will be prompted with the following message:

   ```
   === Are you sure?(y/n):
       All server configurations under this pathmon will be lost
       and the servers will be stopped.
   ```

   Option 3 exits the script.

4. Enter the PATHMON name where the empdb server class will run the empdb server instance. The following message appears.

```
Enter the Guardian location where the DDL dictionary for the empdb sample will be created (e.g. $data.dict)
Note: You should have write access to this location:
```

NOTE: If you specify a Guardian location which already exists, you will be prompted with options as in the following message.

```
The Guardian location you have entered already contains dictionary files.


Please select a task from the following options:
1. Give different guardian location
2. Append or Update to existing dictionary files and purge empdb sample specific files
3. Purge all existing dictionary and empdb sample specific files
4. Quit
Enter your Choice:
```

where,

Option 1 allows you to specify a different Guardian location.

Option 2 , will cause `empdb` sample specific files to be purged and existing dictionaries to be appended/updated with the empdb service dictionaries.

Option 3 purges the existing dictionaries and `empdb` service related files and proceed with current dictionary location.

Option 4 exits the script.

5. Enter the `$<volume>.<subvolume>` location to which you have write access.

   For example:

   ```
   $data.empdict
   ```

   The `deploy.sh` script sets up the empdb environment and starts the empdb service. If the setup is successful, the `deploy.sh` script displays the following message:

   ```
   "empdb sample web service setup completed SUCCESSFULLY"
   ```

   If the setup is not successful, the `deploy.sh` script displays the following message:

   ```
   "empdb sample web service setup failed".
   ```

   For more information on the errors, see the `< NonStop SOAP 4 Deployment Directory>/services/empdb/src/deploy_out` file. This file includes the execution summary and detailed error messages of the `deploy.sh` script.

6. The following files and folders are present in the *<NonStop SOAP 4 Deployment Directory>*:
   - /services
     - /empdb
       - SoapPW_empdb.wsdl - the WSDL file for the empdb sample application
       - /bin
         - empdb - employee database service executable
       - services.xml - the service configuration file for the empdb sample application
       - /src
         - Makefile - Makefile for the empdb service
         - README - readme file describing steps to use the empdb application
         - deploy_out - log file generated during deployment.
         - emp.c - source file for the empdb service
         - emp.h - header file for the empdb service
         - empddl - DDL file for the empdb service
         - empsdl.xml - SDL file for the empdb sample application
   - /client

- /empdb
  - SoapPW_EmpAdd.html - client for the add employee operation
  - SoapPW_EmpDel.html - client for the delete employee operation
  - SoapPW_EmpInfo.html - client for the Info employee operation
  - /pway-xml - sample request/response files
    - SoapPW_EmpAdd.xml - employee Add operation request file
    - SoapPW_EmpAddResponse0.xml - employee Add operation response file
    - SoapPW_EmpAddResponse1.xml - employee Add operation fault response file
    - SoapPW_EmpDel.xml - employee Delete operation request file
    - SoapPW_EmpDelResponse0.xml - employee Delete operation response file
    - SoapPW_EmpDelResponse1.xml - employee Delete operation fault response file
    - SoapPW_EmpInfo.xml - employee Information operation request file
    - SoapPW_EmpInfoResponse0.xml - employee Information operation response file
    - SoapPW_EmpInfoResponse1.xml - employee Information operation fault response file
  - /pway-xsd - XML schema files
    - SoapPW_EmpAdd.xsd - employee Add operation request schema file
    - SoapPW_EmpAddResponse0.xsd - employee Add operation response schema file
    - SoapPW_EmpAddResponse1.xsd - employee Add operation fault response schema file
    - SoapPW_EmpDel.xsd - employee Delete operation request schema file
    - SoapPW_EmpDelResponse0.xsd - employee Delete operation response schema file
    - SoapPW_EmpDelResponse1.xsd - employee Delete operation fault response schema file
    - SoapPW_EmpInfo.xsd - employee Information operation request schema file
    - SoapPW_EmpInfoResponse0.xsd - employee Information operation response schema file
    - SoapPW_EmpInfoResponse1.xsd - employee Information operation fault response schema file

Accessing the empdb service

To access the empdb service, complete the following steps:

1. Ensure that the iTP WebServer/ Pathmon is deployed on the NonStop server. Please see the *iTP WebServer Installation Manual*.
2. Verify that the empdb service is running, using the command:

```
OSS> gtacl -c 'pathcom <PATHMON_name>; status server *'
```

where,

<PATHMON_name> is the name of the PATHMON entered while setting up the empdb environment.

This command returns the following statistics:

```
SERVER            #RUNNING    ERROR   INFO
EMPDB              1
```

3. Access the empdb service using the following Web address pattern:

   http://*<ip address>*:*<port>*/*<url_pattern>*/client/empdb/*<html_client_name>*

   where,

   <ip address>:*<port>*

   is the IP address and port of iTP WebServer integrated with NonStop SOAP.

   url_pattern

   is the string entered in Step 6 in "Setting up the Deployment Environment" (page 38). The default value is axis2c.

   client

   is the name of the directory in *<NonStop SOAP 4 Deployment Directory>* where NonStop SOAP 4 HTML clients for the empdb service are located.

   empdb

   is a sub-directory in the client directory. The empdb sub-directory includes the HTML clients that you must access the operations offered by the empdb service.

   <html_client_name>

   is the name of the HTML client in the client/empdb directory.

   NOTE: To access the WSDL file for the empdb service, enter the following Web address on your browser:

   http://*<ip address>*:*<port>*/*<url_pattern>*/services/empdb?wsdl

   To create a client for the empdb service using the WSDL file, ensure that the <soap:address location> element in the WSDL file is updated to reflect the correct iTP WebServer IP address and port number.

   For example:

```
<service name="empdbService">
<port name="portempdb" binding="def:empdbBinding">
 <soap:address location="http://www.nonstopsoap.com/axis2c/services/empdb"/>
 </port>
 </service>
```

   where,

   www.nonstopsoap.com

   is the iTP WebServer IP address and port number in the format *<ip address>*:*<port>*.

   axis2c

   is the url_pattern. If you have entered a different string for your url_pattern, the same will be reflected in the WSDL file instead of axis2c.

4. You can access the HTML clients using the following Web address:

   http://*<ip address>*:*<port>*/*<url-pattern>*/client

The links to HTML clients for the following `empdb` service operations are displayed on the browser:

Filename	Description

SoapPW_EmpAdd.html	HTML client for the employee add operation.
	For Example:
	`request_code : 02 (request code for add employee operation)`
	`employee_info`
	`employee_number : 0001 (employee number is the primary key)`
	`empname`
	`first : John`
	`last : Doe`
	`middle : M`
	`regnum : 12`
	`branchnum : 12`
	The request code for the add employee operation is 02.

SoapPW_EmpInfo.html	HTML client for the employee info operation.
	For Example:
	`request_code : 01 (request code for Info Employee operation)`
	`employee_number : 0001`
	The request code to receive employee information operation is 01.

SoapPW_EmpDel.html	HTML client for the employee delete operation.
	`request_code : 03 (request code for delete employee operation)`
	`employee_number : 0001`
	The request code to delete an employee operation is 03.

Deploying the `adminserver` Service

Deploying the `adminserver` service enables you to use the NonStop SOAP Administration Utility. To deploy the `adminserver` service, you must set up the `adminserver` environment, which involves the following steps:

1. "Setting up the `adminserver` Environment" (page 50)
2. "Accessing the `adminserver` Service" (page 51)

Setting up the `adminserver` Environment

To set up the `adminserver` environment, complete the following steps:

1. Go to *`<NonStop SOAP 4 Installation Directory>`*`/bin` and run the `deploy.sh` script.

   For example:

   ```
   OSS> cd /usr/tandem/nssoap/t0865h01/bin
   OSS>./deploy.sh
   ```

2. Select `"3 Setup the adminserver web services environment".`

   The following message appears displaying the installation location being used:

   ```
   --- The NonStop SOAP 4.1 installation location is /usr/tandem/nssoap/t0865h01
   === Enter a NonStop SOAP 4 deployment directory path:
   ```

NOTE:

- If you did not exit the `deploy.sh` script after "Setting up the Deployment Environment" (page 38), you will be prompted with default deployment path. Press `Enter` to accept default path or enter the different deployment path.
- If you try to deploy `adminserver` service in the location where the `adminserver` service is already deployed then you will be prompted to redeploy it or not as in following message:

```
=== Enter a NonStop SOAP 4 deployment directory path
: /home/username/soap0
*** Warning: The Adminserver already deployed.
Do you want to redeploy it? (y/n)
Note: This will overwrite the existing deployment.
```

3. Enter the absolute path of the `<NonStop SOAP 4 Deployment Directory>` directory. The `deploy.sh` script sets up the adminserver environment.
4. The following files and directories are present in *`<NonStop SOAP 4 Deployment Directory>`*:
   - `/services`
     - `/adminserver`
       - `libadminserver.so` - the `adminserver` service built using NonStop SOAP 4 server APIs.
       - `services.xml` - NonStop SOAP 4 service configuration file for the `adminserver` service.

Accessing the `adminserver` Service

You can access the `adminserver` service using the NonStop SOAP Administration Utility running on your Microsoft Windows system. For information on NonStop SOAP 4 Administration Utility, see "The NonStop SOAP 4 Administration Utility" (page 215).

3 Migrating NonStop SOAP 3 Services to NonStop SOAP 4 or Higher Versions

NonStop SOAP enables migration of existing NonStop SOAP 3 services to NonStop SOAP 4 or 4.1.

This chapter includes the following topics:

- "Prerequisites" (page 52)
- "Migrating the NonStop SOAP 3 Services" (page 52)
- "Migrating NonStop SOAP 3 Transactions" (page 58)
- "Migrating NonStop SOAP 3 User-exits" (page 61)

Prerequisites

Before you start, ensure that you have the following information:

- *<NonStop SOAP 4 Installation Directory>*

  This is the directory where NonStop SOAP 4 is installed.

  For information on installing NonStop SOAP 4, see Chapter 2: "Installing NonStop SOAP" (page 36).

- *<NonStop SOAP 4 Deployment Directory>*

  This is the directory where NonStop SOAP is deployed.

  For information on deploying NonStop SOAP , see "Setting up the Deployment Environment" (page 38).

- Read and write permissions for the NonStop SOAP deployment directory.
- Read permission to the Guardian location of the DDL dictionary files of the service to be migrated.
- Read permission to the Guardian location of the service definition repository (SDR) file for the Pathway application that is deployed as a Web service in NonStop SOAP 3.
- The IP address and port number of iTP WebServer, in which NonStop SOAP 4 is deployed.

Migrating the NonStop SOAP 3 Services

Migrating the NonStop SOAP 3 services involves the following:

- "Generating SOAP 4 Service Artifacts and Client Files from NonStop SOAP 3" (page 52)
- "Accessing the Migrated Application" (page 56)

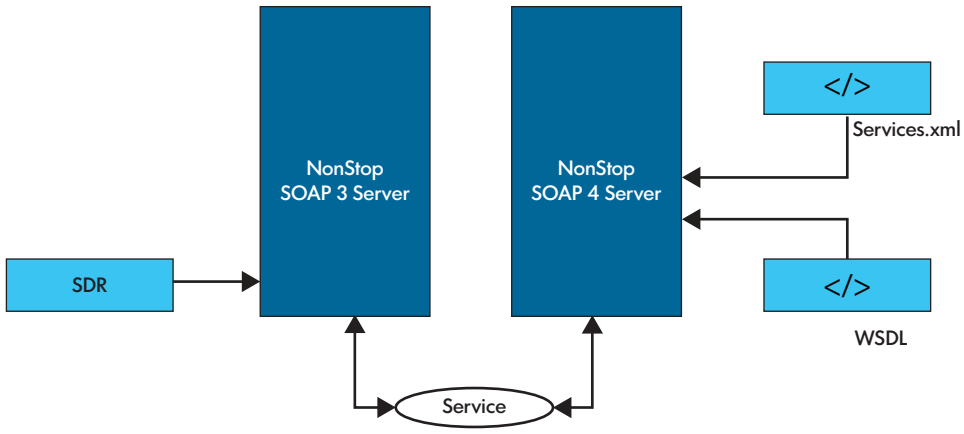
Generating SOAP 4 Service Artifacts and Client Files from NonStop SOAP 3

NonStop SOAP 3 uses the SDR file to map the request and response patterns for the services deployed in it. NonStop SOAP uses the WSDL file with integrated XML schema to map the request and response patterns for the services deployed in it .

NOTE: The SDR file is a key-sequenced Enscribe file stored at a Guardian location.

Figure 7 shows the processing logic in NonStop SOAP 3 and NonStop SOAP 4.1.

Figure 7 Processing Logic in NonStop SOAP 3 and NonStop SOAP 4.1



You can use the `SoapAdminCL` tool (available with the NonStop SOAP 4.1 distribution) to convert the NonStop SOAP 3 SDR file and the DDL dictionary files to the NonStop SOAP 4.1 `services.xml` file and WSDL files. For all the operations of the service, the `SoapAdminCL` tool generates the SOAP request and response XML files, XML schema files, and HTML client files.

NOTE: You must not stop a service to migrate it from NonStop SOAP 3 to NonStop SOAP 4.1. This means that your clients using the service do not experience any downtime.

Migrating and deploying an application from NonStop SOAP 3 to NonStop SOAP 4.1 requires the following tasks:

- "Generating the SDL File from the NonStop SOAP 3 SDR File" (page 53)
- "Modifying the SDL File" (page 54)
- "Generating the HTML Client, WSDL, XML Schema, and `services.xml` Files to Deploy the Service" (page 54)
- "Migration Considerations" (page 56)

Generating the SDL File from the NonStop SOAP 3 SDR File

If you do not have the SDL file for the NonStop SOAP application, you can generate it from the NonStop SOAP 3 SDR file using the `SoapAdminCL` tool.

NOTE: For information about using the `SoapAdminCL` tool, see Chapter 10: "NonStop SOAP Tools" (page 194).

To generate the SDL file from the NonStop SOAP 3 SDR file, complete the following steps:

1. Set the OSS environment variable `SOAP_SDLDB_LOC` to the Guardian location where the SDR file for the NonStop SOAP 3 application is available:

   ```
   OSS> export SOAP_SDLDB_LOC=\$<datavol>.<subvol>
   ```

   For example:

   ```
   OSS> export SOAP_SDLDB_LOC=\$data.soapapp
   ```

2. Add the directory containing the `SoapAdminCL` executable image to the OSS `PATH` variable.

   ```
   OSS>export PATH=<NonStop SOAP 4 Installation Directory>/tools:$PATH
   ```

   For example:

   ```
   OSS>export PATH=/usr/tandem/nssoap/t0865h01/tools:$PATH
   ```

3. Go to the OSS directory where you want to generate the SDL file:

   ```
   OSS> cd <My SDL location>
   ```

   For example:

```
OSS> cd /home/usr/mysdl
```

4. Generate the SDL file using the SoapAdminCL tool:

   ```
   OSS> SoapAdminCL -e <SDL file name>
   ```

   For example:

   ```
   OSS> SoapAdminCL -e soapapp.sdl
   ```

   The SoapAdminCL command extracts all the services defined in the SDR file and generates the SDL file for the NonStop SOAP 3 application in the current directory.

NOTE:

- The default values of the Url and ServerAddress attributes in the sdl element are filled in with default values of /axis2c and http://www.nonstopsoap.com respectively. You must modify these values in the extracted SDL file so that it conforms to your NonStop SOAP deployment environment.
- The DDL dictionaries must be available at the Guardian location specified in the DDLDictionaryLocation attribute of the generated SDL file. If the DDL dictionaries are not available, compile the DDL file to generate the DDL dictionaries at the location mentioned in the DDLDictionaryLocation.
- You must have read access to the Guardian location where the DDL dictionaries are present.

Modifying the SDL File

After generating the SDL file, modify the Url and ServerAddress attributes in the sdl element to the NonStop SOAP 4 deployment values.

To modify the SDL file, complete the following steps:

1. Add or update the value of the Url attribute in the SDL element to access the NonStop SOAP 4 service in the SDL file:

   ```
   Url="/<url_pattern>"
   ```

   For example:

   ```
   <sdl Url="/axis2c" ServerAddress="http://127.0.0.1:8080" >
   ```

   where,

   axis2c is the string entered for the <*url_pattern*> during the installation procedure.

2. Add or update the IP address and port number of iTP WebServer integrated with the NonStop SOAP 4 deployment:

   ```
   ServerAddress="http://<ip address>:<port>"
   ```

   For example:

   ```
   <sdl Url="/axis2c" ServerAddress="http://127.0.0.1:8080" >
   ```

NOTE: You must retain the generated SDL file because NonStop SOAP 4 does not allow you to generate the SDL file from the WSDL files.

Generating the HTML Client, WSDL, XML Schema, and services.xml Files to Deploy the Service

Generate the HTML client file, the WSDL file, XML schema files, and the services.xml file to deploy the service in NonStop SOAP, using the SoapAdminCL tool:

```
OSS> SoapAdminCL -o <NonStop SOAP 4 Deployment Directory> -i <SDL file> -m
```

where,

-o <*NonStop SOAP Deployment Directory*>
specifies that the client and service must be created in the <*NonStop SOAP 4.1 Deployment Directory*>.

`-i <SDL file>`
specifies the location of the input SDL file.

`-m`
is the migration option, and specifies that a separate WSDL file must be created for all operations provided by the service in the `<NonStop SOAP 4 Deployment Directory>/client/<service_name>/wsdl` directory. The operation-specific WSDL files generated are compatible with NonStop SOAP 3.

NOTE: On running the `SoapAdminCL` tool, the SOAP request and response XML files, XML schema files, and Migration WSDL files are also generated.

On successful execution, the following files and directories are generated in the directory specified in the `-o` option (when the SDL file used as an input to the `SoapAdminCL` tool includes one service containing two operations and each operation containing two responses).

```
/client
  /<service_name>
       SoapPW_<operation1_name>.html
       SoapPW_<operation2_name>.html
       /wsdl
          SoapPW_<operation1_name>.wsdl
          SoapPW_<operation2_name>.wsdl
       /pway-xml
          SoapPW_<operation1_name>.xml
          SoapPW_<operation1_name>Response0.xml
          SoapPW_<operation1_name>Response1.xml
          SoapPW_<operation2_name>.xml
          SoapPW_<operation2_name>Response0.xml
          SoapPW_<operation2_name>Response1.xml
      /pway-xsd
          SoapPW_<operation1_name>.xsd
          SoapPW_<operation1_name>Response0.xsd
          SoapPW_<operation1_name>Response1.xsd
          SoapPW_<operation2_name>.xsd
          SoapPW_<operation2_name>Response0.xsd
          SoapPW_<operation2_name>Response1.xsd

/services
  /<service_name>
       SoapPW_<service_name>.wsdl
       services.xml
```

For example:

```
OSS> SoapAdminCL –o /home/soap_usr/nssoap -i soapapp.sdl -m
```

The directory appears as:

```
/home/soap_usr/nssoap
   /client
      /myservice
          SoapPW_myoperation1.html
          SoapPW_myoperation2.html
          /wsdl
              SoapPW_myoperation1.wsdl
              SoapPW_myoperation2.wsdl
          /pway-xml
              SoapPW_myoperation1.xml
              SoapPW_myoperation1Response0.xml
              SoapPW_myoperation1Response1.xml
              SoapPW_myoperation2.xml
              SoapPW_myoperation2Response0.xml
              SoapPW_myoperation2Response1.xml
          /pway-xsd
              SoapPW_myoperation1.xsd
```

```
                SoapPW_myoperation1Response0.xsd
                SoapPW_myoperation1Response1.xsd
                SoapPW_myoperation2.xsd
                SoapPW_myoperation2Response0.xsd
                SoapPW_myoperation2Response1.xsd


/services
   /myservice
        SoapPW_myservice.wsdl
        services.xml
```

In the example, `myservice` is the service name and `myoperation1` and `myoperation2` are the operation names.

The HTML client file, the WSDL file, XML schema files, SOAP request and response XML files, and the `services.xml` file is generated in the NonStop SOAP deployment directory. The service is now deployed in NonStop SOAP.

NOTE: Though the SDL file is not required during runtime, you must retain it in case the HTML client, the WSDL file, XML schema files, SOAP request and response XML files, or the `services.xml` file must be regenerated.

Migration Considerations

- The `RemoveRedundantElementName` SDL attribute is not supported. All the element names are unique in NonStop SOAP.
- The XSD and XML files are generated with NonStop SOAP 3 rules.
- The `SQL DATETIME HOUR TO FRACTION` DDL element is generated as `time`, instead of `string,15` data type.
- The `SOAP_MAP_DEF` attribute in the `nssoap.config` file can have the `strict or lenient` value, which map to the `responseType` attribute values in the `services.xml` file. The `lenient and liberal` values in NonStop SOAP 3 are combined as the `lenient` value in NonStop SOAP.
- The NonStop SOAP 4 server maps DDL elements accurately to the XSD data types, and generates fault response for some data values. The NonStop SOAP 3 server does not generate fault response for these data values. Correct the data values for which fault response is returned in the SOAP client.

Accessing the Migrated Application

You can access the generated files of the migrated service using the following Web address patterns:

- To access the HTML client, use the following Web address pattern:

  `http://<ip address>:<port>/<url_pattern>/client/`
  `<service_name>/<html_client_name>`

  where,

  `<ip address>:<port>`
  : is the IP address and port number of iTP WebServer where NonStop SOAP is deployed.

  `<url_pattern>`
  : is the unique string value entered during the installation procedure. For more information on *`<url_pattern>`*, see Chapter 2: “Installing NonStop SOAP” (page 36).

  `client`
  : is the directory in *`<NonStop SOAP Deployment Directory>`* where the HTML client for the services is created.

<service_name>
is the name of the service migrated from NonStop SOAP 3 and deployed in NonStop SOAP 4.

<html_client_name>
is the name of the generated HTML client that is used to access the migrated service.

- To access the endpoint Web address of the deployed service, use the following Web address pattern:

  `http://<ip address>:<port>/<url_pattern>/services/<service_name>`

  where,

  <ip address>:<port>
  is the IP address and port number of iTP WebServer where NonStop SOAP is deployed.

  <url_pattern>
  is the unique string value entered during the installation procedure. For more information on *<url_pattern>*, see Chapter 2: "Installing NonStop SOAP" (page 36).

  `services`
  is the directory in *<NonStop SOAP 4.1 Deployment Directory>* where the WSDL file and the `services.xml` file for the services is created.

  <service_name>
  is the name of the service migrated from NonStop SOAP 3 and deployed in NonStop SOAP 4.1.

- To access the operation-specific WSDL file (using the `-m` option provided by `SoapAdminCL` tool) for compatibility with NonStop SOAP 3 of the deployed service, use the following Web address pattern:

  ```
  http://<ip address>:<port>/<url_pattern>/client/<service_name>
        /wsdl/SoapPW_<operation_name>.wsdl
  ```

  where,

  <ip address>:<port>
  is the IP address and port number of iTP WebServer where NonStop SOAP 4.1 is deployed.

  <url_pattern>
  is the unique string value entered during the installation procedure. For more information on *<url_pattern>*, see Chapter 2: "Installing NonStop SOAP" (page 36).

  `client`
  is the directory in *<NonStop SOAP Deployment Directory>* where the HTML client and operation-specific WSDL files for the services is created.

  <service_name>
  is the name of the service migrated from NonStop SOAP 3 and deployed in NonStop SOAP 4.1.

  <operation_name>
  is the name of the generated operation-specific WSDL file for the migrated service.

- To access the WSDL file when generating a new NonStop SOAP client for the deployed service, use the following Web address pattern:

  `http://<ip address>:<port>/<url_pattern>/services/<service_name>?wsdl`

  where,

  <ip address>:<port>
  is the IP address and port number of iTP WebServer where NonStop SOAP is deployed.

<url_pattern>
is the unique string value entered during NonStop SOAP installation.

For more information on *<url_pattern>*, see Chapter 2: "Installing NonStop SOAP" (page 36).

`services`
is the directory in *<NonStop SOAP 4.1 Deployment Directory>* where the WSDL file and the `services.xml` file for the services is created.

<service_name>
is the name of the service migrated from NonStop SOAP 3 and deployed in NonStop SOAP 4.1.

- To access the XML schema files, use the following Web address pattern:

  `http://`*<ip address>*`:`*<port>*`/`*<url_pattern>*`/client/`*<service_name>*`/pway-xsd/`*<XML_schema_filename>*

  where,

  <ip address>`:`*<port>*
  is the IP address and port number of iTP WebServer where NonStop SOAP 4.1 is deployed.

  <url_pattern>
  is the unique string value entered during the installation procedure. For more information on *<url_pattern>*, see Chapter 2: "Installing NonStop SOAP" (page 36).

  `client`
  is the directory in *<NonStop SOAP 4 Deployment Directory>* where the client for the services is created.

  <service_name>
  is the name of the service migrated from NonStop SOAP 3 and deployed in NonStop SOAP 4.1.

  `pway-xsd`
  is the directory in *<NonStop SOAP 4 Deployment Directory>*`/client/`*<service_name>* where the XML schema files for each request and response is created.

  <XML_schema_filename>
  is the name of the generated XML schema file that is used to access the schema.

Migrating NonStop SOAP 3 Transactions

The Transaction Management module in NonStop SOAP deprecates the session object, cookie file, and other session related parameters. Therefore, all NonStop SOAP 3 services that use transactions must observe the following steps to seamlessly migrate their transaction and session management environment to NonStop SOAP:

1. Modify the SDL file.

   You must modify the following parameters in the SDL file to migrate transaction management to NonStop SOAP 4:

   - Add or update the value of the `GenerateSessionHeader` attribute to generate the XSD schema for the session header block in the WSDL file.

     ```
     GenerateSessionHeader ="yes"
     ```

   - Add or update the value of the `AbortTransactionOnFault` attribute to instruct NonStop SOAP 4 whether it needs to internally abort the transactions when the service responds with a fault.

     ```
     AbortTransactionOnFault="yes"|"no"
     ```

   - Add or update the value of the `TMFTransactionSupport` attribute to inform NonStop SOAP 4 about the level of transaction support desired by the service.

     ```
     TMFTransactionSupport="Required"|"Supports"|"Never"
     ```

   NOTE: For more information about the `TMFTransactionSupport` attribute, its permissible values, and semantics, see "Transaction Management" (page 236).

2. Generate the WSDL file and the `services.xml` file to deploy the service in NonStop SOAP , using the `SoapAdminCL` tool:

   ```
   OSS> SoapAdminCL -o <NonStop SOAP 4 Deployment Directory> -i <SDL file> -m
   ```

   where,

   `-o <NonStop SOAP 4 Deployment Directory>`
   specifies that the client and service must be created in the *<NonStop SOAP 4 Deployment Directory>*.

   `-i <SDL file>`
   specifies the location of the input SDL file.

   `-m`
   is the migration option, and specifies that a separate WSDL file must be created for all the operations provided by the service in the *<NonStop SOAP 4 Deployment Directory>*`/client/`*<service_name>*`/WSDL` directory. The operation-specific WSDL files generated are compatible with NonStop SOAP 3.

   NOTE: On running the `SoapAdminCL` tool, the SOAP request and response XML files, XML schema files, and migration WSDL files are also generated.

   On successful execution, the following files and directories are generated in the directory specified in the `-o` option (when the SDL file used as an input to the `SoapAdminCL` tool includes one service containing two operations and each operation containing two responses).

   ```
   /client
          /<service_name>
                SoapPW_<operation1_name>.html
                SoapPW_<operation2_name>.html
                /wsdl
                   SoapPW_<operation1_name>.wsdl
                   SoapPW_<operation2_name>.wsdl
                /pway-xml
                   SoapPW_<operation1_name>.xml
                   SoapPW_<operation1_name>Response0.xml
                   SoapPW_<operation1_name>Response1.xml
                   SoapPW_<operation2_name>.xml
                   SoapPW_<operation2_name>Response0.xml
                   SoapPW_<operation2_name>Response1.xml
                /pway-xsd
   ```

```
                SoapPW_<operation1_name>.xsd
                SoapPW_<operation1_name>Response0.xsd
                SoapPW_<operation1_name>Response1.xsd
                SoapPW_<operation2_name>.xsd
                SoapPW_<operation2_name>Response0.xsd
                SoapPW_<operation2_name>Response1.xsd

      /services
        /<service_name>
             SoapPW_<service_name>.wsdl
             services.xml
```

For example:

```
OSS> SoapAdminCL -o /home/soap_usr/nssoap -i soapapp.sdl -m
```

The directory appears as:

```
/home/soap_usr/nssoap
   /client
      /myservice
           SoapPW_myoperation1.html
           SoapPW_myoperation2.html
           /wsdl
               SoapPW_myoperation1.wsdl
               SoapPW_myoperation2.wsdl
           /pway-xml
               SoapPW_myoperation1.xml
               SoapPW_myoperation1Response0.xml
               SoapPW_myoperation1Response1.xml
               SoapPW_myoperation2.xml
               SoapPW_myoperation2Response0.xml
               SoapPW_myoperation2Response1.xml
           /pway-xsd
               SoapPW_myoperation1.xsd
               SoapPW_myoperation1Response0.xsd
               SoapPW_myoperation1Response1.xsd
               SoapPW_myoperation2.xsd
               SoapPW_myoperation2Response0.xsd
               SoapPW_myoperation2Response1.xsd

   /services
      /myservice
           SoapPW_myservice.wsdl
           services.xml
```

In the example, `myservice` is the service name and `myoperation1` and `myoperation2` are the operation names.

The HTML client, the WSDL file, SOAP request and response XML files, XML schema files, and the `services.xml` file is generated in the NonStop SOAP 4.1 deployment directory. The service is now deployed in NonStop SOAP 4.1.

NOTE: Though the SDL file is not required during runtime, you must retain it in case the HTML client, the WSDL file, SOAP request and response XML files, XML schema files, or the `services.xml` file need to be regenerated.

3. Engage the Transaction module with NonStop SOAP.

   Transaction management is implemented through a NonStop SOAP module. Therefore, the NonStop SOAP server administrator needs to engage the transaction module for transaction

management to work for the migrated service. The transaction management module can be engaged at the following levels:

- Global Level

  If you engage the transaction module at this level, all the services deployed in the NonStop SOAP 4 server deployment can avail themselves of the transaction management feature. To engage the transaction module at the global level, add the following line in the `axis2.xml` file located in *<NonStop SOAP 4 Deployment Directory>* :

  ```
  <module ref="transaction" />
  ```

- Service Level

  If you engage the transaction module at this level, the specific service will avail the transaction management feature. To engage the transaction module at the service level, add the line in the `services.xml` file located in *<NonStop SOAP 4 Deployment Directory>*`/services/<service_name>`:

  ```
  <module ref="transaction" />
  ```

4. Generate the clients.

   Generate the NonStop SOAP clients using the WSDL file generated by the `SoapAdminCL` tool, as shown in Step 2. There is no change to the way clients used to access the session management functionality using the session header block.

   For more information on how to utilize the session and transaction management features in NonStop SOAP using the session header block, see "Transaction Management" (page 236).

Because the transaction management module of NonStop SOAP completely deprecates the Cookie file, the following NonStop SOAP 3 specific configuration parameters have also either been deprecated or overridden.

Table 1 lists the NonStop SOAP 3 transaction management parameters and its corresponding parameters in NonStop SOAP 4.1.

Table 1 Transaction Management Parameters in NonStop SOAP 3 and NonStop SOAP 4.1.

NonStop SOAP 3 Parameter	NonStop SOAP 4.1 Parameter	Description
`SOAP_SESSION_TIMEOUT`	`TMFTimeout`	The timeout is now restricted to individual transactions. There is no separate timeout for sessions.
`SOAP_COOKIE_DELETION_INTERVAL`	None	This parameter is deprecated because there is no cookie file.

Migrating NonStop SOAP 3 User-exits

User-exits in NonStop SOAP 3 enable you to customize the default NonStop SOAP 3 behavior. NonStop SOAP 4 does not support user-exits but provides modules, handlers, and Message Receiver User Functions that enable you to customize NonStop SOAP 4 message processing. Modules, handlers, and Message Receiver User Functions are functionally equivalent to user-exits in NonStop SOAP 3.

This section describes the migration of user-exits from NonStop SOAP 3 to NonStop SOAP 4.1.

NOTE: You must have a good understanding of modules and handlers to migrate user-exits from NonStop SOAP 3 to NonStop SOAP 4.1. For information on Modules and Handlers, see "Customizing NonStop SOAP 4 Message Processing" (page 124).

Modules, handlers, and Message Receiver User Functions in NonStop SOAP 4 can be engaged at the service level and at the global level. When engaged at the service level, they enable you

to customize a service. When engaged at the global level, they enable you to customize all the services deployed in a NonStop SOAP 4 deployment.

Table 2 lists the functions of user-exits provided by NonStop SOAP 3, and the equivalent modules, handlers, and Message Receiver User Functions provided by NonStop SOAP 4.

Table 2 User-exits Mapped with Equivalent Functions in NonStop SOAP 4.1

User-exits in NonStop SOAP 3	Equivalent Handlers and Message Receiver User Functions in NonStop SOAP 4.1	Function
`pre_process()`	NonStop SOAP 4 module and handler (engaged in the `PostDispatch` phase)	Enables you to modify the input XML/SOAP document sent by the client.
`pre_service()`	`pre_service` Message Receiver User Function (fixed deployment before service invocation)	Enables you to modify the request buffer passed to the service, the PATHMON name, server class name, and NonStop process name.
`pre_marshal()`	`pre_marshal` Message Receiver User Function (fixed invocation before marshalling)	Enables you to modify the response buffer returned by the service.
`pre_marshal_header()`	`pre_marshal` Message Receiver User Function (engaged in the `MessageOut` phase)	Enables you to modify the output XML document.
`post_process()`	NonStop SOAP 4 module and handler function (engaged in the `MessageOut` phase)	Enables you to modify the XML/SOAP response to the client.

NOTE: NonStop SOAP 4.1 does not support the `skipParse` flag. NonStop SOAP 3 supported this flag, which indicated NonStop SOAP 3 to skip the parsing of the request.

Migrating the NonStop SOAP 3 User-exit Samples

The NonStop SOAP 3 user-exit samples you can migrate are:

- The `mod_empdb` sample: This sample shows the migration of the `pre_process()` and `post_process()` user-exits.
- The `empdb_MRUF` sample: This sample shows the migration of the `pre_service()` and `pre_marshal()` user-exits.
- The `skipMarshal` sample: This sample shows the migration of the `skipMarshal` flag.

NOTE: The NonStop SOAP 3 user-exits samples are located in the *<NonStop SOAP 3 Installation Directory>*`/samples/pathway/UserExits` directory.

This section describes the following steps:

- "Migrating the `pre_process()` and `post_process()` user-exits using `mod_empdb`" (page 62)
- "Migrating the `pre_service()` and `pre_marshal()` user-exits in NonStop SOAP 3 to Message Receiver User Functions in NonStop SOAP 4.1" (page 71)

Migrating the `pre_process()` and `post_process()` user-exits using `mod_empdb`

The `mod_empdb` sample shows the migration of the `pre_process()` and `post_process()` user-exits in NonStop SOAP 3 to the corresponding `mod_empdb` module in NonStop SOAP 4.1.

This section describes the following:

- "Implementation of the `pre_process()` user-exit in NonStop SOAP 3" (page 63)
- "Implementation of the `post_process()` user-exit in NonStop SOAP 3" (page 63)
- "Migration of the `pre_process()` and `post_process()` user-exits in NonStop SOAP 4" (page 64)

Implementation of the `pre_process()` user-exit in NonStop SOAP 3

In NonStop SOAP 3, the `pre_process()` user-exit logs the name of the service being invoked in an audit file. The `pre_process()` user-exit is implemented in the *<NonStop SOAP 3 Installation Directory>*`/samples/pathway/UserExits/SoapUEHandler_impl.cpp` file, using the `SoapUEHandler_Generic::audit()` method. The `SoapUEHandler_Generic::audit()` function is called from the `SoapUEHandler_Generic::pre_process()` method.

The following code sample shows the business logic implemented for the `pre_process()` user-exit.

```
1 void
2 SoapUEHandler_Generic::audit() {
3   if (SoapUEHandler_Generic::auditFP == NULL) {
4    SoapUEHandler_Generic::auditFP = fopen("emp.audit", "a+");
5   if (SoapUEHandler_Generic::auditFP == NULL) {
6      gSoapError.log("Error while opening the audit file %ld\n", errno);
7      return;
8     }
9     gSoapError.log("Opened the audit file\n");
10  }
11  fprintf(SoapUEHandler_Generic::auditFP, "%s\n", serviceName);
12  fflush(SoapUEHandler_Generic::auditFP);
13 }
```

Line 3	Checks for an existing file.
Line 4	Opens a new file (with append permissions) using the `fopen` command, if the `emp.audit` file does not exist.
Line 5 – 8	Error handling of File creation failure error.
Line 11	Logs the `serviceName` in the `emp.audit` file.
Line 12	Flush the file pointer to complete the write.

Implementation of the `post_process()` user-exit in NonStop SOAP 3

In NonStop SOAP 3, the `post_process()` user-exit adds the following XML processing instruction to the response message for the `empdb` service:

```
<?xml-stylesheet type="text/xsl" href="empinfo.xsl" ?>
```

The `post_process()` user-exit is implemented in the *<NonStop SOAP 3 Installation Directory>*`/samples/pathway/UserExits/SoapUEHandler_impl.cpp` file using the `stuffXSLT()` method. The `stuffXSLT()` function is called from the `SoapUEHandler_Generic::post_process()` method.

The following code sample shows the business logic implemented for the `post_process()` user-exit:

```
1  Static void  stuffXSLT(OutputXML &outputXml, char *XSLTFileName)
2 {

3  if (!XSLTFileName ||  !XSLTFileName[0]) {
4    return;
5  }
```

```
6  int iPos = 0;

7  iPos = outputXml.find("?>");
8  if (iPos > 0) {

9    char replStr[500];

10    memset(replStr, 0, sizeof(replStr));
11    strcpy(replStr, "\n<?xml-stylesheet type=\"text/xsl\" href=\"");

12    strcat(replStr, XSLTFileName);
13    strcat(replStr, "\" ?>\n");

14    outputXml.replace(iPos + 2,1, replStr);

15  }

16 }
```

Line 3-5	Checks if the Extensible Stylesheet Language (XSL) template file specified in the argument exists.
Line 7	Invokes the `outputXml.find` function to search for the `?>` tag.
Line 8-15	Builds the Extensible Stylesheet Language Tranns (XSLT) processing instruction to be appended to the response XML message.

Migration of the `pre_process()` and `post_process()` user-exits in NonStop SOAP 4

NonStop SOAP 3 user-exits cannot be migrated directly to NonStop SOAP 4.1. You must develop a module and handler to implement the same functionality in NonStop SOAP 4.1.

To migrate the `pre_process()` user-exit to an equivalent module and handler in NonStop SOAP 4.1:

- Implement the `empdbPreProcessHandler` as a part of the `mod_empdb` module.
- Attach the handler to the `PostDispatch` phase of the `inflow`.

To migrate the `post_process()` user-exit to an equivalent module and handler in NonStop SOAP 4.1:

- Implement `empdbPostProcessHandler` handler as a part of the `mod_empdb` module.
- Attach the handler to the `MessageOut` phase of the `outflow`.

Developing the `mod_empdb` module for NonStop SOAP 4.1 involves the following tasks:

1. Running the `SoapAdminCL` tool to create the module stub and configuration files (page 64)
2. Migrating the `pre_process()` user-exit business logic (page 66)
3. Migrating the `post_process()` user-exit business logic (page 67)
4. Compiling the `mod_empdb` module (page 69)
5. Deploying the `mod_empdb` module (page 69)
6. Running the NonStop SOAP `empdb` sample with the `mod_empdb` module (page 70)

Running the `SoapAdminCL` tool to create the module stub and configuration files

The first step is to generate the module stub and configuration files required to implement the `mod_empdb` module. You must have the `empdb` service deployed in NonStop SOAP 4.1. For more

information about deploying the empdb service, see Chapter 2: "Installing NonStop SOAP" (page 36).

- The empdb service is located in:

  <NonStop SOAP 4 Deployment Directory>/services/empdb/

- The SDL file for the empdb service is located in:

  <NonStop SOAP 4 Deployment Directory>/services/empdb/src/empsdl.xml

To generate the module stub and configuration files, complete the following steps:

1. Open the empsdl.xml SDL file for the empdb service located at *<NonStop SOAP 4 Deployment Directory>*/services/empdb/src and modify the file using an editor of your choice.
2. Edit the following parameters in the empsdl.xml file:

   Url

   Update the Url attribute of the SDL element with the Web address pattern entered while running the deploy.sh setup script for the deployment. This attribute must have a valid relative path to the ServerAddress attribute.

   By default, the value of the Url attribute is /axis2c.

   For example:

   ```
   Url= "/mytest"
   ```

   where,

   mytest
   : is the Web address pattern entered while running the deploy.sh deployment script to create the NonStop SOAP deployment directory.

   ServerAddress

   Update the ServerAddress attribute with the IP and port number of the iTP WebServer where the NonStop SOAP has been deployed. By default, the value of the ServerAddress attribute is http://www.nonstopsoap.com.

   For example:

   ```
   ServerAddress ="http://<ip address>:<port>"
   ```

   GenerateModuleHandlerFiles

   Set the GenerateModuleHandlerFiles attribute of the <Service> element to "yes".

   For example:

   ```
   <Service Name="empdb"
         ServerClassName="empdb"
         GenerateModuleHandlerFiles="yes"
         ...
   ```

3. Add the directory containing the SoapAdminCL executable image to the OSS PATH variable, if the path has not been set.

   ```
   OSS>export PATH=<NonStop SOAP 4 Installation Directory>/tools:$PATH
   ```

   For example:

   ```
   OSS>export PATH=/usr/tandem/nssoap/t0865h01/tools:$PATH
   ```

4. Run the SoapAdminCL tool using the OSS command:

   ```
   OSS> SoapAdminCL –i <SDL file for the empdb service>
                    -o <NonStop SOAP 4 Deployment Directory>
   ```

   For example:

   ```
   OSS>SoapAdminCL –i empsdl.xml –o /home/usr1/mynssoap
   ```

This command will generate only the module handler files.

NOTE:

- Use the `-f` option with the `SoapAdminCL` command to overwrite the existing files. If you do not specify the `-f` option, `SoapAdminCL` will generate new files based on the SDL attributes and report a failure for the existing files which are not overwritten.

5. Verify that the following directory structure is created in *<NonStop SOAP Deployment Directory>*:

```
<NonStop SOAP 4 Deployment Directory>
    /modules
      /mod_empdb
         module.xml
         /src
            Makefile
            mod_empdb.c
            mod_empdb.h
            empdb_pre_process_handler.c
            empdb_post_process_handler.c
```

where,

`module.xml`
: is the configuration file for the `mod_empdb` module.

`Makefile`
: is the Makefile for the `mod_empdb` module.

`mod_empdb.c`
: is the C skeleton source file that implements the interface between the NonStop SOAP 4 server and the `mod_empdb` module.

`mod_empdb.h`
: is the header file for the `mod_empdb.c` source.

`empdb_pre_process_handler.c`
: is the C stub file where you must implement the `pre_process` business logic for the handler. In this example, the business logic logs the name of the service invoked in an audit file.

`empdb_post_process_handler.c`
: is the C stub file where you must implement the `post_process` business logic for the handler. In this example, the business logic inserts an XML processing instruction in the response XML message.

Migrating the `pre_process()` user-exit business logic

To migrate the business logic for the `pre_process()` user-exit to the pre_process handler, complete the following steps:

1. Open the `empdb_pre_process_handler.c` stub file generated in the *<NonStop SOAP 4 Deployment Directory>/modules/mod_empdb/src* directory.
2. Implement the business logic of the `pre_process()` user-exits to the `axutil_empdb_pre_process_invoke()` function.

The following code sample shows the business logic implemented in the `axutil_empdb_pre_process_invoke()` function in the `empdb_pre_process_handler.c` file:

```
1  axis2_status_t AXIS2_CALL
2  axutil_empdb_pre_process_invoke(
3      struct axis2_handler * handler,
4      const axutil_env_t * env,
5      struct axis2_msg_ctx * msg_ctx)
6  {
```

```
 7      axiom_soap_envelope_t *soap_envelope = NULL;
 8      axis2_svc_t* service_struct = NULL;
 9      axis2_op_t* operation_struct = NULL;
10      const axis2_char_t* service_name = NULL;
11      const axis2_char_t* operation_name = NULL;
12      const axutil_qname_t* operation_qname = NULL;
13
14      int bytes_written = 0;
15      FILE* fp = NULL;
16
17      AXIS2_ENV_CHECK(env, AXIS2_FAILURE);
18      AXIS2_PARAM_CHECK(env->error, msg_ctx, AXIS2_FAILURE);
19      AXIS2_LOG_DEBUG(env->log,AXIS2_LOG_SI, "Starting empdb pre_process handler .........");
20
21      service_struct= axis2_msg_ctx_get_svc (msg_ctx,env);
22      service_name = axis2_svc_get_name(service_struct, env);
23
24      fp = fopen("../modules/mod_empdb/emp.audit", "a+");
25      if(!fp){
26              AXIS2_LOG_DEBUG(env->log,AXIS2_LOG_SI,"Error in opening theemp.audit file");
27              fclose(fp);
28              return AXIS2_FAILURE;
29      }
30      bytes_written = fwrite(axutil_strcat(env,service_name,"\n",NULL),
31                             sizeof(axis2_char_t), axutil_strlen(service_name), fp);
32      fclose(fp);
33
34  /* The following section should be used when one needs to implement
35  the business logic at the operation level*/
36      /*operation_struct = axis2_msg_ctx_get_op(msg_ctx,env);*/
37      /*operation_qname = axis2_op_get_qname(operation_struct, env);*/
38      /*operation_name = axutil_qname_get_localpart(operation_qname, env);*/
39            /*if(!axutil_strcmp(operation_name,"EmpInfo")){*/
40                    /* TODO :: Insert your business logic here */
41                  /*}*/
42           /*if(!axutil_strcmp(operation_name,"EmpDel")){*/
43                    /* TODO :: Insert your business logic here */
44                  /*}*/
45           /*if(!axutil_strcmp(operation_name,"EmpAdd")){*/
46                    /* TODO :: Insert your business logic here */
47                  /*}*/
48      return AXIS2_SUCCESS;
49  }
```

Line 1 - 13	The `SoapAdminCL` tool generates the code that lists the function and variable declaration.
Line 15	Initializes a file pointer variable.
Line 17 - 19	Checks for errors.
Line 21	Fetches the service structure that includes details about the invoked service.
Line 22	Retrieves the service structure name fetched in step 21.
Line 24 - 32	Opens the `emp.audit` file and writes the service name in it.
Line 36 - 47	This code is useful when you have a business logic that needs to be implemented at the operation level.

NOTE: The NonStop SOAP distribution includes the complete implementation of the `pre_process()` handler. You can either write your own business logic or copy the source code of the `pre_process()` handler from the NonStop SOAP installation directory using the following command:

```
OSS>cp <NonStop SOAP 4 Installation Directory>/sample_services/modules
       /mod_empdb/src/empdb_pre_process_handler.c
       <NonStop SOAP 4 Deployment Directory>/modules
       /mod_empdb/src/empdb_pre_process_handler.c
```

Migrating the `post_process()` user-exit business logic

To migrate the business logic for the `post_process()` user-exit to the post_process handler, complete the following steps:

1. Open the `empdb_post_process_handler.c` stub file generated in the *<NonStop SOAP Deployment Directory>*`/modules/mod_empdb/src` directory.

2. Implement the business logic of the `post_process()` user-exits to the `axutil_empdb_post_process_invoke()` function.

The following code sample shows the business logic implemented in the `axutil_empdb_post_process_invoke()` function in the `empdb_post_process_handler.c` file.

```
 1  axis2_status_t AXIS2_CALL
 2  axutil_empdb_post_process_invoke(
 3      struct axis2_handler * handler,
 4      const axutil_env_t * env,
 5      struct axis2_msg_ctx * msg_ctx)
 6  {
 7      axiom_soap_envelope_t *soap_envelope = NULL;
 8      axis2_svc_t* service_struct = NULL;
 9      axis2_op_t* operation_struct = NULL;
10      const axis2_char_t* service_name = NULL;
11      const axis2_char_t* operation_name = NULL;
12      const axutil_qname_t* operation_qname = NULL;
13
14      axis2_char_t* target = "xml-stylesheet";
15      axis2_char_t* data = "type='text/xsl' href='../../empdb.xsl'";
16      axutil_hash_t* processing_instructions_ht = NULL;
17
18      AXIS2_ENV_CHECK(env, AXIS2_FAILURE);
19      AXIS2_PARAM_CHECK(env->error, msg_ctx, AXIS2_FAILURE);
20      AXIS2_LOG_DEBUG(env->log,AXIS2_LOG_SI, "Starting empdb post_processhandler .........");
21
22      processing_instructions_ht = axutil_hash_make(env);
23      axutil_hash_set(processing_instructions_ht,target,
24                      AXIS2_HASH_KEY_STRING , data);
25      axis2_msg_ctx_set_processing_instructions(msg_ctx, env,
26                                                processing_instructions_ht);
27
28  /* The following section should be used when one needs to implement
29  the business logic at the service level*/
30
31  /*    service_struct= axis2_msg_ctx_get_svc (msg_ctx,env);*/
32  /*    service_name = axis2_svc_get_name(service_struct, env); */
33  /* The following section should be used when one needs to implement
34  the business logic at the operation level*/
35
36      /*operation_struct = axis2_msg_ctx_get_op(msg_ctx,env);*/
37      /*operation_qname = axis2_op_get_qname(operation_struct, env);*/
38      /*operation_name = axutil_qname_get_localpart(operation_qname, env);*/
39            /*if(!axutil_strcmp(operation_name,"EmpInfo")){*/
40                      /* TODO :: Insert your business logic here */
41                  /*}*/
42          /*if(!axutil_strcmp(operation_name,"EmpDel")){*/
43                      /* TODO :: Insert your business logic here */
44                  /*}*/
45          /*if(!axutil_strcmp(operation_name,"EmpAdd")){*/
46                      /* TODO :: Insert your business logic here */
47                  /*}*/
48      return AXIS2_SUCCESS;
49  }
```

Line 1 - 13	The `SoapAdminCL` tool generates the code that lists the function and variable declaration.
Line 14 - 16	Initializes variables.
Line 18 - 20	Checks for errors.
Line 22 - 24	Creates the processing instruction.
Line 25 - 26	Adds the processing instruction to the SOAP message.
Line 31 - 47	This code is useful when you have a business logic that needs to be implemented at the operation level.

NOTE: The NonStop SOAP distribution includes the complete implementation of the `post_process()` handler. You can either write your own business logic or copy the source code of the `post_process()` handler from the NonStop SOAP 4.1 installation directory using the following command:

```
OSS>cp <NonStop SOAP 4 Installation Directory>/sample_services/modules
       /mod_empdb/src/empdb_post_process_handler.c
       <NonStop SOAP 4 Deployment Directory>/modules
       /mod_empdb/src/empdb_post_process_handler.c
```

Compiling the `mod_empdb` module

After implementing the business logic in the `axutil_empdb_pre_process_invoke()` and `axutil_empdb_post_process_invoke()` methods, you must compile the `mod_empdb` module.

To compile the `mod_empdb` module, complete the following steps:

1. Go to the *<NonStop SOAP 4 Deployment Directory>*`/modules/mod_empdb/src` directory using the OSS command:

   ```
   OSS> cd <NonStop SOAP Deployment Directory>/modules/mod_empdb/src
   ```

   For example:

   ```
   OSS> cd /home/usr1/mynssoap/modules/mod_empdb/src
   ```

2. Set the environment variable AXIS2C_HOME to *<NonStop SOAP 4 Deployment Directory>* directory using the OSS command:

   ```
   OSS> export AXIS2C_HOME=<NonStop SOAP 4 Deployment Directory>
   ```

   For example:

   ```
   OSS> export AXIS2C_HOME=/home/usr1/mynssoap
   ```

3. To compile and generate the module DLL, run the `make` command on the OSS prompt:

   ```
   OSS>make
   ```

   On successful completion of the `make` command, the `libaxis2_mod_empdb.so` file is created in the *<NonStop SOAP 4 Deployment Directory>*`/modules/mod_empdb` directory.

Deploying the `mod_empdb` module

After creating the `libaxis2_mod_empdb.so` file, complete the following steps to deploy the module:

1. Attach the `pre_process()` handler to the `PostDispatch` phase. To do so, edit the `module.xml` file in *<NonStop SOAP Deployment Directory>*`/modules/mod_empdb` to set the `phase` attribute of the `order` element in `inflow` to `PostDispatch`.

   ```
   <inflow>
     <handler name="empdbPreProcessHandler" class="axis2_mod_empdb">
       <order phase="PostDispatch"/>
     </handler>
   </inflow>
   ```

2. Attach the `post_process()` handler to the `MessageOut` phase. To do so, in the `module.xml` file, set the `phase` attribute of the `order` element in `outflow` to `MessageOut`.

   ```
   <outflow>
     <handler name="empdbPostProcessHandler" class="axis2_mod_empdb">
       <order phase="MessageOut"/>
     </handler>
   </outflow>
   ```

3. Add the module name to the `ref` attribute of the `module` element in the `services.xml` file in the *<NonStop SOAP 4 Deployment Directory>*`/services/empdb` directory.

   ```
   <module ref="mod_empdb"/>
   ```

where,

mod_empdb
: is the name of the module attached.

4. Restart iTP WebServer using the OSS commands:

```
OSS> cd <iTP_Installation_Directory>/conf
OSS> ./restart
```

Running the NonStop SOAP empdb sample with the mod_empdb module

To verify if the empdbPreProcessHandler handler of the mod_empdb module has been engaged, complete the following steps:

1. Access the EmpInfo, EmpAdd, or EmpDel operations of the empdb application using the HTML client available at the following Web address:

   ```
   http://<ip address>:<port_number>/<url_pattern>/client/empdb
   ```

   where,

   <ip address>:<port_number>
   : is the IP address and port number of iTP WebServer integrated with NonStop SOAP.

   <url_pattern>
   : is the URL pattern entered during NonStop SOAP installation.

2. On receiving the response from the service, verify if the emp.audit file is created in the <NonStop SOAP Deployment Directory>/modules/mod_empdb directory.
3. Open the emp.audit file and verify that the entry with the service name empdb is present.

To verify if the empdbPostProcessHandler handler of the mod_empdb module is engaged, complete the following steps:

1. Copy the empdb.xsl file from the <NonStop SOAP 4 Installation Directory> directory to the <iTP_Installation_Directory>/root directory using the OSS command:

   ```
   OSS> cp <NonStop SOAP 4 Installation Directory>/sample_services/modules/mod_empdb/empdb.xsl
         <iTP WebServer Installation Directory>/root/empdb.xsl
   ```

2. Access the EmpInfo operation of the empdb application using the HTML client.

   The response is a formatted output based on the style sheet specifications.

   Figure 8 shows the formatted output.

Figure 8 Stylesheet Based Formatted Output

Migrating the pre_service() and pre_marshal() user-exits in NonStop SOAP 3 to Message Receiver User Functions in NonStop SOAP 4.1

The empdb_MRUF sample shows the migration of the pre_service() and pre_marshal() user-exits in NonStop SOAP 3 to the corresponding Message Receiver User Functions attached to the empdb sample service deployed in NonStop SOAP 4.1. For more information about deploying the empdb service, see Chapter 2: "Installing NonStop SOAP" (page 36).

The following sections describe:

- "Implementation of the pre_service() and pre_marshal() user-exit in NonStop SOAP 3" (page 71)
- "Migration of pre_service() and pre_marshal() user-exits in NonStop SOAP 4.1" (page 72)

Implementation of the pre_service() and pre_marshal() user-exit in NonStop SOAP 3

The pre_service() user-exit in NonStop SOAP 3 sets the least significant digit of the branchnum parameter to 0 if it is set to 9 and the operation being invoked is EmpAdd. The pre_service() user-exit is implemented in the SoapUEHandler_impl.cpp file located in the *<NonStop SOAP 3 Installation Directory>*/samples/pathway/UserExits directory using the SoapUEHandler_Generic::pre_service() method.

The following code implements the business logic for the pre_service() user-exit in NonStop SOAP 3:

```
1 long
2 SoapUEHandler_Generic::pre_service(ServiceReqResponse *req,
                                     SoapFault *sf)
3 {
4 gSoapTrace.log("Entered SoapUEHandler_Generic::pre_service\n");
5 gSoapTrace.log("DDL Buf Len: %ld\n", req->getDDLBufferLen());

6  if (!strcmp(serviceName, "EmpAdd")) {
7   employee_add_def *eadd_req;

8    eadd_req = (employee_add_def *) req->getDDLBuffer();

9    // If the branch number is not set.. set it to '9'
10      if (eadd_req->employee_info.branchnum[0] == 9) {
11      eadd_req->employee_info.branchnum[0] = '0';
12    }
13  }

14  gSoapTrace.log("Leaving SoapUEHandler_Generic::pre_service\n");

15  return 0;
   }
```

Line 6	Checks if the service name is EmpAdd.
Line 8	Extracts the request structure.
Line 10-11	Sets branchnum to 0 if it is 9.

The pre_marshal() user-exit in NonStop SOAP 3 sets the skipMarshal flag, which skips the marshalling routine of NonStop SOAP 3 if the service name is EmpAdd.

This functionality is implemented in the SoapUEHandler_impl_2.cpp file located in the *<NonStop SOAP 3 Installation Directory>*/samples/pathway/UserExits directory using the SoapUEHandler_Generic::pre_marshal() method, which implements the business logic for the pre_marshal() user-exits.

The following code sample shows the implementation of business logic for the `pre_marshal()` Message Receiver User Function:

```
1  long
2  SoapUEHandler_Generic::pre_marshal(ServiceReqResponse *resp,
3                                     SoapFault          *sf)
4  {
5   if (!strcmp(serviceName, "EmpAdd")) {
6     getServiceEnv()->skipMarshal();
7   }
8
9   return 0;
10 }
```

Line 5	Checks if the service name is `EmpAdd`.
Line 6	Sets the `skipMarshal` flag.

NOTE: When migrating user-exits from NonStop SOAP 3 to NonStop SOAP 4 Message Receiver User Functions, take the following into consideration:

- NonStop SOAP 3 user-exit codes are written using C++ classes but modules and handlers in NonStop SOAP 4 must be implemented using the C programming language.
- NonStop SOAP 3 works through environment variables. NonStop SOAP 4 works on AXIOM nodes and configuration context structures. Therefore, you must derive the service name from these structures.

Migration of `pre_service()` and `pre_marshal()` user-exits in NonStop SOAP 4.1

The `pre_service()` and `pre_marshal()` user-exits in NonStop SOAP 3 must be migrated to an equivalent Message Receiver User Function in NonStop SOAP 4.1.

Developing the `empdb_MRUF` sample for NonStop SOAP involves the following tasks:

1. Running the `SoapAdminCL` tool to generate the `empdb_MRUF` stub files (page 72)
2. Migrating the business logic for the `pre_service()` user-exit (page 73)
3. Compiling the Message Receiver user Functions (page 75)
4. Deploying the Message Receiver user Functions (page 75)
5. Running the NonStop SOAP 4 `empdb` sample with `empdb_MRUF` (page 75)

Running the `SoapAdminCL` tool to generate the `empdb_MRUF` stub files

The first step is to generate the `empdb_MRUF` stub files using the `SoapAdminCL` tool. You must have the `empdb` service deployed in NonStop SOAP. For more information about deploying the `empdb` service, see Chapter 2: "Installing NonStop SOAP" (page 36).

- The `empdb` service is located in:

  <NonStop SOAP 4.1 Deployment Directory>`/services/empdb/`

- The SDL file for the `empdb` service is located in:

  <NonStop SOAP 4.1 Deployment Directory>`/services/empdb/src/empsdl.xml`

To generate the `empdb_MRUF` stub files, complete the following steps:

1. Open the empsdl.xml file and complete the following:

   Set the GenerateMessageReceiverUserFunctionsFiles attribute in the <Service> element to "yes".

   For example:

   ```
   <Service Name="empdb"
          ServerClassName="empdb"
          GenerateMessageReceiverUserFunctionsFiles="yes"
          ...
   ```

2. Add the directory containing the SoapAdminCL executable image to the OSS PATH variable, if the path has not been previously set.

   ```
   OSS>export PATH=<NonStop SOAP 4.1 Installation Directory>/tools:$PATH
   ```

   For example:

   ```
   OSS>export PATH=/usr/tandem/nssoap/t0865h01/tools:$PATH
   ```

3. Run the SoapAdminCL tool to generate the required stub files.

   ```
   OSS> SoapAdminCL -i <name of the empdb SDL file>
                    -o <NonStop SOAP Deployment Directory>
   ```

   For example:

   ```
   OSS>SoapAdminCL -i empsdl.xml -o /home/usr1/mynssoap
   ```

   NOTE:

   - Use the -f option with the SoapAdminCL command to overwrite the existing files. If you do not specify the -f option, SoapAdminCL will generate new files based on the SDL attributes and report a failure for the existing files which are not overwritten.

4. Verify that the following directory structure is created in <NonStop SOAP 4 Deployment Directory>:

   ```
   <NonStop SOAP 4 Deployment Directory>
     /services
       /empdb
          /MRUF_src
           Makefile
           empdb_MRUF.c
           empdb_MRUF.h
     SoapPW_empdb.wsdl
     services.xml
   ```

   where,

   Makefile
   : is the Makefile to build the empdb_MRUF.c code

   empdb_MRUF.c
   : is the source file that implements the pre_service() and pre_marshal() business logic for the handler.

   empdb_MRUF.h
   : is the header file for the empdb_MRUF.c file.

Migrating the business logic for the pre_service() user-exit

After the empdb_MRUF stub files are generated, migrate the business logic for the pre_service() user-exit to Message Receiver User Function. To do so, edit the empdb_MRUF.c file located in the <NonStop SOAP 4.1 Deployment Directory>/services/empdb/MRUF_src directory to implement the business logic of the pre_service user-exit in the empdb_pre_service_function().

The following code sample shows the business logic implemented in the `empdb_pre_service_function()` in the `empdb_MRUF.c` file.

```
1 int empdb_pre_service_function(
2                       axis2_MessageReceiverUserFunctions_t *MessageReceiverUserFunctions,
3                       axis2_char_t * payload,
4                      const axutil_env_t * env,
5                      int ReqLen,
6                      struct axis2_msg_ctx * in_msg_ctx,
7                      struct axis2_msg_ctx * out_msg_ctx){
8
9       axis2_char_t* operation_name = NULL;
10
11       operation_name = axis2_msg_recv_get_operationName(MessageReceiverUserFunctions, env);
12       if(!axutil_strcmp(operation_name,"EmpAdd")){
13        if (payload)
14    {
15       if(payload[ReqLen-2] == '0' && payload[ReqLen-1] == '9'){
16          payload[ReqLen-1] = '0';
17       }
18    }
19       }
20       if(!axutil_strcmp(operation_name,"EmpDel")){
21           /* TODO :: Insert your business logic here */
22       }
23       if(!axutil_strcmp(operation_name,"EmpInfo")){
24           /* TODO :: Insert your business logic here */
25       }
26       return ReqLen;
27}
```

Line 11	Gets the operation name.
Line 12	Checks whether the operation name is `EmpAdd`.
Line 15-16	Modifies the `branchnum` value.

NOTE: NonStop SOAP provides the complete implementation of the `pre_service` Message Receiver User Function. Copy the source code of the `pre_service` Message Receiver User Function from the NonStop SOAP installation location using the following command:

```
OSS> cp <NonStop SOAP 4 Installation Directory>/sample_services/modules/empdb_MRUF
      /src/empdb_MRUF.c
        <NonStop SOAP 4 Deployment Directory>/services/
        empdb/MRUF_src/empdb_MRUF.c
```

The business logic for the `pre_marshal` handler is also implemented in the `empdb_pre_marshal_function( )` method in the `empdb_MRUF.c` file. The following code sample shows the business logic for the `pre_marshal( )` method:

```
1int empdb_pre_marshal_function(
2                   axis2_MessageReceiverUserFunctions_t *MessageReceiverUserFunctions,
3                   axis2_char_t * payload,
4                   const axutil_env_t * env,
5                   int ResLen,
7                   struct axis2_msg_ctx * in_msg_ctx,
8                   struct axis2_msg_ctx * out_msg_ctx)
9{
10
11   axis2_char_t* operation_name = NULL;
12   operation_name = axis2_msg_recv_get_operationName(MessageReceiverUserFunctions, env);
13
14         if(!axutil_strcmp(operation_name,"EmpInfo")){
15      /* TODO :: Insert your business logic here */
16         }
17        if(!axutil_strcmp(operation_name,"EmpDel")){
18      /* TODO :: Insert your business logic here */
19        }
20        if(!axutil_strcmp(operation_name,"EmpAdd")){
21         axis2_msg_recv_set_skipMarshal(MessageReceiverUserFunctions, env, AXIS2_TRUE);
22        }
23
24    return ResLen;
```

```
25}
```

Line 12	Gets the operation name.
Line 20	Checks whether the operation name is `EmpAdd`.
Line 21	Sets the `skipMarshal` flag.

Compiling the Message Receiver user Functions

To build `empdb_MRUF`, complete the following steps:

1. Go to the source code location of Message Receiver User Functions using the OSS command:

   ```
   OSS> cd <NonStop SOAP 4 Deployment Directory>/
           services/empdb/MRUF_src/
   ```

2. Set the environment variable AXIS2C_HOME to *<NonStop SOAP 4 Deployment Directory>* directory using the OSS command:

   ```
   OSS> export AXIS2C_HOME=<NonStop SOAP 4 Deployment Directory>
   ```

3. Run the `make` command to compile and generate the module DLL:

   ```
   OSS> make
   ```

   On successful completion of the `make` command, the `libaxis2_MRUF_empdb.so` module is created in the *<NonStop SOAP 4 Deployment Directory>*`/MRUserFunctions` directory.

Deploying the Message Receiver user Functions

After creating the Message Receiver user Function DLL , you must engage the `libaxis2_MRUF_empdb.so` module with the `empdb` application in the NonStop SOAP 4 deployment.

1. Verify that the following entry is present in the `services.xml` file in the *<NonStop SOAP 4 Deployment Directory>*`/services/empdb` directory to engage the Message Receiver User Functions.

   ```
   <parameter name="MessageReceiverUserFunctions">axis2_MRUF_empdb</parameter>
   ```

   where,

   `axis2_MRUF_empdb`
   : is the name of the DLL for Message Receiver User Functions. NonStop SOAP 4 generates the name of the DLL by adding `lib` as the prefix and `so` as the suffix.

2. Restart iTP WebServer to engage the new Message Receiver User Functions:

   ```
   OSS> cd <iTP_Installation_Directory>/conf
   OSS> ./restart
   ```

Running the NonStop SOAP 4 `empdb` sample with `empdb_MRUF`

To verify if the `empdb_MRUF` is engaged, complete the following steps:

1. Verify if the `pre_service()` Message Receiver User Function is attached correctly with NonStop SOAP 4:
   1. Access the `EmpAdd` operation of the `empdb` application using the HTML client. Ensure that you enter the value of the `branchnum` as 09. This will generate an empty body response.
   2. Access the `EmpInfo` operation for the same `empdb` record.
   3. On receiving the response, verify if the value of the `branchnum` is set to 00.

2. Verify if the `pre_marshal()` Message Receiver User Function is attached correctly with NonStop SOAP 4:
    1. Access the `EmpAdd` operation of the `empdb` application using the HTML client.
    2. On receiving the response, you will receive an empty SOAP body indicating that the marshal procedure of the response is skipped.

       This indicates that the `pre_service()` and `pre_marshal()` Message Receiver User Functions are successfully engaged with the `empdb` service.

4 Getting Started with NonStop SOAP 4

This chapter describes the procedure to deploy a Pathway application and a NonStop process as a Web service.

For information on developing DLL as a Web service, see Chapter 5: "NonStop SOAP 4 Service APIs" (page 85).

NonStop SOAP 4 can be used to develop SOAP clients that can communicate with SOAP services. For information on developing NonStop SOAP 4 clients, see Chapter 6: "NonStop SOAP 4 Client APIs" (page 105).

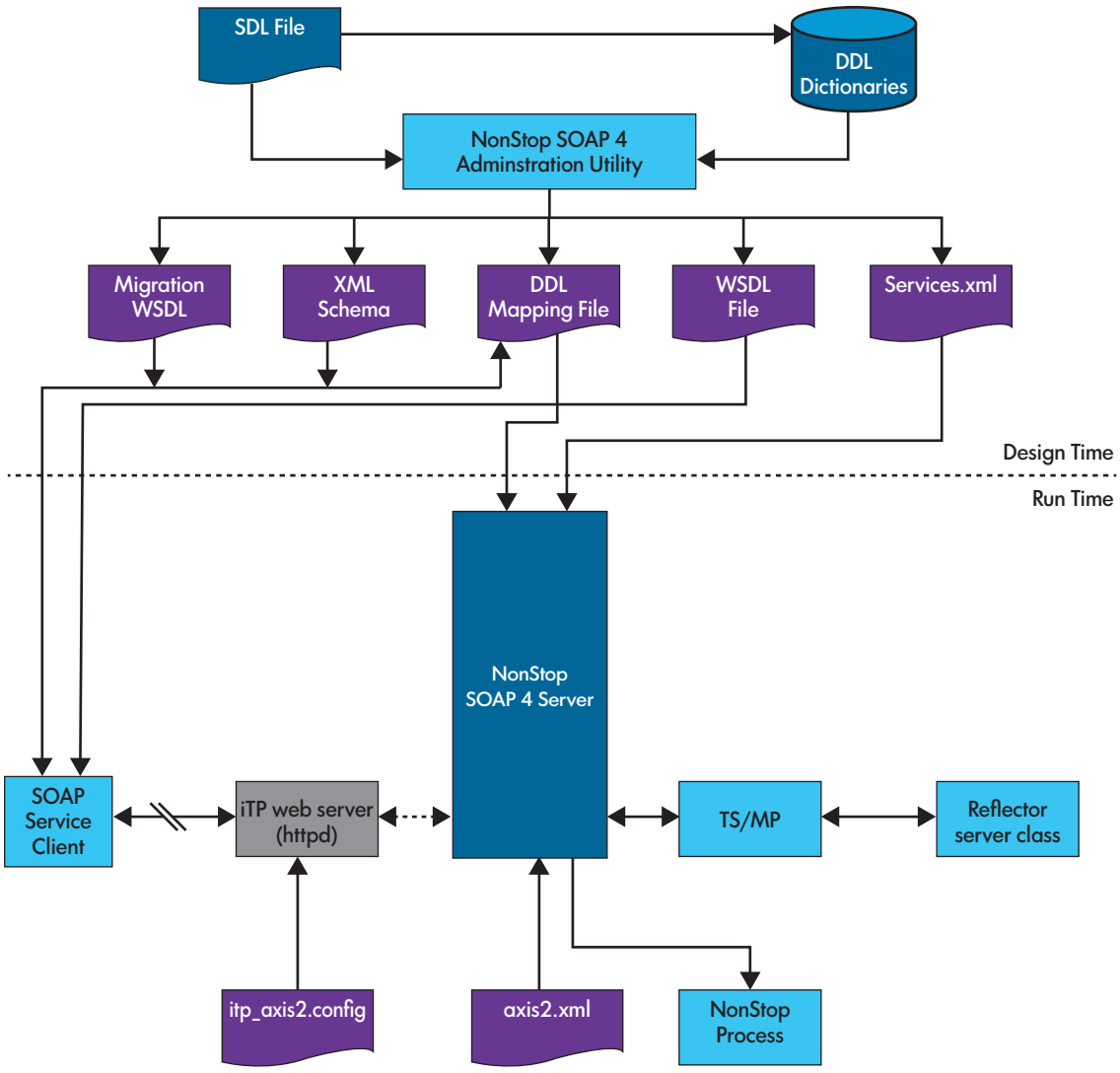
This chapter includes the following topic:

"Deploying a Pathway Application and a NonStop Process as a Web Service " (page 77).

Deploying a Pathway Application and a NonStop Process as a Web Service

This section describes the procedure to deploy a pathway application as a Web service. Figure 9 shows the Web service deployment and message flow between Web client , Pathway application and Nonstop process.

Figure 9 Web Service deployment of NonStop Process/Pathway Application



NOTE: The following example has the steps to deploy the Web service using the sample reflector Pathway application. However, if you already have an existing pathway server class and a corresponding WSDL, then making changes to the `services.xml` should suffice in being able to get the server class deployed as a Web service.

The steps to deploy the Web service using the sample reflector Pathway application are as follows:

1. Getting the sample application files from sample_services
2. Creating DDL dictionary files
3. Creating SDL files
4. Creating Web service files using SOAPAdminCL tools
5. Compiling reflector.c
6. Configuring and starting the reflector application
7. Accessing the Web service

NOTE: The steps mentioned can be used to deploy any server class other than reflector.

Getting the sample application files from sample_services

To copy the files from sample_services, complete the following steps:

1. Navigate to the `<Reflector Source Directory>` folder. This is a directory of your choice where all reflector service related files reside.
2. Copy the `reflector` sample from `<NonStop SOAP 4 Installation Directory>`/sample_services/reflector.

```
OSS>cp –r <NonStop SOAP 4 Installation Directory>/sample_services/reflector .
```

Creating DDL dictionary files

A DDL dictionary is required to expose a NonStop server class or Pathway application as a Web service. The DDL data definitions are used to create the data fields in the WSDL file. To create the DDL dictionaries, compile the DDL files that include the request and response structure for the Pathway application.

The `<Reflector Source Directory>`/src directory includes the `reflectorddl` file for the reflector application. Navigate to this folder and edit the following parameter in the `reflectorddl` file:

```
$<VOLUME>.<SUBVOLUME>
```

is a Guardian location where you want the dictionary files for the reflector service to be created. for example, `$DATA.REFL`. You must have read and write permissions for this directory. The specified dictionary location should NOT have any existing dictionary files.

Create the DDL dictionaries using the following command:

```
OSS > gtacl –p ddl < reflectorddl
```

NOTE: Make sure that the command returns no errors. For information on developing DDL definitions, see the *Data Definition Language (DDL) Reference Manual*.

For example, if you specified `$DATA.REFL` as the dictionary location, successful compilation of `reflectorddl` generates the following output:

```
OSS> gtacl -p fup info \$DATA.REFL.*
         CODE    EOF     LAST         MODIFIED   OWNER    RWEP     PExt    SExt
DICTALT 201A   12288   07NOV2008       18:40   255,255   NUNU       4      32
DICTCDF 207A       0   07NOV2008       18:40   255,255   NUNU       4      32
DICTDDF 200       30   07NOV2008       18:40   255,255   NUNU      14      14
DICTKDF 206A       0   07NOV2008       18:40   255,255   NUNU       4      32
DICTMAP 209A       0   07NOV2008       18:40   255,255   NUNU       4      32
```

```
DICTOBL 204A  12288  07NOV2008      18:40  255,255  NUNU      4      32
DICTODF 202A  12288  07NOV2008      18:40  255,255  NUNU      4      32
DICTOTF 203A  12288  07NOV2008      18:40  255,255  NUNU      4      32
DICTOUF 208A      0  07NOV2008      18:40  255,255  NUNU      4      32
DICTOUK 208A      0  07NOV2008      18:40  255,255  NUNU      4      32
DICTRDF 205A      0  07NOV2008      18:40  255,255  NUNU      4      32
DICTTKN 209A      0  07NOV2008      18:40  255,255  NUNU      4      32
DICTTYP 209A      0  07NOV2008      18:40  255,255  NUNU      4      32
DICTVER 209A      0  07NOV2008      18:40  255,255  NUNU      4      32
```

Creating SDL files

For any new service creation the sdl file has to be created manually or using SOAP administrators utility. For convenience, in this example the `reflectorsdl.xml` is shipped along with the product and is located in the following directory.
Open the `reflectorsdl.xml` file which is located *<Reflector Source Directory>*`/src` and edit the following parameters:

- PathwayEnvironment

  Update the `PathmonName` attribute of the `PathwayEnvironment` element to reflect the PATHMON process name specified in the `pathConf` file. The `pathConf` file is located at *<Reflector Source Directory>*`/src`)

  ```
  <PathwayEnvironment PathmonName="$REFL">
  ```

- ServerAddress

  Update the `ServerAddress` attribute with the IP and port number of the iTP WebServer where NonStop SOAP 4 has been deployed. For example:

  ```
  ServerAddress ="http://<ip address>:<port>"
  ```

- Url

  Update the `Url` attribute of the SDL element. This attribute must have a valid relative path to the `ServerAddress` attribute. For example, `Url ="/mytest"` where, `mytest` is the URL pattern entered while running the `deploy.sh` deployment script to create the NonStop SOAP 4 deployment directory.

- DDLDictionary Location

  Update the `DDLDictionaryLocation` attribute of the service tag with the dictionary file location you have mentioned in the `reflectorddl` file see (page 78)

  ```
  DDLDictionaryLocation="$DATA.REFL"
  ```

  NOTE: The `DDLDictionaryLocation` must be the same as the dictionary location specified in the `reflectorddl` file.

Creating Web Service files using SOAPAdminCL tool

To invoke the `SoapAdminCL` tool from the *<NonStop SOAP 4 Installation Directory>*`/tools directory`, complete the following steps:

1. Add the directory containing the `SoapAdminCL` executable image to the OSS PATH variable, using the following command:

   ```
   OSS> export PATH=<NonStop SOAP 4 Installation Directory>/tools:$PATH
   ```

   For example, if NonStop SOAP 4 is installed at the default location

   ```
   (/usr/tandem/nssoap/t0865h01),
   ```

   the OSS command is:

   ```
   OSS> export PATH=/usr/tandem/nssoap/t0865h01/tools:$PATH
   ```

2. Run `SoapAdminCL` to generate the WSDL file, HTML clients, XML schema files, SOAP request and response XML files, and the `services.xml` file for the reflector service. Navigate to *<Reflector Source Directory>*/src and enter the following command:

   ```
   OSS> SoapAdminCL -o <NonStop SOAP 4 Deployment Directory> -i reflectorsdl.xml
   ```

   The `SoapAdminCL` tool generates the following files in the *<NonStop SOAP 4 Deployment Directory>* directory:

File Name	Description
SoapPW_reflector.wsdl	This file contains the service definition details and the XML schema definitions for the reflector service. NonStop SOAP 4 uses this file at runtime.
SoapPW_REFLECTOR.html	This file is the HTML client used to access the reflector service.
services.xml	This file contains the service configuration details used by the NonStop SOAP 4 server at runtime.
SoapPW_REFLECTOR.xsd	This file contains the XML schema for the SOAP request structure.
SoapPW_REFLECTORRESPONSE0.xsd	This file contains the XML schema for the SOAP response structure.
SoapPW_REFLECTOR.xml	This file is a sample SOAP request XML file.
SoapPW_REFLECTORRESPONSE0.xml	This file is a sample SOAP response XML file.

3. Verify that the SOAP server is running.

   ```
   OSS> gtacl -c "pathcom \<PATHMON name>;status server *"
   ```

   where,

   ```
   <PATHMON_name> is the name of the PATHMON on which iTP WebServer is running.
   ```

   This command returns the following statistics:

   ```
   SERVER          #RUNNING     ERROR     INFO
   AXIS2CGI               1
   GENERIC-CGI            1
   HTTPD                  1
   ```

   NOTE:

   - `AXIS2CGI` represents the ServerClass name.
   - A value of 1 for `GENERIC-CGI` or `HTTPD` indicates that iTP WebServer is running.
   - A value of 0 for `GENERIC-CGI` or `HTTPD` indicates that iTP WebServer is not running and an error number is displayed in the respective ERROR column. For more information about the error and its resolution, see the *NonStop TS/MP Pathsend and Server Programming Manual.*
   - Use the **-f** option with the `SoapAdminCL` command to overwrite the existing files. If you do not specify the **-f** option, SoapAdminCL will generate new files based on the SDL attributes and report a failure for the existing files which are not overwritten.

     If it is not running, start the SOAP server, go to the *<iTP WebServer Deployment Directory>*`/conf` directory and run the restart script

     ```
     OSS> cd <iTP _WebServer_Deployment Directory>/conf
     OSS> ./restart
     ```

     This `command` restarts the iTP WebServer and NonStop SOAP 4 and the `reflector` service is deployed.
   - NonStop SOAP 4 also provides a NonStop SOAP 4 administrator tool that provides a GUI interface to develop and deploy NonStop SOAP 4 SDL files. For more information about the NonStop SOAP 4 administrator tool, see "NonStop SOAP Tools" (page 194)

Compiling reflector.c

To compile reflector.c, complete the following step:

- Go to the *<Reflector Service Directory>*`/src` directory. This directory contains the `reflector.c` source file. Compile the `reflector.c` file using the command:

  ```
  OSS> cd <Reflector Service Directory>/src
  OSS> c89 -I . –Wextensions reflector.c –o reflector
  ```

  The reflector server object is created.

  NOTE: This step is optional if you already have the binary available on your system. This is applicable only for the reflector application.

Configuring and starting the reflector application

To configure and start the reflector application for a particular PATHMON process, use the `pathConf` script file.

To configure and start the reflector application, complete the following steps:

1. Go to the <*Reflector Service Directory*>/conf directory to access the pathConf script file. Provide execute permissions to the pathConf file, using the following command:

   ```
   OSS> cd <Reflector Service Directory>/conf
   OSS> chmod u+x pathConf
   ```

2. Update the PATHWAY_SUBVOL variable in the pathConf file, to point to a Guardian location where the PATHMON process configuration files will be created. For example, PATHWAY_SUBVOL=\$DATA.SOAP
3. Run the pathConf script file to start the Pathway environment and reflector application. The pathConf file uses the PATHMON name in which the reflector server class must be started as a command-line argument.

   For example,

   ```
   OSS> ./pathConf \$REFL
   ```

   where,

   $REFL is the name of the PATHMON process in which the reflector service will run.
4. Verify that the reflector application is running under the PATHMON named $REFL. Run the pathcom command to invoke the PATHCOM interface.

   ```
   OSS>gtacl -c "pathcom \$REFL; status server *"
   ```

   This command displays the following statistics:

   ```
   SERVER #RUNNING ERROR INFO
   REFLECTOR 1
   ```

   NOTE: 1 indicates that one instance of the reflector application is running under $REFL.

   These steps need to be carried out only if you have created a new PATHMON process.

   The reflector application is deployed under the PATHMON named $REFL.

Accessing the Web Service deployment

You can access the Web service, WSDL file, and HTML clients after the service is deployed on NonStop SOAP 4.

- You can access the WSDL file for your service using the following Web address:

  ```
  http://<ip address>:<port>/<url_pattern>/services/<service_name>? wsdl
  ```

  where,

  <ip address>:<port>

  is the IP address and port of iTP WebServer where NonStop SOAP 4 is deployed.

  <url_pattern>

  is the unique string value you entered during installation. For more information on url_pattern, see "Installing NonStop SOAP" (page 36)

  For example, the Web address to access the WSDL file will be:

  ```
  http://127.0.0.1:8088/mytest/services/reflector?wsdl
  ```

- You can access the HTML clients for your service using the following Web address:

  ```
  http://<ip address>:<port>/<url_pattern>/client/<service_name>/SOAP_PW<operation_name>.html
  ```

  where,

  <ip address>:<port>

  is the IP address and port of iTP WebServer where NonStop SOAP 4 is deployed.

  <url_pattern>

  is the unique string value you entered during installation. For more information on url_pattern, see "Installing NonStop SOAP" (page 36)

client is the directory in <NonStop SOAP 4 Deployment Directory> where the HTML clients for the services are located.

To verify the deployed service, complete the following steps:

1. Access the SoapPW_REFLECTOR.html using the following Web address:

   http://<*ip address*>:<*port*>/<url_pattern>/client/reflector/SoapPW_REFLECTOR.html

2. Enter the string Welcome in the text box in the SoapPW_REFLECTOR.html file and click **Submit**.

The string you entered as the SOAP response, in XML format, appears. For example, the following is a sample SOAP Response XML file:

```
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
<soapenv:Header/>
<soapenv:Body>
<tns1:REFLECTORResponse0 xmlns:tns1="urn:cpq_tns_REFLECTOR"
xmlns="urn:cpq_tns_REFLECTOR" xmnls:xsd="https://www.w3.org/2001/XMLSchema">
<outstr>Welcome</outstr>
</tns1:REFLECTORResponse0>
</soapenv:Body>
<soapenv:Envelope>
```

Deploying a NonStop process as a Web Service

Deploying NonStop process as a Web service involves the following steps:

1. Compile the reflector application. Please refer "Compiling reflector.c" (page 81) for compilation steps.
2. "Creating DDL dictionary files" (page 78)
3. Creating SDL files

   The procedure to deploy a service running as a NonStop process in NonStop SOAP 4 is similar to the procedure used to deploy a Pathway application. However, the SDL needs to specify the attributes of the NonStop process being deployed under NonStop SOAP. In the SDL, this is accomplished using the Process element rather than the Pathway element.

   Create an SDL file reflectorsdl_process.xml file which contains the following:

```
<sdl Url="/axis2c" ServerAddress ="http://www.nonstopsoap.com">
<Process>
  <ProcessEnvironment Name="$<PROCESS>">
   <ProcessDetails Name="Reflector"
     DDLDictionaryLocation="$<Volume>.<SubVolume>"
     SrvrEncoding="UTF-8"
     language="C"
     stringTermination="NullTerminated">
   <Operation Name="REFLECTOR"
    TMFTransactionSupport="Never"
    AbortTransactionOnFault="yes"
    NameSpaceQualified="yes"
    SoapMessageType="document"
    ProcessSoapDDLComments="no"
    SoapDDLAttribute="no"
    UseDDLDefaultValue="no">
    <OperationDescription>Reflector</OperationDescription>
    <RequestInfo>
      <DDLDefinitionName>INPUT</DDLDefinitionName>
    </RequestInfo>
    <ResponseInfo>
      <DDLDefinitionName>OUTPUT</DDLDefinitionName>
    <ResponseSelection defaultResp="yes"/>
    </ResponseInfo>
    </Operation>
   </ProcessDetails>
  </ProcessEnvironment>
```

```
</Process>
</sdl>
```

Update the following parameters in the reflectorsdl_process.xml file:

`Url`

Update the Url attribute of the SDL element. This attribute must have a valid relative path to the ServerAddress attribute. For example:

```
Url= "/mytest"
```

where,

mytest is the URL pattern entered while running the `deploy.sh` deployment script to create the NonStop SOAP 4 deployment directory.

`ServerAddress`

Update the ServerAddress attribute with the IP and port number of the iTP WebServer where the NonStop SOAP 4 has been deployed. For example:

```
ServerAddress ="http://<ip address>:<port>"
```

`ProcessEnvironment`

Update the Process name of the ProcessEnvironment element to reflect the process name specified in the Step 2. For example:

```
<ProcessEnvironment Name="$REF">
```

`DDLDictionaryLocation`

Update the DDLDictionaryLocation attribute of the service tag with the dictionary file location you have mentioned in the `reflectorddl` file (see "Creating DDL dictionary files" (page 78)).

```
DDLDictionaryLocation="$DATA.REFL"
```

NOTE: The DDLDictionaryLocation must be the same as the dictionary location specified in the `reflectorddl` file.

For more information on developing SDL files for NonStop process-based services, see "NonStop SOAP 4 Service Description Language" (page 153).

NOTE: You can generate the SDL file using the NonStop SOAP 4 Administration Utility. For information about generating the SDL file using the NonStop SOAP 4 Administration Utility, see "NonStop SOAP Tools" (page 194).

4. "Creating Web Service files using SOAPAdminCL tool" (page 79)
5. Starting the `reflector` application. Go to the *<Reflector Service Directory>*`/src` directory where the `reflector` object is created. Run the following command:

   ```
   run -name=/G/<PROCESS> ./<reflector_object>
   ```

   where,

   `<PROCESS>`

   is the NonStop process you want to create.

   `<reflector_object>`

   is the `reflector` object you have created in previous step.
6. "Accessing the Web Service deployment" (page 82)

5 NonStop SOAP 4 Service APIs

This chapter describes the NonStop SOAP 4 application programming interfaces (APIs) you can use to develop a DLL-based Web service deployed in NonStop SOAP 4.

This chapter includes the following topics:

- "Basic NonStop SOAP 4 Service APIs" (page 85)
- "Developing NonStop SOAP 4 Services using Service APIs" (page 95)

Basic NonStop SOAP 4 Service APIs

NonStop SOAP 4 service APIs provide a stack of function calls to develop Web services that you can deploy in NonStop SOAP 4.

The following service APIs are described in this section:

- "APIs to Implement the Service Interface with NonStop SOAP 4" (page 85)
- "APIs to Implement the Service Skeleton Structure Interface" (page 87)
- "APIs to Extract the Input Parameters and Return the Response" (page 89)
- "APIs for Logging" (page 95)

APIs to Implement the Service Interface with NonStop SOAP 4

Services developed using NonStop SOAP 4 service APIs must implement the following functions:

- "The `axis2_get_instance()` Function" (page 85)
- "The `axis2_remove_instance()` Function" (page 86)

The `axis2_get_instance()` Function

NonStop SOAP 4 calls the `axis2_get_instance()` function when it loads the service for the first time. It does not call this function in subsequent invocations of the service. The `axis2_get_instance()` function creates an instance of the service skeleton structure.

The following tasks are performed by the `axis2_get_instance()` function implemented in your service:

1. "Creating the service skeleton structure" (page 86)
2. "Setting the service skeleton operations structure" (page 86)
3. "Returning the service skeleton structure to the service" (page 86)

Synopsis:

```
AXIS2_EXPORT int axis2_get_instance(
    struct axis2_svc_skeleton **inst,
    const axutil_env_t * env)
```

Parameters:

`inst`

is an output parameter that points to the service skeleton structure.

`env`

is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return Values:

- `AXIS2_SUCCESS` on success.
- `AXIS2_FAILURE` on failure.

Creating the service skeleton structure

To create the service skeleton, you must:

1. Define the service skeleton structure of type `axis2_svc_skeleton_t`.
2. Allocate memory for the service skeleton structure.

For example:

```
axis2_svc_skeleton_t *soap4service_svc_skeleton = NULL;
   soap4service_svc_skeleton = AXIS2_MALLOC
  (env->allocator, sizeof(axis2_svc_skeleton_t));
```

Setting the service skeleton operations structure

After you create the service skeleton structure, the callback functions that implement the service initialization, business logic, and fault handling must be set in the operations structure of the service skeleton structure of type `axis2_svc_skeleton_ops_t`.

Synopsis:

```
static const axis2_svc_skeleton_ops_t
       soap4service_svc_skeleton_ops_var =
{
 soap4service_init
 soap4service_invoke
 soap4service_on_fault
 soap4service_free
}
```

where,

`soap4service_init`
: is the function that implements the service initialization logic for the service.

`soap4service_invoke`
: is the function that implements the business logic for the service.

`soap4service_on_fault`
: is the function that implements the fault handling for the service.

`soap4service_free`
: is the function that implements the service cleanup activities.

NOTE: The `soap4service_svc_skeleton_ops_var` structure must be a global variable in the service implementation.

After you create the service skeleton operations structure, assign it to the `ops` field in the created service skeleton structure.

For example:

```
soap4service_svc_skeleton->ops = &soap4service_svc_skeleton_ops_var;
```

Returning the service skeleton structure to the service

After you create and set the service skeleton structure, return the service skeleton structure to NonStop SOAP 4.

For example:

```
return soap4service_svc_skeleton;
```

The `axis2_remove_instance()` Function

The `axis2_remove_instance()` function removes the service skeleton structure from the memory. NonStop SOAP 4 calls the `axis2_remove_instance()` function when the NonStop SOAP 4 server exits.

Synopsis:

```
AXIS2_EXPORT int  axis2_remove_instance(
    axis2_svc_skeleton_t * inst,
    const axutil_env_t * env)
```

Parameters:

`inst`
is an input parameter and is an instance of the service skeleton structure for the service.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return Values:

- `AXIS2_SUCCESS` on success.
- `AXIS2_FAILURE` on failure.

APIs to Implement the Service Skeleton Structure Interface

After you create the service skeleton structure, implement the following functions defined in the service skeleton operations structure:

- "The init Function" (page 87)
- "The invoke Function" (page 87)
- "The fault Function" (page 88)
- "The free Function" (page 88)

The init Function

The init function, set in the service skeleton structure, implements the service initialization logic for the service and is called by NonStop SOAP 4 while processing a request.

Synopsis:

```
int AXIS2_CALL soap4service_init(axis2_svc_skeleton_t *svc_skeleton,
                                 const axutil_env_t *env);
```

Parameters:

`svc_skeleton`
is an input parameter that points to the service skeleton structure created in the `axis2_get_instance()` function.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return values:

- `AXIS2_SUCCESS` on success.
- `AXIS2_FAILURE` on failure.

The invoke Function

The invoke function, set in the service skeleton structure, implements the business logic for the service and is called by NonStop SOAP 4 while processing a request.

Synopsis:

```
axiom_node_t *AXIS2_CALL soap4service_invoke(axis2_svc_skeleton_t *svc_skeleton,
                                             const axutil_env_t *env,
```

```
                                                    axiom_node_t* node,
                                                    axis2_msg_ctx_t *msg_ctx);
```

Parameters:

`svc_skeleton`
is an input parameter that points to the service skeleton structure created in the `axis2_get_instance()` function.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`node`
is an input parameter and is an AXIOM node that contains the SOAP request from the client.

`msg_ctx`
is an input parameter and is the message context structure of the service.

NOTE:

- The AXIOM node is the principle data structure exposed by the NonStop SOAP 4 server that defines the request and response data elements.
- Message context is the structure that contains the request-specific parameters for a SOAP request.

Return values:

Pointer to the AXIOM node that points to the response received from the service.

The fault Function

The fault function, set in the service skeleton structure, implements the fault handling logic of the service and is called by NonStop SOAP 4 if a fault occurs while processing a request in the service.

Synopsis:

```
axiom_node_t *AXIS2_CALL soap4service_on_fault(axis2_svc_skeleton_t *srv_skeleton,
                                               const axutil_env_t *env,
                                               axiom_node_t* node);
```

Parameters:

`srv_skeleton`
is an input parameter that points to the service skeleton structure created in the `axis2_get_instance()` function.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`node`
is an input parameter and is the address of the AXIOM node that contains the SOAP request from the client.

Return values:

Pointer to the AXIOM node that points to the fault received from the service.

The free Function

The free function, set in the service skeleton structure, is called to free the memory after request processing.

Synopsis:

```
int AXIS2_CALL soap4service_free(axis2_svc_skeleton_t *srv_skeleton,
                                 const axutil_env_t *env);
```

Parameters:

`srv_skeleton`

is an input parameter which points to the service skeleton structure created in the `axis2_get_instance()` function.

`env`

is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return values:

- `AXIS2_SUCCESS` on success.
- `AXIS2_FAILURE` on failure.

APIs to Extract the Input Parameters and Return the Response

NonStop SOAP 4 calls the `invoke()` function from the service skeleton operation structure to pass the request to the Web service in the form of an AXIOM node. After processing the request, the Web service builds the response in the AXIOM node and returns the response to NonStop SOAP 4.

The AXIOM APIs, from the NonStop SOAP 4 service APIs, that enable you to read and create AXIOM nodes are described in the following sections:

- "Reading an AXIOM node using NonStop SOAP 4 APIs" (page 89)
- "Creating an AXIOM node using NonStop SOAP 4 APIs" (page 91)

Reading an AXIOM node using NonStop SOAP 4 APIs

NonStop SOAP 4 provides the following functions to read an AXIOM node to process a request:

- "The `axiom_element_get_qname()` Function" (page 89)
- "The `axiom_node_get_data_element()` Function" (page 90)
- "The `axiom_attribute_get_qname()` Function" (page 90)
- "The `axiom_element_get_attribute()` Function" (page 90)

The `axiom_element_get_qname()` Function

The `axiom_element_get_qname()` function returns the qualified name of a given AXIOM element.

Synopsis:

```
AXIS2_EXTERN axutil_qname_t* axiom_element_get_qname
                              (axiom_element_t* om_element,
                               const axutil_env_t *  env,
                               axiom_node_t* om_node)
```

Parameters:

`om_element`

is an input parameter and is a pointer to the element whose `qname` must be retrieved.

`env`

is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`om_node`

is an input parameter and is a pointer to the `axiom_node_t` node that contains `om_element`.

Return Values:

Pointer to the `qname` of the element. If an error occurs, it returns `NULL`.

The axiom_node_get_data_element() Function

The axiom_node_get_data_element() function retrieves the AXIOM element structure from the AXIOM node.

Synopsis:

```
AXIS2_EXTERN void axiom_node_get_data_element
                            ( axiom_node_t* om_node,
                              const axutil_env_t * env)
```

Parameters:

om_node
: is an input parameter and is a pointer to the node for which a data element is required.

env
: is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

Return Values:

Pointer to the structure included in the AXIOM node. If an error occurs, it returns NULL.

NOTE: You must cast the returned void pointer into an axiom_element_t* data type. This will allow you to use the data element present in the AXIOM node.

The axiom_attribute_get_qname() Function

The axiom_attribute_get_qname() function retrieves the qualified name of an AXIOM attribute.

Synopsis:

```
AXIS2_EXTERN axutil_qname_t* axiom_attribute_get_qname
                                  (struct axiom_attribute_t* om_attribute,
                                    const axutil_env_t * env)
```

Parameters:

om_attribute
: is an input parameter and is a pointer to the attribute structure for which qname must be returned.

env
: is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

Return Values:

Pointer to the qname for a given attribute. If an error occurs, it returns NULL.

The axiom_element_get_attribute() Function

The axiom_element_get_attribute() function finds the attribute using the given qualified name.

Synopsis:

```
AXIS2_EXTERN axiom_attribute_t* axiom_element_get_attribute
                                ( axiom_element_t *  om_element,
                                  const axutil_env_t * env,
                                  axutil_qname_t *  qname )
```

Parameters:

element
: is an input parameter and is a pointer to the element whose attribute must be found.

env

is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

Return Values:

Pointer to the attribute with the given qname. If an error occurs, it returns NULL and sets the error code in the environment error structure.

Creating an AXIOM node using NonStop SOAP 4 APIs

NonStop SOAP 4 provides APIs to create the AXIOM node using one of the following two approaches:

- "Creating an AXIOM node using AXIOM Node Create APIs" (page 91)
- "Creating an AXIOM node from an XML using AXIOM Document APIs" (page 93)

Creating an AXIOM node using AXIOM Node Create APIs

NonStop SOAP 4 provides the following functions to create the response AXIOM node:

- "The axiom_node_create() Function" (page 91)
- "The axiom_node_add_child() Function" (page 91)
- "The axiom_element_create() Function" (page 92)
- "The axiom_element_add_attribute() Function" (page 92)
- "The axiom_attribute_create() Function" (page 93)

The axiom_node_create() Function

The axiom_node_create() function creates an AXIOM node structure.

Synopsis:

```
AXIS2_EXTERN axiom_node_t* axiom_node_create
                           (const axutil_env_t *  env)
```

Parameters:

env

is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

Return Values:

Pointer to the newly created node structure. If an error occurs, it returns NULL.

The axiom_node_add_child() Function

The axiom_node_add_child() function adds an AXIOM node as a child to a parent AXIOM node.

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_node_add_child
                           ( axiom_node_t* om_node,
                             const axutil_env_t *  env,
                             axiom_node_t* child
                           )
```

Parameters:

om_node

is an input parameter and points to the parent node. It cannot have a NULL value.

env
: is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

child
: is an input parameter and is a pointer to the child node of type axiom_node_t. It cannot have a NULL value.

Return Values:

- AXIS2_SUCCESS on success.
- AXIS2_FAILURE on failure.

The axiom_element_create() Function

The axiom_element_create() function creates an AXIOM element using the name specified.

Synopsis:

```
AXIS2_EXTERN axiom_element_t* axiom_element_create
                              ( const axutil_env_t *  env,
                                axiom_node_t *  parent,
                                const axis2_char_t *  localname,
                                axiom_namespace_t *  ns,
                                axiom_node_t **  node )
```

Parameters:

env
: is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

parent
: is an input parameter and is the pointer to the parent of the element node to be created. It cannot have a NULL value.

localname
: is an input parameter and is the pointer to the local name of the element. It cannot have a NULL value.

ns
: is an input parameter and is the pointer to the namespace, if any, of the attribute. It can have a NULL value.

node
: is an output parameter and is a pointer to the parameter that returns the node corresponding to the element created.

Return Values:

Pointer to the newly created element structure. If an error occurs, it returns NULL.

The axiom_element_add_attribute() Function

The axiom_element_add_attribute() function adds an attribute to the current element.

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_element_add_attribute
                           ( axiom_element_t *  om_element,
                             const axutil_env_t *  env,
                             axiom_attribute_t *  attribute,
                             axiom_node_t *  node )
```

Parameters:

`om_element`
is an input parameter and is a pointer to the element to which the attribute is added. It cannot have a `NULL` value.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`attribute`
is an input parameter and is a pointer to the attribute to be added to the element. It cannot have a `NULL` value.

`node`
is an input parameter and is a pointer to the `axiom_node_t` node that contains `om_element`. It cannot have a `NULL` value.

Return Values:

- `AXIS2_SUCCESS` on success.
- `AXIS2_FAILURE` on failure.

The `axiom_attribute_create()` Function

The `axiom_attribute_create()` function creates an AXIOM attribute structure.

Synopsis:

```
AXIS2_EXTERN axiom_attribute_t* axiom_attribute_create
                                ( const axutil_env_t *  env,
                                  const axis2_char_t *  localname,
                                  const axis2_char_t *  value,
                                  axiom_namespace_t *  ns )
```

Parameters:

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`localname`
is an input parameter and is a pointer to the local name of the attribute. It cannot have a `NULL` value.

`value`
is an input parameter and is a pointer to the normalized attribute value. It cannot have a `NULL` value.

`ns`
is an input parameter and is a pointer to the namespace, if any, of the attribute. It is an optional parameter and can have a `NULL` value.

Return Values:

Pointer to the newly created attribute structure. If an error occurs, it returns `NULL`.

Creating an AXIOM node from an XML using `AXIOM Document` APIs

If you create the service response from an XML format, use the `AXIOM document` APIs to create the AXIOM node from the response XML.

NOTE: You need not use the `AXIOM document` APIs if the service builds the response using AXIOM APIs. The `AXIOM document` APIs interpret the hierarchy and relationship between the elements in an XML message and generates the AXIOM node that mirrors the relationships in the XML message. Unlike the `AXIOM document` APIs, when you use the AXIOM API to build the AXIOM node, you need to set the relationships between the elements.

NonStop SOAP 4 uses the StAX parser to convert the XML payload into an AXIOM node. The parser traverses the XML payload and converts the payload into the corresponding AXIOM node. The StAX parser consumes the XML payload and produces an AXIOM document structure. The AXIOM document structure is queried to retrieve the AXIOM nodes.

The NonStop SOAP 4 APIs provide the following functions to process the AXIOM document structure:

- "The `axiom_document_create()` Function" (page 94)
- "The `axiom_document_build_all()` Function" (page 94)
- "The `axiom_document_free()` Function" (page 95)

The `axiom_document_create()` Function

The `axiom_document_create()` function creates an `axiom_document_t` structure. This structure holds the AXIOM node representation of the input XML payload.

Synopsis:

```
AXIS2_EXTERN axiom_document_t* axiom_document_create
                               ( const axutil_env_t *  env,
                                 axiom_node_t *  root,
                                 struct axiom_stax_builder *  builder )
```

Parameters:

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`root`
is an input parameter and is a pointer to the root node of the XML document.

`builder`
is an input parameter and is a pointer to the StAX parser.

Return Value:

Pointer to the newly created AXIOM document. If an error occurs, it returns `NULL`.

The `axiom_document_build_all()` Function

The `axiom_document_build_all()` function builds the XML input stream from the current position of the StAX parser till the root element is complete.

Synopsis:

```
AXIS2_EXTERN axiom_node_t* axiom_document_build_all
                             ( struct axiom_document * document,
                               const axutil_env_t * env )
```

Parameters:

`document`
is an input parameter and is a pointer to the `axiom_document_t` structure to be built.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return Value:

Pointer to the AXIOM node built from the document. If an error occurs, it returns `NULL`.

The `axiom_document_free()` Function

The `axiom_document_free()` function is used to free the document structure from the memory.

Synopsis:

```
AXIS2_EXTERN void axiom_document_free
                                ( struct axiom_document_t* document,
                                  const axutil_env_t *  env)
```

Parameters:

`document`

is an input parameter and is a pointer to the document structure that becomes free from the memory.

`env`

is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return Value:

Void.

APIs for Logging

The NonStop SOAP 4 service API includes the logging APIs to create and log messages.

Logging APIs enable you to create the log file structure, which includes the location and the name of the log file.

Creating the log file structure

The `axutil_log_create()` function is called to create the log file structure.

Synopsis:

```
 AXIS2_EXTERN axutil_log_t *  axutil_log_create
                              (axutil_allocator_t *allocator,
                               axutil_log_ops_t *ops,
                               const axis2_char_t *stream_name)
```

Parameters:

`allocator`

is an input parameter and is a pointer to the `axutil_allocator` structure.

`ops`

is an input parameter and is a pointer to the `options` structure for logging.

`stream_name`

is an input parameter and is a pointer to the file name and location where the log must be created.

Return Values:

Pointer to the newly created log structure. If an error occurs, it returns `NULL`.

Developing NonStop SOAP 4 Services using Service APIs

This section describes the procedure to develop and deploy a NonStop SOAP 4 service, using the `math` service as an example.

The `math` service can implement the following mathematical operations:

- Addition of two integers
- Subtraction of two integers
- Multiplication of two integers
- Division of two integers

NOTE: The source file for the `math` service is available in the *<NonStop SOAP 4 Installation Directory>*`/sample_services/math` directory.

To develop NonStop SOAP 4 services, use one of the following:

- The service API stack

  The service API stack enables you to develop NonStop SOAP 4 services manually. This approach provides higher flexibility to customize the services based on your business requirement.

- The `WSDL2C` tool

  The `WSDL2C` tool is a Java-based tool that is distributed with NonStop SOAP 4. You can use this tool to generate the client and service skeleton files in the C programming language, based on the WSDL file for the service. Your application business logic must be integrated with the skeleton files.

  For most applications, the `WSDL2C` tool is an optimal way to generate the service skeleton files because you do not need to write the skeleton files. For more information on the `WSDL2C` tool, see "NonStop SOAP Tools" (page 194).

Developing and deploying NonStop SOAP 4 services, involves the following tasks:

1. "Defining the XML Request and Response Payload" (page 96)
2. "Creating the SDL File for the Service" (page 98)
3. "Generating the WSDL File and the `services.xml` File" (page 99)
4. "Generating the Service Skeleton Files" (page 100)
5. "Implementing the Business Logic in the Service Skeleton Files" (page 102)
6. "Compiling the Service Code and Deploying the Service" (page 102)
7. "Testing the Service" (page 103)

Defining the XML Request and Response Payload

To develop a NonStop service, you must specify the XML request and response payload in the DDL file of the service. The XML payload helps to specify the parameters and their relationship in the XML tree structure.

To define the XML request and response payload and to compile the DDL file, complete the following steps:

1. Define the request and response XML payload for the service.

   For example, for the `math` service:

   - The request XML payload for the `add` operation is:

     ```
     <Body>
           <add>
                <param1>1< /param1>
                <param2>2< /param2>
           </add>
     </Body>
     ```

   - The response XML payload for the `add` operation is:

     ```
     < Body>
           < addResponse>
                <result>3< /result>
           < /addResponse>
     </Body>
     ```

2. Copy the `mathddl` file from the *<NonStop SOAP 4 Installation Directory>*`/sample_services/math/service` to an OSS location.

   This OSS location will be referred as *<Math Source Directory>*.

   NOTE: The NonStop SOAP 4 distribution includes the `mathddl` DDL file for all the operations of the `math` service.

3. Edit the DDL file copied to the *<Math Source Directory>* to modify the `?DICT` location. Set the `?DICT` location to a valid subvolume location on your system.

   NOTE: You can create the DDL file for the `math` service using a text editor in the *<Math Source Directory>*.

   The DDL file must contain the following entries:

   ```
   ?dict $<volume>.<subvolume>
   ?comments
    DEFINITION PARAMETER.
         10 PARAM1 TYPE binary 32.
         10 PARAM2 TYPE binary 32.
    END

    DEFINITION RESULT.
         10 Response TYPE binary 32.
    END
   ```

   where,

   `$<volume>.<subvolume>`
   : is a valid Guardian location where you want to create the dictionary files for the `math` service.

   Ensure that you have read and write permissions for this directory.

4. To compile the DDL file, use the following command:

   `oss> gtacl -p ddl < "`*`service DDL file`*`"`

   For example:

   `OSS> gtacl -p ddl < mathddl`

   Ensure that the DDL command does not return any errors. For information about DDL errors, see the *Data Definition Language (DDL) Reference Manual*.

On successfully completing the DDL dictionary compilation, the dictionary files are generated in the Guardian subvolume specified in the service DDL file.

For example, if you specified `$DATA.MATH` as the dictionary location in the `mathddl` file, successful compilation of `mathddl` generates the following files in the `$DATA.MATH` location:

```
oss> exit

GUARDIAN> volume $DATA.MATH
$DATA.MATH> fileinfo *
           CODE             EOF   LAST MODIFIED  OWNER  RWEP    PExt    SExt
DICTALT     201A           12288 07NOV2008 18:40 255,255 NUNU       4      32
DICTCDF     207A               0 07NOV2008 18:40 255,255 NUNU       4      32
DICTDDF     200               30 07NOV2008 18:40 255,255 NUNU      14      14
DICTKDF     206A               0 07NOV2008 18:40 255,255 NUNU       4      32
DICTMAP     209A               0 07NOV2008 18:40 255,255 NUNU       4      32
DICTOBL     204A           12288 07NOV2008 18:40 255,255 NUNU       4      32
DICTODF     202A           12288 07NOV2008 18:40 255,255 NUNU       4      32
DICTOTF     203A           12288 07NOV2008 18:40 255,255 NUNU       4      32
DICTOUF     208A               0 07NOV2008 18:40 255,255 NUNU       4      32
DICTOUK     208A               0 07NOV2008 18:40 255,255 NUNU       4      32
DICTRDF     205A               0 07NOV2008 18:40 255,255 NUNU       4      32
DICTTKN     209A               0 07NOV2008 18:40 255,255 NUNU       4      32
DICTTYP     209A               0 07NOV2008 18:40 255,255 NUNU       4      32
DICTVER     209A               0 07NOV2008 18:40 255,255 NUNU       4      32

$DATA.MATH> osh

OSS>
```

Creating the SDL File for the Service

After you define the XML request and response payload, you must create the SDL file for the service because the `SoapAdminCL` tool uses the SDL file as an input to generate the WSDL file. NonStop SOAP 4 uses the generated WSDL file to process requests.

You can create an SDL file manually or you can use the NonStop SOAP 4 Administration Utility to create the SDL file. For the `math` sample, the completed SDL file, `mathsdl.xml` is available in the *<NonStop SOAP 4 Installation Directory>*`/sample_services/math/service` directory.

After the SDL file is available, complete the following steps:

1. Create the SDL file for the service using one of the following approaches:
   - Manually write the SDL file for the service. Create the directory structure `/math/src` under the *<NonStop SOAP 4 Deployment Directory>*`/services` using the following OSS commands:

     ```
     OSS>mkdir <NonStop SOAP 4 Deployment Directory>/services/math
     OSS>mkdir  <NonStop SOAP 4 Deployment Directory>/services/math/src
     ```

   - Use the NonStop SOAP 4 Administration Utility.

     NOTE: If you have created the SDL file using the NonStop SOAP 4 Administration Utility, do not copy the `mathsdl.xml` file. The NonStop SOAP 4 Administration Utility replaces the SDL file in the appropriate location.

     For information on generating the SDL file using the NonStop SOAP 4 Administration Utility, see "NonStop SOAP Tools" (page 194).

   You must place the SDL file for the `math` service at this location. The SDL file for the `math` service is provided with the NonStop SOAP 4 distribution and is available in *<NonStop SOAP 4 Installation Directory>*`/sample_services/math/service/mathsdl.xml`.

   You may choose to copy this SDL file to *<NonStop SOAP 4 Deployment Directory>*`/services/math/src`.

2. Update the following attributes in the `mathsdl.xml` file:

   - **`Url`**

     Verify the `Url` attribute of the `sdl` element. This attribute must have a Web address pattern.

     For example,

     `Url= "/mytest"`

     where,

     `"/mytest"` is the Web address pattern entered while creating the NonStop SOAP 4 deployment directory by running the `deploy.sh` deployment script. For more information, see Step 6 in "Setting up the Deployment Environment" (page 38).

   - **`ServerAddress`**

     Update the `ServerAddress` attribute of the `sdl` element with the iTP WebServer IP address and port number, where NonStop SOAP 4 is deployed. You must set the value for the `ServerAddress` attribute in the following format:

     `ServerAddress ="http://<ip address>:<port>"`

     where,

     `<ip address>:<port>`
     is the IP address and port of iTP WebServer integrated with NonStop SOAP 4.

   - **`DDLDictionaryLocation`**

     Update the `DDLDictionaryLocation` attribute of the `Service` element with the Guardian subvolume location specified in the service DLL file.

     For example:

     `DDLDictionaryLocation="$DATA.MATH"`

Generating the WSDL File and the `services.xml` File

To generate the WSDL file and the `services.xml` file using the `SoapAdminCL` tool, complete the following steps:

1. Add the directory containing the `SoapAdminCL` executable image to the OSS `PATH` variable:

   `OSS> export PATH=<NonStop SOAP 4 Installation Directory>/tools:$PATH`

   For example, if you have installed NonStop SOAP 4 in the default location (`/usr/tandem/nssoap/t0865h01`), the OSS command to update the `PATH` environment variable is:

   `OSS> export PATH=/usr/tandem/nssoap/t0865h01/tools:$PATH`

2. Run the `SoapAdminCL` tool to generate the WSDL file and the `services.xml` file for the service using the command:

   `OSS> SoapAdminCL -o <NonStop SOAP 4 Deployment Directory> -i <SDL filename>`

   The `SoapAdminCL` command generates the following files:

   - *`<NonStop SOAP 4 Deployment Directory>`*`/services/`*`<service_name>`*`/SoapPW_`*`<service_name>`*`.wsdl`

     The `SoapPW_`*`<service_name>`*`.wsdl` file contains the service definition details and the XML schema definitions generated for the service.

   - *`<NonStop SOAP 4 Deployment Directory>`*`/services/`*`<service_name>`*`/services.xml`

     The `services.xml` file contains the service configuration details that are required by the NonStop SOAP 4 server during runtime.

- *<NonStop SOAP 4 Deployment Directory>*/client/*<service_name>*/SoapPW_*<operation name>*.html

  The SoapPW_*<operation name>*.html file is HTML client for an operation of the service.

- *<NonStop SOAP 4 Deployment Directory>*/client/*<service_name>*/pway-xsd/SoapPW_*<request_name>*.xsd

  The SoapPW_*<request_name>*.xsd file is XML schema file for request structure of an operation of the service.

- *<NonStop SOAP 4 Deployment Directory>*/client/*<service_name>*/pway-xsd/SoapPW_*<request_name>*Response0.xsd

  The SoapPW_*<request_name>*Response0.xsd file is XML schema file for response structure of an operation of the service.

NOTE: On successful execution, the SoapAdminCL tool generates client-specific XML, .html files, and .xsd files in the *<NonStop SOAP 4 Deployment Directory>*/client/*<service_name>* directory.

For the math service, invoke the SoapAdminCL tool by running the following OSS commands:

```
OSS>cd <Math Source Directory>
OSS>SoapAdminCL -o <NonStop SOAP 4 Deployment Directory> -i mathsdl.xml
```

On successful execution, the following files are generated:

- *<NonStop SOAP 4 Deployment Directory>*/services/math/SoapPW_math.wsdl
- *<NonStop SOAP 4 Deployment Directory>*/services/math/services.xml
- *<NonStop SOAP 4 Deployment Directory>*/client/math/SoapPW_add.html
- *<NonStop SOAP 4 Deployment Directory>*/client/math/SoapPW_sub.html
- *<NonStop SOAP 4 Deployment Directory>*/client/math/SoapPW_mul.html
- *<NonStop SOAP 4 Deployment Directory>*/client/math/SoapPW_div.html

The request and response XML files and the XML schema files are also generated with the WSDL file, services.xml file, and HTML clients.

Generating the Service Skeleton Files

After you generate the WSDL file and the services.xml file for the service, either create the service skeleton and client stub files manually, or use the WSDL2C tool to automate the creation of the files.

To create the service skeleton files using the WSDL2C tool, complete the following steps:

1. Set the OSS environment variable NSSOAP_HOME to the OSS location where the NonStop SOAP 4 installation directory is located:

   ```
   OSS> export NSSOAP_HOME=<NonStop SOAP 4 Installation Directory>
   ```

   For example:

   ```
   OSS> export NSSOAP_HOME=/usr/tandem/nssoap/t0865h01
   ```

   where,

   /usr/tandem/nssoap/t0865h01 is the NonStop SOAP 4 installation directory location.

2. Add the directory containing the WSDL2C executable image to the OSS PATH variable, using the command:

   ```
   OSS> export PATH=<NonStop SOAP 4 Installation Directory>/tools:$PATH
   ```

For example:

```
OSS> export PATH=/usr/tandem/nssoap/t0865h01/tools:$PATH
```

where,

`/usr/tandem/nssoap/t0865h01`
is the NonStop SOAP 4 installation directory.

3. Add the <*Java Installation Directory*>/bin directory to the PATH environment variable, using the command:

```
OSS> export PATH=<Java Installation Directory>/bin:$PATH
```

For example:

```
OSS> export PATH=/usr/tandem/java/bin:$PATH
```

where,

`/usr/tandem/java/`
is the Java installation directory.

4. Generate the service skeleton files using the WSDL2C tool. You can generate the service skeleton files using the following command:

```
OSS>  WSDL2C [options] –ss -uri [wsdl path]
```

where,

`-ss`
creates the server side skeleton files.

`-uri [wsdl path]`
specifies the location of the WSDL file.

[options] include the following:

`-d <data binding name>`
specifies the data binding option to be used.

`-o`
specifies a directory path for the generated code.

`-pn <port_name>`
enables you to select a specific port when there are multiple ports in the WSDL file.

`-sn <service_name>`
enables you to select a specific service when there are multiple services in the WSDL file.

`-wv <version>`
specifies the WSDL versions.

For example:

```
OSS> WSDL2C -o "<NonStop SOAP 4 Deployment Directory>/services/math/src" -ss -d none
     -uri "<NonStop SOAP 4 Deployment Directory>/services/math/SoapPW_math.wsdl"
```

5. On successful execution, the following service skeleton files are generated:
   - The NonStop SOAP 4 skeleton source file, where the `axis2_svc_skeleton` interface functions are implemented
   - The service skeleton source file, where you must implement the business logic for your application
   - Header file for the service skeleton file

   For example, for the `math` service, the following files are generated in the *<NonStop SOAP 4 Deployment Directory>*`/services/math/` directory:
   - `axis2_svc_skel_mathService.c`
   - `axis2_skel_mathService.c`
   - `axis2_skel_mathService.h`

Implementing the Business Logic in the Service Skeleton Files

After you generate the service skeleton and client stub files for the service, you must implement the business logic.

For example, for the `math` service, you need to implement the business logic in the `axis2_skel_mathService.c` file. To add the business logic for the add operation, update the `axis2_skel_mathService_add()` function.

A sample implementation of the `axis2_skel_mathService_add()` is available in the *<NonStop SOAP 4 Installation Directory>*`/sample_services/math/service/axis2_skel_mathService.c` file. If you do not want to edit the source files, you can replace the *<NonStop SOAP 4 Deployment Directory>*`/services/math/src/axis2_skel_mathService.c` file with the *<NonStop SOAP 4 Installation Directory>*`/sample_services/math/service/axis2_skel_mathService.c` file, which implements the business logic for all the `math` operations.

Compiling the Service Code and Deploying the Service

After you implement the business logic in the service skeleton file, you must deploy the service.

To deploy the service complete the following steps:

1. Copy the Makefile from the *<NonStop SOAP Installation Directory>*`/sample_services/math/service` directory to the *<NonStop SOAP Deployment Directory>*`/services/math/src` directory.
2. Set AXIS2C_HOME to *<NonStop SOAP 4 Deployment Directory>* using the following command:

   ```
   OSS> export AXIS2C_HOME=<NonStop SOAP 4 Deployment Directory>
   ```
3. Run the `make` command to build the `libmath.so` DDL file:

   ```
   OSS> make
   ```

   The `make` command copies the `libmath.so` file to the *<NonStop SOAP 4 Deployment Directory>*`/services/math` directory.
4. Ensure that the `services.xml` and the service DLL (`.so`) files are located in the *<NonStop SOAP 4 Deployment Directory>*`/services/math` directory.

   ```
   OSS> cd <NonStop SOAP 4 Deployment Directory>/services/<service_name>
   OSS> ls
        libmath.so services.xml
   ```

   The `math` service is deployed in NonStop SOAP 4. To access the service WSDL, go to the following Web address:

   ```
   http://< ip address>:< port number>/url_pattern/services/math?wsdl
   ```

In case of the math service, the libmath.so file and the services.xml file must be located in the <NonStop SOAP 4 Deployment Directory>/services/math directory.

The math service is now deployed.

Testing the Service

To test the math service, you must create the math client. For information on creating a math client, see "NonStop SOAP 4 Client APIs" (page 105).

To run the math client complete the following steps:

1. Access the options offered by the math service using the following Web address pattern:

   http://<ip address>:<port>/<url_pattern>/client/math/<html_client_name>

   where,

   <ip address>:<port>
   : is the IP address and port of iTP WebServer integrated with NonStop SOAP 4.

   url_pattern
   : is the string entered in Step 6 in "Setting up the Deployment Environment" (page 38). The default value is axis2c.

   client
   : is the name of the directory in <NonStop SOAP 4 Deployment Directory> where NonStop SOAP 4 HTML clients for the math service are located.

   math
   : is a sub-directory in the client directory. The math sub-directory includes the HTML clients needed to access the operations offered by the math Web service.

   <html_client_name>
   : is the name of the HTML client in the client/math directory.

   NOTE: You can access the HTML clients for the math service using the following Web address:

   http://<ip address>:<port>/<url-pattern>/client/math

   The links to HTML clients for the following math service operations are displayed on the browser:

   ```
   SoapPW_add.html - HTML client for the addition operation
   SoapPW_sub.html - HTML client for the subtraction operation
   SoapPW_mul.html - HTML client for the multiplication operation
   SoapPW_div.html - HTML client for the division operation
   ```

2. Perform the required operation. For example, to perform an addition operation, complete the following steps:
   1. Go to the SoapPW_add.html Web page.

2. Enter the values for the two parameters to be added, and then click **submit**.

   For example:

```
param1 : 20
param2 : 30
```

   On successful submission, the following result appears in the Web browser:

```
- <soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
  < soapenv:Header />
  < soapenv:Body>
     < ns1:addResponse0 xmlns:ns1="urn:cpq_tns_add">
        < ns1:response>50< /ns1:response>
     < /ns1:addResponse0>
  < /soapenv:Body>
  < /soapenv:Envelope>
```

NOTE: You can use similar steps to perform subtraction, multiplication, and division operations.

6 NonStop SOAP 4 Client APIs

This chapter describes the NonStop SOAP 4 APIs you can use to develop client applications that consume SOAP Web services.

This chapter includes the following topics:

- "NonStop SOAP 4 Client APIs" (page 105)
- "Developing NonStop SOAP 4 Clients Using Client APIs" (page 119)

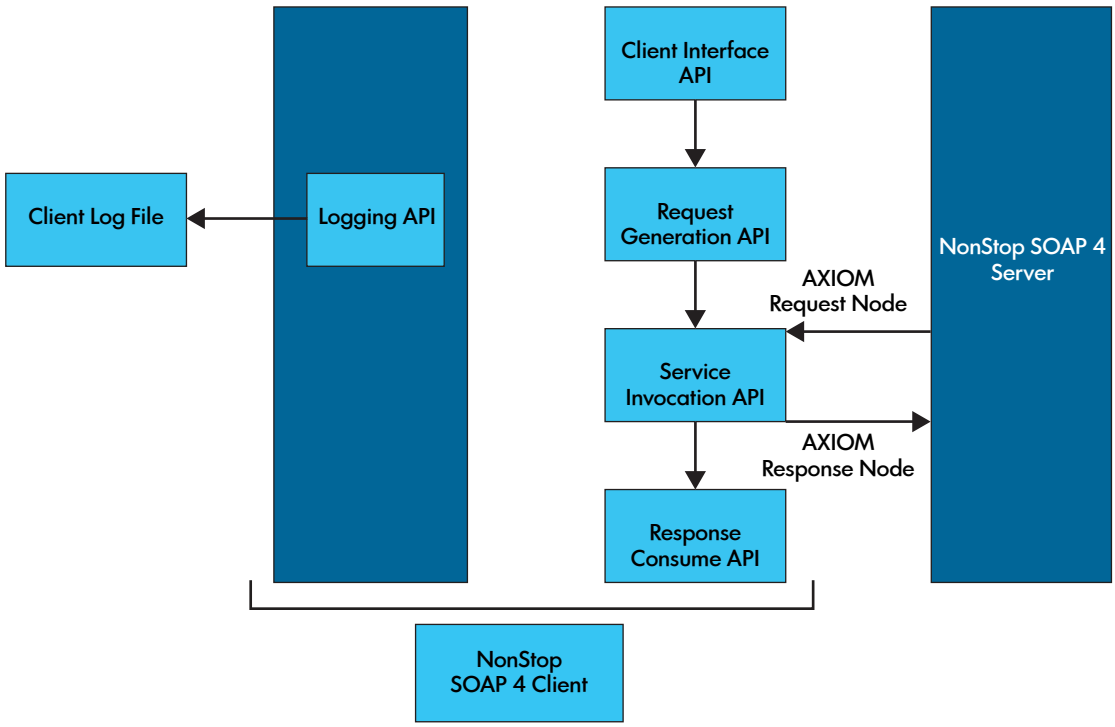
NonStop SOAP 4 Client APIs

NonStop SOAP 4 client APIs provide the following stack of function calls to develop client applications that consume SOAP Web services.

- "APIs to Implement the Client Interface with NonStop SOAP 4" (page 105)
- "APIs to Generate the Request Node and Consume the Response Node" (page 106)
- "APIs to Invoke the Web Service" (page 113)
- "APIs for Logging" (page 115)
- "ADB APIs for creating requests" (page 117)

Figure 10 shows the NonStop SOAP 4 client APIs and how they communicate with the NonStop SOAP 4 server.

Figure 10 Client APIs in NonStop SOAP 4



APIs to Implement the Client Interface with NonStop SOAP 4

Clients that are developed using NonStop SOAP 4 client APIs must implement the following functions:

- "The `axis2_svc_client_create()` Function" (page 106)
- "The `axis2_svc_client_free()` Function" (page 106)

NOTE: The `axis2_svc_client_create()` and `axis2_svc_client_free()` functions are generated by the `WSDL2C` tool for NonStop SOAP 4 clients. This tool consumes the WSDL file of the service to generate client stubs which contain the definition for these functions. For more information about the `WSDL2C` tool, see "NonStop SOAP Tools" (page 194).

The `axis2_svc_client_create()` Function

The `axis2_svc_client_create()` function creates and returns a pointer to the `axis2_svc_client_t` NonStop SOAP 4 client structure. This function also allocates memory to the service client structure and initializes the structure parameters to contain default values.

Synopsis:

```
AXIS2_EXTERN axis2_svc_client_t* axis2_svc_client_create
( const axutil_env_t *  env,
  const axis2_char_t *  client_home
)
```

Parameters:

`env`
: is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`client_home`
: is name of the directory that contains the Axis2/C repository.

Return values:

Pointer to the newly created service client structure. It returns `NULL` in case of an error and sets the corresponding error code in environment's error structure.

The `axis2_svc_client_free()` Function

The `axis2_svc_client_free()` function frees the NonStop SOAP 4 client structure from memory. This function is called to release the memory allocated to the service client structure that is no longer needed. This helps in reducing the memory footprint of the NonStop SOAP 4 client.

Synopsis:

```
AXIS2_EXTERN void axis2_svc_client_free
( axis2_svc_client_t *  svc_client,
  const axutil_env_t *  env
 )
```

Parameters:

`svc_client`
: is a pointer to the service client structure.

`env`
: is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return values:

None

APIs to Generate the Request Node and Consume the Response Node

NonStop SOAP 4 client sends a request and receives a response from the Nonstop SOAP 4 server in the form of an AXIOM node.

The AXIOM APIs that enable you to generate the request AXIOM node and consume the response AXIOM node are described in the following section:

- "Generating the Request AXIOM node using NonStop SOAP 4 APIs" (page 107)
- "Consuming the Response AXIOM node using NonStop SOAP 4 APIs" (page 111)

Generating the Request AXIOM node using NonStop SOAP 4 APIs

NonStop SOAP 4 provides APIs to generate the request AXIOM node using one of the following approaches:

- "Creating an AXIOM node using `AXIOM Node Create` APIs" (page 107)
- "Creating an AXIOM node from an XML using `AXIOM Document` APIs" (page 110)

Creating an AXIOM node using `AXIOM Node Create` APIs

NonStop SOAP 4 provides the following functions to create the response AXIOM node:

- "The `axiom_node_create()` Function" (page 107)
- "The `axiom_node_add_child()` Function" (page 107)
- "The `axiom_element_create()` Function" (page 108)
- "The `axiom_element_add_attribute()` Function" (page 108)
- "The `axiom_attribute_create()` Function" (page 109)
- "The `axiom_document_build_all()` Function" (page 109)

The `axiom_node_create()` Function

The `axiom_node_create()` function creates an AXIOM node structure.

Synopsis:

```
AXIS2_EXTERN axiom_node_t* axiom_node_create
                           (const axutil_env_t *  env)
```

Parameters:

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return value:

Pointer to the newly created node structure. If an error occurs, it returns `NULL`.

The `axiom_node_add_child()` Function

The `axiom_node_add_child()` function adds an AXIOM node as a child to a parent AXIOM node.

Synopsis:

```
AXIS2_EXTERN axis2_status_t* axiom_node_add_child
                             ( axiom_node_t* om_node,
                               const axutil_env_t *  env,
                               axiom_node_t* child
                             )
```

Parameters:

`om_node`
is an input parameter and points to the parent node. It cannot have a `NULL` value.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

child
is an input parameter and is a pointer to the child node of type axiom_node_t. It cannot have a NULL value.

Return values:

- AXIS2_SUCCESS on success.
- AXIS2_FAILURE on failure.

The axiom_element_create() Function

The axiom_element_create() function creates an AXIOM element using the specified name.

Synopsis:

```
AXIS2_EXTERN axiom_element_t* axiom_element_create
                              ( const axutil_env_t *  env,
                                axiom_node_t *  parent,
                                const axis2_char_t *  localname,
                                axiom_namespace_t *  ns,
                                axiom_node_t **  node )
```

Parameters:

env
is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

parent
is an input parameter and is a pointer to the parent of the element node to be created. It cannot have a NULL value.

localname
is an input parameter and is a pointer to the local name of the element. It cannot have a NULL value.

ns
is an input parameter and is a pointer to the namespace, if any, of the element. It can have a NULL value.

node
is an output parameter and is a pointer to the node corresponding to the created element.

Return value:

Pointer to the newly created element structure. If an error occurs, it returns NULL.

The axiom_element_add_attribute() Function

The axiom_element_add_attribute() function adds an attribute to the current element.

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_element_add_attribute
                            ( axiom_element_t *  om_element,
                              const axutil_env_t *  env,
                              axiom_attribute_t *  attribute,
                              axiom_node_t *  node )
```

Parameters:

om_element
is an input parameter and is a pointer to the element to which the attribute is added. It cannot have a NULL value.

env
is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

`attribute`
is an input parameter and is a pointer to the attribute to be added to the element. It cannot have a `NULL` value.

`node`
is an input parameter and is a pointer to the `axiom_node_t` node that contains `om_element`. It cannot have a `NULL` value.

Return values:

- `AXIS2_SUCCESS` on success.
- `AXIS2_FAILURE` on failure.

The `axiom_attribute_create()` Function

The `axiom_attribute_create()` function creates an AXIOM attribute structure.

Synopsis:

```
AXIS2_EXTERN axiom_attribute_t* axiom_attribute_create
                                ( const axutil_env_t *  env,
                                  const axis2_char_t *  localname,
                                  const axis2_char_t *  value,
                                  axiom_namespace_t *  ns )
```

Parameters:

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`localname`
is an input parameter and is a pointer to the local name of the attribute. It cannot have a `NULL` value.

`value`
is an input parameter and is a pointer to the normalized attribute value. It cannot have a `NULL` value.

`ns`
is an input parameter and is a pointer to the namespace, if any, of the attribute. It is an optional parameter and can have a `NULL` value.

Return values:

Pointer to the newly created attribute structure. If an error occurs, it returns `NULL`.

The `axiom_document_build_all()` Function

The `axiom_document_build_all()` function builds the XML input stream from the current position of the XML input stream till the end of the root element.

```
AXIS2_EXTERN axiom_node_t* axiom_document_build_all
                           ( struct axiom_document * document,
                             const axutil_env_t * env )
```

Parameters:

`document`
is an input parameter and is a pointer to the `axiom_document_t` structure to be built.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return values:

Pointer to the AXIOM node created from the XML stream.

Creating an AXIOM node from an XML using AXIOM Document APIs

If you create the service response AXIOM node from an XML message, use the AXIOM document APIs to create the AXIOM node from the response XML.

NOTE: You need not use the AXIOM document APIs if the service builds the response using AXIOM APIs. The AXIOM document APIs interpret the hierarchy and relationship between the elements in an XML message and generate the AXIOM node that mirrors the relationships in the XML message. Unlike the AXIOM document APIs, when you use the AXIOM API to build the AXIOM node, you need to set the relationships between the elements.

NonStop SOAP 4 uses the Streaming API for XML (StAX) parser to convert the XML payload into an AXIOM node. The StAX parser is a pull-based XML parser that transfers the parsing control to the client. This allows the NonStop SOAP 4 clients to request the next token from the parser. StAX parses the XML payload and converts it into the corresponding AXIOM node. The StAX parser consumes the XML payload and produces an AXIOM document structure. The AXIOM document structure is queried to retrieve the AXIOM nodes.

The NonStop SOAP 4 APIs provide the following functions to process the AXIOM document structure:

- "The axiom_document_create() Function" (page 110)
- "The axiom_document_build_all() Function" (page 110)
- "The axiom_document_free() Function" (page 111)

The axiom_document_create() Function

The axiom_document_create() function creates an axiom_document_t structure. This structure holds the AXIOM node representation of the input XML payload.

Synopsis:

```
AXIS2_EXTERN axiom_document_t* axiom_document_create
                                ( const axutil_env_t *  env,
                                  axiom_node_t *  root,
                                  struct axiom_stax_builder *  builder )
```

Parameters:

env
: is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

root
: is an input parameter and is a pointer to the root node of the XML document.

builder
: is an input parameter and is a pointer to the StAX parser.

Return Value:

Pointer to the newly created AXIOM document. If an error occurs, it returns NULL.

The axiom_document_build_all() Function

The axiom_document_build_all() function builds the XML input stream from the current position of the StAX parser till the root element is complete.

Synopsis:

```
AXIS2_EXTERN axiom_node_t* axiom_document_build_all
                              ( struct axiom_document * document,
                                const axutil_env_t * env )
```

Parameters:

`document`

is an input parameter and is a pointer to the `axiom_document_t` structure to be built.

`env`

is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return Value:

Pointer to the AXIOM node built from the document. If an error occurs, it returns `NULL`.

The `axiom_document_free()` Function

The `axiom_document_free()` function is used to free the document structure from memory.

Synopsis:

```
AXIS2_EXTERN void axiom_document_free
                                ( struct axiom_document_t* document,
                                  const axutil_env_t *  env)
```

Parameters:

`document`

is an input parameter and is a pointer to the document structure that becomes free from the memory.

`env`

is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return values:

None.

Consuming the Response AXIOM node using NonStop SOAP 4 APIs

NonStop SOAP 4 provides the following functions to consume the response AXIOM node:

- "The `axiom_element_get_qname()` Function" (page 111)
- "The `axiom_node_get_data_element()` Function" (page 90)
- "The `axiom_attribute_get_qname()` Function" (page 112)
- "The `axiom_element_get_attribute()` Function" (page 112)

The `axiom_element_get_qname()` Function

The `axiom_element_get_qname()` function returns the qualified name of a given AXIOM element.

Synopsis:

```
AXIS2_EXTERN axutil_qname_t* axiom_element_get_qname
                               (axiom_element_t* om_element,
                                const axutil_env_t *  env,
                                axiom_node_t* om_node)
```

Parameters:

`om_element`

is an input parameter and is a pointer to the element whose `qname` must be retrieved.

`env`

is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

om_node
is an input parameter and is a pointer to the axiom_node_t node that contains om_element.

Return Values:

Pointer to the qname of the element. If an error occurs, it returns NULL.

The axiom_node_get_data_element() Function

The axiom_node_get_data_element() function retrieves the AXIOM element structure from the AXIOM node.

Synopsis:

```
AXIS2_EXTERN void axiom_node_get_data_element
                            ( axiom_node_t* om_node,
                              const axutil_env_t * env)
```

Parameters:

om_node
is an input parameter and is a pointer to the node for which a data element is required.

env
is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

Return Values:

Pointer to the structure included in the AXIOM node. If an error occurs, it returns NULL. You must cast the returned void pointer into an axiom_element_t* data type before using the pointed data element further in the code.

The axiom_attribute_get_qname() Function

The axiom_attribute_get_qname() function retrieves the qualified name of an AXIOM attribute.

Synopsis:

```
AXIS2_EXTERN axutil_qname_t* axiom_attribute_get_qname
                                (struct axiom_attribute_t* om_attribute,
                                  const axutil_env_t * env)
```

Parameters:

om_attribute
is an input parameter and is a pointer to the attribute structure for which qname must be returned.

env
is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

Return Values:

Pointer to the qname for a given attribute. If an error occurs, it returns NULL.

The axiom_element_get_attribute() Function

The axiom_element_get_attribute() function finds the attribute using the given qualified name.

Synopsis:

```
AXIS2_EXTERN axiom_attribute_t* axiom_element_get_attribute
                               ( axiom_element_t *  om_element,
                                 const axutil_env_t * env,
                                 axutil_qname_t *  qname )
```

Parameters:

`element`

is an input parameter and is a pointer to the element whose attribute must be found.

`env`

is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return Values:

Pointer to the attribute with the given `qname`. If an error occurs, it returns `NULL` and sets the error code in the environment error structure.

APIs to Invoke the Web Service

NonStop SOAP 4 clients can invoke their respective NonStop SOAP 4 services in synchronous (blocking) or asynchronous (non-blocking) mode using the following APIs:

- "The `axis2_svc_client_send_receive()` Function" (page 113)
- "The `axis2_svc_client_send_receive_non_blocking()` Function" (page 113)
- "The `axis2_svc_client_send_robust()` Function" (page 114)
- "The `axis2_svc_client_fire_and_forget()` Function" (page 114)

The `axis2_svc_client_send_receive()` Function

The `axis2_svc_client_send_receive()` function sends an XML request and receives the XML response. This function is used to interact with a service operation whose message exchange pattern (MEP) is IN-OUT.

Synopsis:

```
AXIS2_EXTERN axiom_node_t* axis2_svc_client_send_receive
( axis2_svc_client_t *  svc_client,
  const axutil_env_t *  env,
  const axiom_node_t *  payload
 )
```

Parameters:

`svc_client`

is a pointer to the service client structure.

`env`

is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`payload`

is a pointer to the AXIOM node, which represents the XML payload to be sent. The caller controls the payload until the service client releases it.

Return Value:

Is a pointer to the OM node representing the XML response.

The `axis2_svc_client_send_receive_non_blocking()` Function

The `axis2_svc_client_send_receive_non_blocking()` function sends an XML request and receives the XML response, but it does not block for the response. This function is used to interact in the non-blocking mode with a service operation, which has an IN-OUT message exchange pattern.

Synopsis:

```
AXIS2_EXTERN void axis2_svc_client_send_receive_non_blocking
( axis2_svc_client_t *  svc_client,
```

```
  const axutil_env_t *  env,
  const axiom_node_t *  payload,
  axis2_callback_t *  callback
 )
```

Parameters:

`svc_client`
: is a pointer to the service client structure.

`env`
: is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`payload`
: is a pointer to the AXIOM node representing the XML payload to be sent. The caller controls the payload until the service client releases it.

`callback`
: is a pointer to the callback structure used to capture response. The callback structure is passed as an argument to the `axis2_svc_client_send_receive_non_blocking()` function. The program runs in parallel and returns the response back to the calling function as and when it becomes available.

Return values:

Void

The `axis2_svc_client_send_robust()` Function

The `axis2_svc_client_send_robust()` function is used to send an XML message. This function invokes a service operation whose MEP is Robust Out-Only. If a fault triggers on the server side, the `axis2_svc_client_send_robust()` function reports an error to the caller.

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_svc_client_send_robust
( axis2_svc_client_t *  svc_client,
  const axutil_env_t *  env,
  const axiom_node_t *  payload
 )
```

Parameters:

`svc_client`
: is a pointer to the service client structure.

`env`
: is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`payload`
: is a pointer to the AXIOM node representing the XML payload to be sent. The caller controls the payload until the service client releases it.

Return values:

- `AXIS2_SUCCESS` on success.
- `AXIS2_FAILURE` on failure.

The `axis2_svc_client_fire_and_forget()` Function

The `axis2_svc_client_fire_and_forget()` function sends a message and forgets about it. This function is used to interact with a service operation whose MEP is In-Only.

Synopsis:

```
AXIS2_EXTERN void axis2_svc_client_fire_and_forget
( axis2_svc_client_t *  svc_client,
  const axutil_env_t *  env,
  const axiom_node_t *  payload
 )
```

Parameters:

`svc_client`
is a pointer to the service client structure.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`payload`
is a pointer to the AXIOM node representing the XML payload to be sent. The caller controls the payload until the service client releases it.

Return Value:

Void

APIs for Logging

The NonStop SOAP 4 API includes the AXIOM logging APIs used to create and log messages. The logging API enables you to specify the name and location of the log file and the level of messages to be logged.

Creating the log file structure

The `axutil_log_create()` function is called to create the log file structure.

Synopsis:

```
AXIS2_EXTERN axutil_log_t *  axutil_log_create
                             (axutil_allocator_t *allocator,
                              axutil_log_ops_t *ops,
                              const axis2_char_t *stream_name)
```

Parameters:

`allocator`
is an input parameter and a pointer to the `axutil_allocator` structure.

`ops`
is an input parameter and is a pointer to the `options` structure for logging.

`stream_name`
is an input parameter and is a pointer to the file name and location where the log must be created.

Return Values:

Pointer to the newly created log structure. If an error occurs, it returns `NULL`.

Logging messages at different log levels

After you have created the log structure, use the following functions to log messages at different logging levels:

- "The `axutil_log_impl_log_warning()` Function" (page 116)
- "The `axutil_log_impl_log_info()` Function" (page 116)
- "The `axutil_log_impl_log_user()` Function" (page 116)

- "The axutil_log_impl_log_debug() Function" (page 116)
- "The axutil_log_impl_log_trace() Function" (page 116)

The axutil_log_impl_log_warning() Function

The axutil_log_impl_log_warning() function logs warning messages in the specified log file.

Synopsis:

```
AXIS2_EXTERN void  axutil_log_impl_log_warning
                   (axutil_log_t *log, const axis2_char_t *filename,
                    const int linenumber, const axis2_char_t *format,...)
```

The axutil_log_impl_log_info() Function

The axutil_log_impl_log_info() function logs information messages in the specified log file.

Synopsis:

```
AXIS2_EXTERN void  axutil_log_impl_log_info
                   (axutil_log_t *log, const axis2_char_t *filename, const int linenumber, const axis2_char_t
*format,...)
```

The axutil_log_impl_log_user() Function

The axutil_log_impl_log_user() function logs user-level debug messages in the specified log file.

Synopsis:

```
AXIS2_EXTERN void  axutil_log_impl_log_user
                  (axutil_log_t *log, const axis2_char_t *filename,
                   const int linenumber, const axis2_char_t *format,...)
```

The axutil_log_impl_log_debug() Function

The axutil_log_impl_log_debug() function logs the debug level logs. It logs all the information in the specified log file.

Synopsis:

```
AXIS2_EXTERN void  axutil_log_impl_log_debug
                  (axutil_log_t *log, const axis2_char_t *filename,
                   const int linenumber, const axis2_char_t *format,...)
```

The axutil_log_impl_log_trace() Function

The axutil_log_impl_log_trace() function logs the trace level logs. It is enabled with the compiler time option AXIS2_TRACE.

Synopsis:

```
AXIS2_EXTERN void  axutil_log_impl_log_trace
                   (axutil_log_t *log, const axis2_char_t *filename,
                    const int linenumber, const axis2_char_t *format,...)
```

Releasing the log structure

You can use the axutil_log_free AXIOM API call to release the log structure.

Synopsis:

```
AXIS2_EXTERN void  axutil_log_free
                   (axutil_allocator_t *allocator,
                    struct axutil_log *log)
```

ADB APIs for creating requests

The WSDL2C with ADB (Axis2) data binding creates getter and setter methods at both schema level and the message level. You can use these APIs to create SOAP requests. You must implement the following functions to develop applications using NonStop SOAP 4 client ADB APIs.

- "The `adb_<messagename>_create()` Function"
- "The `adb_<messagename>_free()` Function"
- "The `adb_<complextype>_create_with_values()` Function"
- "The `adb_<messagename>_set_<complextype>()` Function"
- "The `adb_<messagename>_get_<complextype>()` Function"

The `adb_<messagename>_create()` Function

The `adb_<messagename>_create()` function creates and returns a pointer to the `<messagetype>` structure. This function allocates memory for the `<messagetype>` structure and initializes the structure parameters with default values.

Synopsis:

```
adb_<messagetype> * AXIS2_CALL adb_<messagename>_create
                    (axutil_env_t *env)
```

Parameters:

`env`

is an input parameter and points to the `axutil_env_t` environment structure. The value cannot be NULL.

Return value:

The function returns a pointer to the `adb_<messagetype>` structure. It returns NULL in case of an error and sets the corresponding error code in the environment's error structure.

The `adb_<messagename>_free()` Function

The `adb_<messagename>_free()` function frees the memory allocated to a `<messagetype>` structure.

Synopsis:

```
axis2_status_t AXIS2_CALL adb_<messagename>_free
                    (<messagetype *> _object, const axutil_env_t *env)
```

Parameters:

`_object`

is an input parameter and points to the structure to be freed. The value cannot be NULL.

`env`

is an input parameter and points to the `axutil_env_t` structure.

Return value:

The return value is the status of the operation. The function returns AXIS2_SUCCESS if the object is freed, else returns AXIS2_FAILURE.

The `adb_<complextype>_create_with_values()` Function

The `adb_<complextype>_create_with_values()` function creates and returns a pointer to the `<complextype>` element in the NonStop SOAP 4 client structure. This function also allocates memory to the `<complextype>` element in the structure and initializes the structure parameters with the values provided.

Synopsis:

```
adb_<complextype> * AXIS2_CALL adb_<complextype>_create_with_values
                    (const axutil_env_t *env, paramtype param)
```

Parameters:

`env`
is an input parameter and points to the `axutil_env_t` structure.

`param`
is an input parameter.

Return value:

The function returns the pointer to the `<complextype>` element in the structure. It returns NULL in case of an error and sets the error code in the environment's error structure.

The `adb_<messagename>_set_<complextype>()` Function

The `adb_<messagename>_set_<complextype>()` function sets the `<complextype>` element in the message.

Synopsis:

```
axis2_status AXIS2_CALL adb_<messagename>_set_<complextype>
                    (adb_<messagetype>* _messageStruct,
        const axutil_env_t *env,
        <complextype> _complesStruct)
```

Parameters:

`_messageStruct`
is an input parameter and points to the `<messagetype>` structure, which must be updated with the `<complextype>` element.

`env`
is an input parameter and points to the `axutil_env_t` structure. The value cannot be NULL.

`_complexStruct`
is an input parameter and points to the `<complextype>` element, which will be set in `_messageStruct`.

Return value:

The function returns the status of the operation. It returns AXIS2_SUCCESS if the message structure is set, else returns AXIS2_FAILURE.

The `adb_<messagename>_get_<complextype>()` Function

The `adb_<messagename>_get_<complextype>()` function gets the `<complextype>` elements from the response message.

Synopsis:

```
<complextype> AXIS2_CALL adb_<messagename>_get_<complextype>
                    (adb_<messagetype>* _messageStruct, const axutil_env_t *env)
```

Parameters:

`_messageStruct`
is an input parameter and points to the `<messagetype>` structure. The `<complextype>` element is extracted from this structure.

`env`
is an input parameter and points to the `axutil_env_t` structure. The value cannot be NULL.

Return value:

Returns the `<complextype>` element structure that is extracted from `_messageStruct`.

Developing NonStop SOAP 4 Clients Using Client APIs

This section describes the procedure to develop and deploy a NonStop SOAP 4 client using the NonStop SOAP 4 client API. This section describes the procedure to develop a client for the `math` service.

To develop NonStop SOAP 4 clients, use one of the following:

- The client API stack

  The client API stack enables you to develop NonStop SOAP 4 clients manually. This approach provides higher flexibility to customize the client applications based on your business requirements.

- The `WSDL2C` tool

  The `WSDL2C` tool is a Java-based tool that is distributed with NonStop SOAP 4. You can use this tool to generate the client stub and service skeleton files in the C programming language, based on the WSDL file for the service. Your application business logic must be integrated with the stub files.

  For most applications, the `WSDL2C` tool is used to generate the client stub files because you need not write the basic interface code manually. For more information on the `WSDL2C` tool, see "NonStop SOAP Tools" (page 194).

Developing and deploying NonStop SOAP 4 clients involves the following tasks:

- Generating NonStop SOAP 4 Client Stubs using the `WSDL2C` tool
- Implementing Business Logic in Client Stubs
- Compiling the Client Code and Deploying the Client
- Testing the Client

Generating NonStop SOAP 4 Client Stubs using the `WSDL2C` tool

To create client stubs using the `WSDL2C` tool, complete the following steps:

NOTE: Ensure that the WSDL file for your service is located at `< NonStop SOAP 4 Deployment Directory>/services/< service_name>`. If the WSDL file is not present, to create a WSDL file for your service, use the `SoapAdminCL` tool. For information, see NonStop SOAP 4 Configuration Files (page 177).

1. Set the OSS environment variable `NSSOAP_HOME` to the OSS location where the NonStop SOAP 4 installation directory is located:

   ```
   OSS> export NSSOAP_HOME=<NonStop SOAP 4 Installation Directory>
   ```

   For example:

   ```
   OSS> export NSSOAP_HOME=/usr/tandem/nssoap/t0865h01
   ```

   where,

   `/usr/tandem/nssoap/t0865h01`

   is the NonStop SOAP 4 installation directory location.

2. Add the directory containing the `WSDL2C` executable image to the OSS `PATH` variable.

   ```
   OSS> export PATH=<NonStop SOAP 4 Installation Directory>/tools:$PATH
   ```

   For example:

   ```
   OSS> export PATH=/usr/tandem/nssoap/t0865h01/tools:$PATH
   ```

3. Add the `<Java Installation Directory>/bin` directory to the `PATH` environment variable, using the command:

   ```
   OSS> export PATH=<Java Installation Directory>/bin:$PATH
   ```

For example:

```
OSS> export PATH=/usr/tandem/java/bin:$PATH
```

where,

`/usr/tandem/java/`
is the Java installation directory.

4. Create a directory named `src` in *<NonStop SOAP 4 Deployment Directory>*`/client/`*<service_name>* using the following command:

   ```
   OSS> mkdir <NonStop SOAP 4 Deployment Directory>/client/<service_name>/src
   ```

   For example:

   ```
   OSS> mkdir <NonStop SOAP 4 Deployment Directory>/client/math/src
   ```

   The *<NonStop SOAP 4 Deployment Directory>*`/client/math/src` directory will be called the *<Math Client Source Directory>* throughout the chapter.

5. Generate the client stub files using the `WSDL2C` command:

   ```
   OSS>  WSDL2C [options] -uri [wsdl path]
   ```

   where,

   `[options]` includes the following:

   `-o` *<output location>*
   specifies an output location where the client stubs and build scripts (`build.sh`) must be generated.

   `-a`
   generates the client stubs to invoke a Web service in asynchronous or non-blocking mode. In this mode, the client sends a request to the Web service and continues processing without waiting for a response.

   `-d` *<data binding>*
   specifies the data binding, if any, to be used by the client while communicating with the service.

   NOTE: Use the `-u` option if the data binding value is “adb”.

   `-s`
   generates the client stubs to invoke a Web service in synchronous mode or blocking mode. In this mode, the client sends a request to the Web service and waits for a response.

   NOTE: By default, the `WSDL2C` tool runs in the synchronous mode.

   `-wv` *<version>*
   specifies the WSDL file version.

   `-f`
   flattens the generated files.

   `-or`
   overwrites the existing files.

   `-u`
   unpacks the data binding classes.

   NOTE: Do not use the `-a` and `-s` options together.

   `-uri` *<wsdl_path>*
   specifies the location of the WSDL file.

For example:

```
OSS> WSDL2C -o "<NonStop SOAP 4 Deployment Directory>/client/math/src" -d none
     -uri "<NonStop SOAP 4 Deployment Directory>/services/math/SoapPW_math.wsdl"
```

NOTE: In this example, ensure that the `math` service is deployed in the NonStop SOAP 4 deployment before running the `OSS> WSDL2C [options] -u –uri [wsdl path]` command.

6. On successful execution, the following client stubs will be generated:
   - Client source stub file that will implement the `main()` function called when an executable file is run.
   - The header file for the client source stub file that holds the declarations of the functions that are implemented in the client stub source file.

   In case of the `math` service, the following files are generated in the *<NonStop SOAP 4 Deployment Directory>*`/client/math/src` directory:
   - `axis2_stub_mathService.c`
   - `axis2_stub_mathService.h`

Implementing Business Logic in Client Stubs

After generating the client stubs, you must implement the processing logic in the `main()` function in the `axis2_stub_mathService.c` file.

You must code the business logic to implement all four operations, namely addition, subtraction, multiplication, and division using the NonStop SOAP 4 Client API.

The complete code for the `math` client is present in the *<NonStop SOAP 4 Installation Directory>*`/sample_services/math/client/axis2_stub_mathService.c` file. You may choose to replace the *<Math Client Source Directory>*`/axis2_stub_mathService.c` file with this file.

You can also copy the `axis2_stub_mathservice.c` file from *<NonStop SOAP 4 Installation Directory>*`/sample_services/math/client` to the OSS location *<NonStop SOAP 4 Deployment Directory>*`/client/math/src`.

Compiling the Client Code and Deploying the Client

After the client stubs are generated and updated with the `main()` function, and the business logic is implemented in the service skeleton, you must compile the client code.

The client code can be complied either using the Makefile or manually.

Compiling the Client Code using the Makefile

To compile the client code using the Makefile and deploy the client, complete the following steps:

1. Copy the Makefile of the `math` client to *<Math Client Source Directory>*.

   NOTE:
   - You must copy the Makefile of the `math` client to *<Math Client Source Directory>* because the `WSDL2C` tool does not generate the Makefile.
   - The Makefile of the `math` client is located in *<NonStop SOAP 4 Installation Directory>*`/sample_services/math/client/Makefile`.

2. Set AXIS2C_HOME to *<NonStop SOAP 4 Deployment Directory>* using the following command:

   ```
   OSS> export AXIS2C_HOME=<NonStop SOAP 4 Deployment Directory>
   ```

3. Run the `Make` command to build your `math.exe` file:

```
OSS> make
```

The `math.exe` file is created at *<NonStop SOAP 4 Deployment Directory>*`/client/math/src`.

4. Ensure that the `services.xml` and the service DLL (`.so`) files are located in the *<NonStop SOAP 4 Deployment Directory>*`/services/`*<service_name>* directory.

```
OSS> cd <NonStop SOAP 4 Deployment Directory>/services/<service_name>
OSS> ls
     lib <service_name>.so services.xml
```

For the `math` service, the `libmath.so` and the `services.xml` file must be located in *<NonStop SOAP 4 Deployment Directory>*`/services/math`.

This completes the deployment of the `math` client.

Compiling the Client Code manually

To compile the client code manually and deploy the client, complete the following steps:

1. Compile the client code using the C89 compiler with the following options:

   Table 3 lists the C89 compiler options.

Table 3 C89 Compiler Options

C89 Compiler Options	Description
`-Wextensions`	Enables HP extensions.
`-g`	Generates debugging symbols.
`-I`	Includes the directory that contains the required header files. In NonStop SOAP 4 the required header files are located at *<NonStop SOAP 4 Installation Directory>*`/include`.
`-Wsystype=oss`	Specifies the OSS execution environment.
`-Wtarget=tns/e`	Specifies the tns/e system environment.
`-Weld="-unres_symbols Ignore"`	Ignores the missing symbol reference.
`-Weld=-Noverbose`	Avoids printing the library file locations on the screen.
`-l`	Includes the libraries to be linked.
`-WBdynamic`	Dynamically links the libraries specified in the `-l` command. In NonStop SOAP 4 the required library files are located at *<NonStop SOAP 4 Installation Directory>*`/lib`.

2. Run the following command:

```
c89 -c axis2_stub_mathService.c -o math.exe \
-Wextensions \
-g \
-I/usr/tandem/nssoap/t0865h01/include \
-Wsystype=oss \
-Wtarget=tns/e \
-Weld="-unres_symbols Ignore" \
-Weld=-Noverbose \
-Weld="-first_l /usr/tandem/nssoap/t0865h01/lib" "-L /usr/local/lib" \
-WBdynamic \
-l /usr/tandem/nssoap/t0865h01/lib/libaxis2_engine.so \
-l /usr/tandem/nssoap/t0865h01/lib/libneethi.so \
-l /usr/tandem/nssoap/t0865h01/lib/libaxutil.so \
-l /usr/tandem/nssoap/t0865h01/lib/libaxis2_axiom.so \
-l /usr/tandem/nssoap/t0865h01/lib/libaxis2_parser.so \
-l /usr/tandem/nssoap/t0865h01/lib/libaxis2_http_receiver.so \
```

```
-l /usr/tandem/nssoap/t0865h01/lib/libaxis2_http_sender.so \
-l /usr/tandem/nssoap/t0865h01/lib/libguththila.so
```

3. Ensure that the `services.xml` and the service DLL (`.so`) files are located in the *<NonStop SOAP 4 Deployment Directory>*`/services/`*<service_name>* directory.

   ```
   OSS> cd <NonStop SOAP 4 Deployment Directory>/services/<service_name>
   OSS> ls
        lib <service_name>.so services.xml
   ```

   For the `math` service, the `libmath.so` and the `services.xml` file must be located in *<NonStop SOAP 4 Deployment Directory>*`/services/math`.

   This completes the deployment of the `math` client.

Testing the Client

To run the `math` client, complete the following steps:

1. Provide `execute` permissions to the executable file `math.exe`:

   ```
   OSS> chmod +x math.exe
   ```

2. Set the `AXIS2C_HOME` environment variable to *<NonStop SOAP 4 Deployment Directory>* for the log files to be generated inside the *<NonStop SOAP 4 Deployment Directory>*`/logs` folder.

   ```
   OSS> export AXIS2C_HOME=<NonStop SOAP 4 Deployment Directory>
   ```

3. Execute the `math` client using the following command.

   ```
   OSS> cd <Math Client Source Directory>
   OSS> ./math.exe <operation> <param1> <param2> <endpoint url>
   ```

   where,

   <operation> is the operation name: `add`, `sub`, `mul`, or `div`.

   <param1> and *<param2>* are the integer values being passed to the service.

   <endpoint url> is the iTP WebServer address in the format.

   ```
   http://<iTP WebServer address>:<portnumber>
          /<url_pattern>/services/math
   ```

7 Customizing NonStop SOAP 4 Message Processing

NonStop SOAP 4 processes a request based on the default message process. This chapter provides information on customizing the default NonStop SOAP 4 message process using Phases, Modules, Handlers, and Message Receiver User Functions.

This chapter describes the following topics:

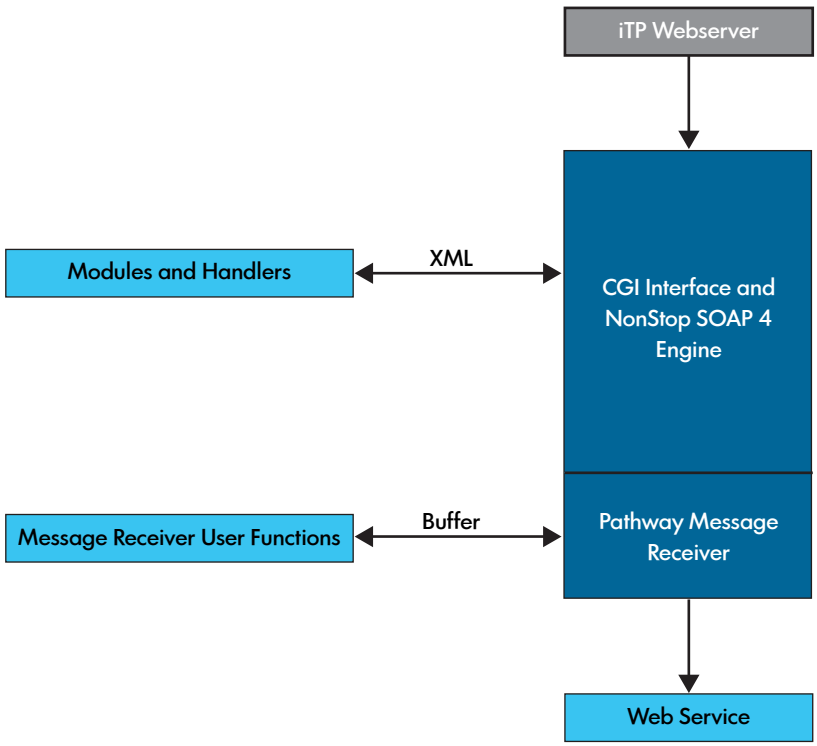
- "Overview" (page 124)
- "Customizing the NonStop SOAP 4 Message Process Using Handlers" (page 126)
- "Customizing the NonStop SOAP 4 Message Process Using Message Receiver User Functions" (page 137)

Overview

You can customize NonStop SOAP 4 using:

- "Modules and Handlers" (page 124)
- "Message Receiver User Functions" (page 125)

Figure 11 Customizing NonStop SOAP 4



Modules and Handlers

Modules and handlers are the user-written plug-ins that attach to the NonStop SOAP 4 engine during different stages in the SOAP message process and modify the SOAP message headers. Modules and handlers interrupt the default process of the SOAP message inside NonStop SOAP 4 and change the SOAP message. Modules and handlers can be attached at the global level, which means that the handlers are called for every service invocation in a NonStop SOAP 4 deployment, or at the service level, which means that the handlers are called only when a particular service in invoked.

For more information on modules and handlers, see "Request Processing in NonStop SOAP 4" (page 127).

Message Receiver User Functions

In the NonStop SOAP 4 architecture, a message receiver converts the incoming SOAP request from the NonStop SOAP 4 internal format (AXIOM node) to a format that is compatible with the deployed service.

NonStop SOAP 4 supports the following message receivers:

- Pathway message receiver: used to invoke services running as Pathway applications or NonStop processes.
- Raw XML message receiver: used when the service is developed using NonStop SOAP 4 service APIs and is implemented as a DLL.

NOTE: MRUF can be attached only to the pathway message receiver.

The message receiver is attached by NonStop SOAP 4 during service invocation and is configured using the `messageReceiver` element in the `axis2.xml` or `services.xml` configuration file.

For example:

```
<messageReceiver class="axis2_pway_receiver"/>
```

NonStop SOAP 4 allows you to customize or extend the default behavior of the Pathway message receiver using Message Receiver User Functions (MRUF). MRUFs enable you to modify the message buffer, Pathway, or NonStop process service invocation attributes, and customize the message flow in the Pathway message receiver phase. MRUFs can be attached at the global level, which means that the MRUF is called for every service invocation in a NonStop SOAP 4 deployment, or at the service level, which means that the MRUF is called only when a particular service in invoked.

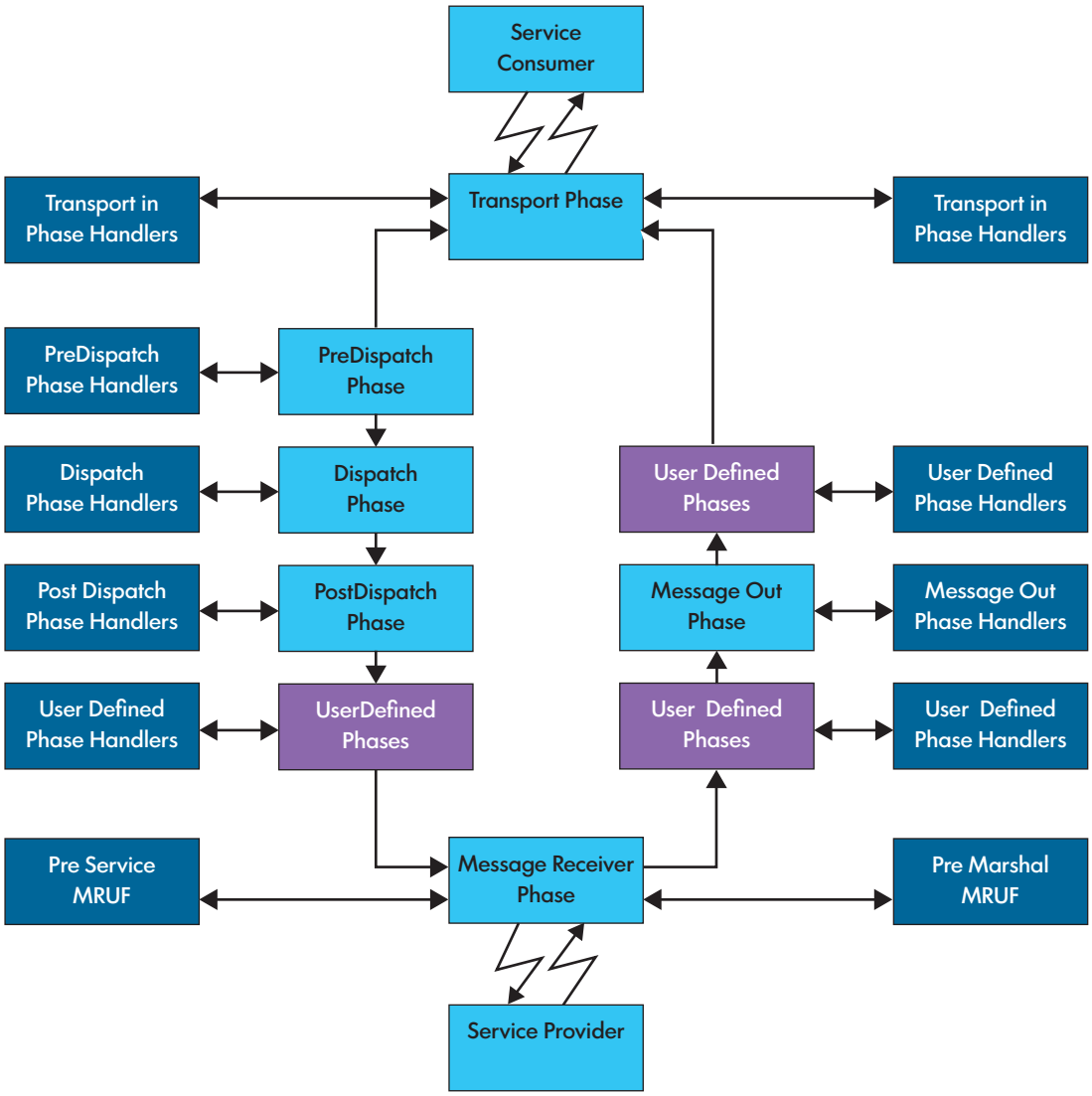
For more information about customizing the Pathway message receiver, see "Customizing the NonStop SOAP 4 Message Process Using Message Receiver User Functions" (page 137).

NonStop SOAP 4 Message Processing Customization

NonStop SOAP 4 allows you to customize the default message process using user-defined phases, modules, handlers, and MRUFs attached to the Pathway message receiver.

Figure 12 shows the execution points where handlers can be attached to the default NonStop SOAP 4 message process to customize its behavior.

Figure 12 Modified Message Process in NonStop SOAP 4



Customizing the NonStop SOAP 4 Message Process Using Handlers

To customize the NonStop SOAP 4 message process using handlers, you must have a good understanding of the request process in NonStop SOAP 4.

Customizing the NonStop SOAP 4 server using handlers involves the following steps:

- "Creating a User-Defined Phase" (page 131)
- "Deploying and Attaching a Module" (page 132)
- "Developing a Sample Module for NonStop SOAP 4" (page 132)

NOTE: In most cases, handlers that are attached with the default phases of NonStop SOAP 4 meet most of the customization requirements. Therefore, you rarely need to create a user-defined phase.

This section describes the request processing flow in NonStop SOAP 4 followed by the steps to customize NonStop SOAP 4 message using handlers.

Request Processing in NonStop SOAP 4

Request processing in NonStop SOAP 4 takes place through the interaction of the following components:

- "Flows" (page 127)
- "Phases" (page 128)
- "Handlers" (page 130)
- "Modules" (page 130)

The default message process in NonStop SOAP 4 uses only flows and phases; it does not use handlers and modules. You can customize the default message process using handlers and modules to meet your business requirements.

Flows

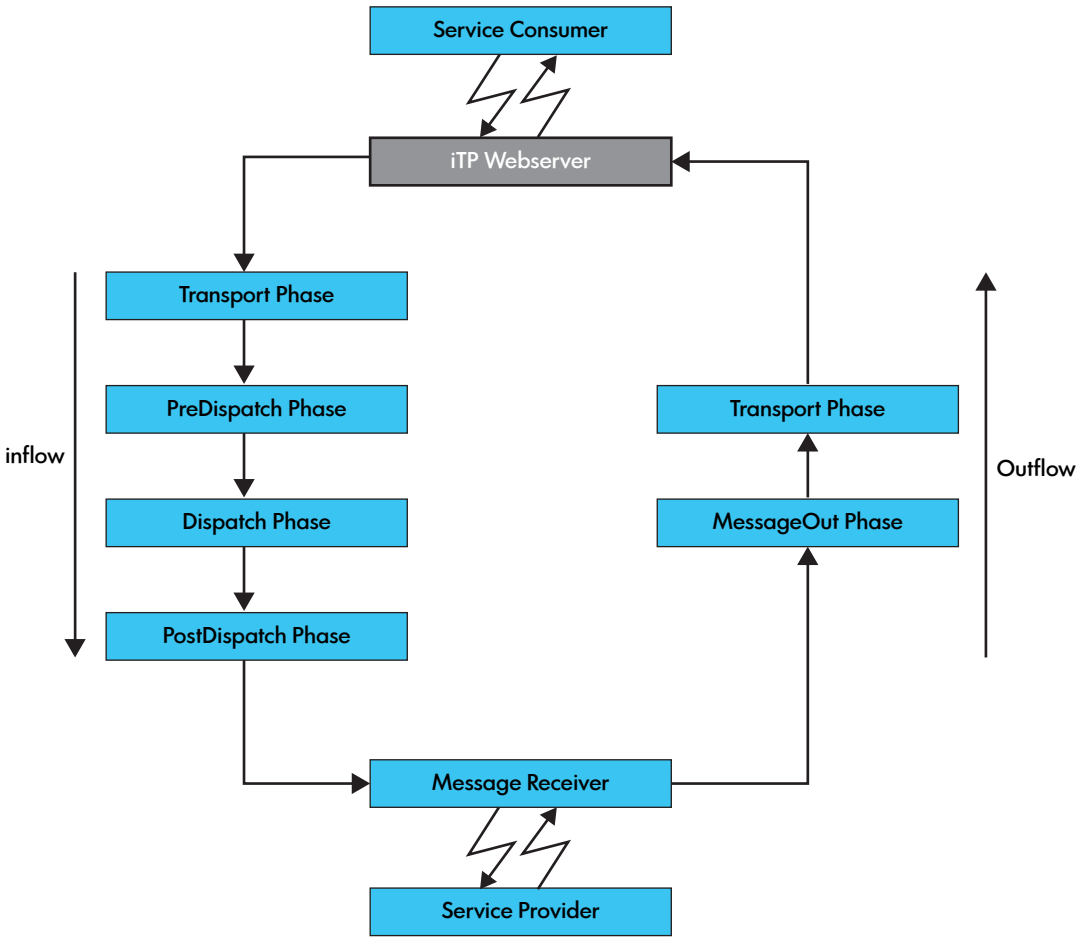
A Flow is a sequence of activities where messages flow in and out of NonStop SOAP 4.

NonStop SOAP 4 invokes the following flows based on the configured message exchange pattern (MEP):

- `inflow`: invoked when a client sends a request to the service.
- `infaultflow`: invoked during request processing.
- `outflow`: invoked when a service sends a response to a client.
- `outfaultflow`: invoked during response processing.

Figure 13 illustrates the message process in NonStop SOAP 4.

Figure 13 Message Process in NonStop SOAP 4



Phases

A Phase is a stage of processing or a time interval in the message process. A collection of phases forms a Flow. The phases within a flow, and the order in which they are invoked, are defined within the `phaseOrder` element in the `axis2.xml` configuration file.

For more information on the `axis2.xml` configuration file, see "NonStop SOAP 4 Configuration Files" (page 177).

Pre-defined phases in `inflow`

The `inflow` comprises a set of pre-defined phases in the following sequence:

1. "`Transport`" (page 128)
2. "`PreDispatch`" (page 128)
3. "`Dispatch`" (page 128)
4. "`PostDispatch`" (page 129)

`Transport`

This phase processes transport-specific information, such as validating incoming messages and adding data to the message context structure.

The handlers attached to the `Transport` phase can access transport-specific information and perform the following tasks:

- Receive raw XML requests
- Read `Transport` headers
- Modify `Transport` headers
- Validate incoming messages by examining the `Transport` headers
- Add data to the message context structure
- Update data in the message context structure

`PreDispatch`

This phase populates the message context structure before forwarding the message to the `Dispatch` phase. For example, the addressing headers of the SOAP message are processed in this phase. The addressing handlers extract information from the message headers and include the information in the message context structure.

The handlers attached to the `PreDispatch` phase can access the message context structure. The handlers in this phase perform the following tasks:

- Read SOAP headers for SOAP messages
- Modify SOAP headers for SOAP messages
- Remove SOAP headers from SOAP messages
- Add data to the message context structure
- Update data in the message context structure

`Dispatch`

This phase locates the required service and operation for which the message in the request is intended.

The handlers attached to the PostDispatch phase can access the service name and operation name of the target service for which the message is intended after the PostDispatch phase. The handlers in this phase perform the following tasks:

- Get/Set the service name
- Get/Set the operation name
- Update the message context structure

PostDispatch

This phase verifies if the dispatchers have been able to find a service and an operation for the message. If not, the processing of the message halts and an error is reported.

NOTE: NonStop SOAP 4 does not allow you to change the sequence of inflow.

The handlers in this phase verify if the dispatchers have been able to find a service and an operation for the message. If a service is not found, the handlers report an error. The handlers in this phase perform the following tasks:

- Get/Set the service name
- Get/Set the operation name
- Skip the service invocation
- Update the message context structure

User-defined phases in inflow

You can define phases in the inflow only after the PostDispatch phase. This is because the default phase order of pre-defined phases cannot be changed and the first phase in inflow must be the transport phase.

The handlers in the user-defined phase perform the following tasks:

- Get/Set the service name
- Get/Set the operation name
- Skip the service invocation
- Update the message context structure

For example:

```
<phaseOrder type="inflow">
      <phase name="Transport"/>
      <phase name="PreDispatch"/>
      <phase name="Dispatch"/>
      <phase name="PostDispatch"/>
      <phase name="myPhase"/>
</phaseOrder>
```

where,

myPhase
  is a user-defined phase.

Pre-defined phases in outflow

The outflow comprises a set of pre-defined phases in the following sequence:

1. "MessageOut" (page 130)
2. "Transport" (page 130)

MessageOut

This phase executes the transport handlers from the associated transport configuration. The last handler is always a transport sender that sends the SOAP message to the target endpoint.

The handlers attached to the MessageOut phase can access the response message. Handlers attached to this phase perform the following tasks:

- Modify the response buffer
- Modify the response XML

Transport

This phase is part of the inflow and outflow. In the outflow, it adds HTTP-specific header elements to the SOAP response message received from the MessageOut and forwards it to the iTP WebServer.

User-defined phases in outflow

You can define phases in the outflow either before or after the MessageOut phase.

The handlers attached to the User-Defined phase can access the response message. Handlers attached to this phase perform the following tasks:

- Modify the response buffer
- Modify the response XML

For example:

```
<phaseOrder type="outflow">
      <phase name="myPhase1"/>
      <phase name="MessageOut"/>
      <phase name="myPhase2"/>
</phaseOrder>
```

where,

myPhase1 and myPhase2
  are user-defined phases.

Handlers

A handler is the smallest unit of invocation within a phase. Handlers are an independent unit of execution that perform specific activities, such as logging a message, consuming addressing headers, and so on.

Phases define the handlers that need to be invoked in each phase and the sequence of invocation based on the sequence defined in the module.xml configuration file.

Modules

A module is a collection of one or more handlers along with their configuration information, such as handler names, the phase in which they must be invoked, and so on. You can use modules to customize end the behavior of the NonStop SOAP 4 server by adding new handlers to the phases.

NOTE: Handlers and modules cannot be used to modify or extend the default behavior of a message receiver.

A module includes the module.xml configuration file that determines the sequence of the handlers in a phase. Each NonStop SOAP 4 module is configured using its module.xml file located in the *<NonStop SOAP 4 Deployment Directory>*/modules/*<module_name>* directory. When configuring a module, you must place the handlers in the pre-defined phase or user-defined phase of the NonStop SOAP 4 server.

The following is a sample module.xml file for a module that includes four handlers:

```
<module name="samplemodule" class="axis2_mod_sam">
    <inflow>
        <handler name="handler1" class="axis2_mod_sam">
            <order phase="Transport"/>
        </handler>
        <handler name="handler2" class="axis2_mod_sam">
            <order phase="PreDispatch"/>
        </handler>
    </inflow>
    <outflow>
        <handler name="handler3" class="axis2_mod_sam">
            <order phase="MessageOut"/>
        </handler>
    </outflow>
    <Outfaultflow>
        <handler name="handler4" class="axis2_mod_sam">
            <order phase="MessageOut"/>
        </handler>
    </Outfaultflow>
</module>
```

NOTE: For more information on the `module.xml` configuration file, see "NonStop SOAP 4 Configuration Files" (page 177).

The following table lists the configuration to achieve the handler deployment in the sample `module.xml` file.

Handler Name	Deployment Flow	Deployment Phase
Handler1	In	`Transport`
Handler2	In	`PreDispatch`
Handler3	Out	`MessageOut`
Handler4	Out	`MessageOut`

Creating a User-Defined Phase

NonStop SOAP 4 allows you to create user-defined phases using the `axis2.xml` configuration file.

You can engage user-defined phases at different stages of the NonStop SOAP 4 message process. The following restrictions are applicable when defining user-defined phases:

- You cannot modify the sequence of the pre-defined phases for `inflow`.
- You can define one or more user-defined phases to the `inflow` but only after the `PostDispatch` phase.
- You must assign a unique name to each user-defined phase.

User-defined phases can be configured in the `inflow` and `outflow` under the `phaseOrder` element of the `axis2.xml` configuration file.

For example, to add:

- `userPhase1` to the `inflow` and
- `userPhase2` and `userPhase3` to the `outflow`

add the highlighted entries to the `phaseOrder` elements in the `axis2.xml` configuration file:

```
<phaseOrder type="inflow">
        <phase name="Transport"/>
        <phase name="PreDispatch"/>
        <phase name="Dispatch"/>
        <phase name="PostDispatch"/>
```

```
        <phase name="userPhase1"/>
</phaseOrder>
<phaseOrder type="outflow">
        <phase name="userPhase2"/>
        <phase name="MessageOut"/>
        <phase name="userPhase3"/>
</phaseOrder>
```

Update the `axis2.xml` configuration file and restart the NonStop SOAP 4 server. A user-defined phase is added to the phase order. You can now attach handlers and modules to the user-defined phase.

Deploying and Attaching a Module

Modules can be attached to a user-defined phase or a pre-defined phase in a NonStop SOAP 4 deployment.

A module is deployed by copying the handler DLLs and the `module.xml` file to the *<NonStop SOAP Deployment Directory>*`/modules/`*<module_ name>* directory. The NonStop SOAP 4 server loads each `module.xml` configuration file during startup and attaches the handlers with the phase defined in the `module.xml` configuration file.

For more information about developing handlers and modules, see "Developing a Sample Module for NonStop SOAP 4" (page 132).

You can attach a module at the global level or service level. When a module is configured at the global level, it will be attached to all the services in a particular NonStop SOAP 4 deployment. To attach a module at the global level, set the `ref` attribute of the module element in the `axis2.xml` configuration file.

When a module is configured at the service level, it will be attached only for a specific service in the NonStop SOAP 4 deployment. To attach a module at the service level, set the `ref` attribute of the module element in the `services.xml` configuration file.

For example, to attach the `transaction` module available in the NonStop SOAP 4 distribution for a particular service in your NonStop SOAP 4 deployment, add the `<module ref="transaction"/>` element under the Service element in the `services.xml` file. Similarly, to attach the `transaction` module at the global level, add the `<module ref="transaction"/>` element under the global module section in the `axis2.xml` file.

During startup, NonStop SOAP 4 will load the transaction handler DLL from the *<NonStop SOAP 4 Deployment Directory>*`/modules/transaction/`*<transaction handler DLL name>*`.so` . The transaction handler will be attached to the phase defined in the *<NonStop SOAP 4 Deployment Directory>*`/modules/transaction/module.xml` configuration file. For more information about services.xml, axis2.xml, and module.xml, see "NonStop SOAP 4 Configuration Files" (page 177).

Developing a Sample Module for NonStop SOAP 4

This section describes the procedure to develop a sample module named `logging` and its handlers: `loggingInHandler` and `loggingOutHandler`.

The `logging` module enables you to log the input and output XML message in a log file.

- The `loggingInHandler` handler is engaged in the `inflow` and logs the request message to a log file.
- The `loggingOutHandler` handler is engaged in the `outflow` and logs the response message to a log file.

  The `logging` module is engaged with the `empdb` service. You must have the `empdb` service deployed in your NonStop SOAP 4 deployment. For more information on deploying the `empdb` service, see "Installing NonStop SOAP" (page 36).

Developing the `logging` module involves the following tasks:

1. "Running the SoapAdminCL Tool to Generate the Module Handler Stub Files" (page 133)
2. "Implementing the Business Logic in the Module Handler Stub Files" (page 134)
3. "Engaging the Module Handler at the Service Level" (page 136)
4. "Verifying the Module Handler Output" (page 137)

Running the SoapAdminCL Tool to Generate the Module Handler Stub Files

The SoapAdminCL tool enables you to generate the module handler stub files and the module.xml file. The SoapAdminCL tool generates the skeleton files for handlers in the inflow and outflow.

To run the SoapAdminCL tool, complete the following steps:

1. Add the directory that includes the SoapAdminCL executable image to the OSS PATH variable.

   ```
   OSS>export PATH=<NonStop SOAP 4 Installation Directory>/tools:$PATH
   ```

   For example:

   ```
   OSS>export PATH=/usr/tandem/nssoap/t0865h01/tools:$PATH
   ```

2. Run the SoapAdminCL tool to generate the module handler stub files:

   ```
   OSS> SoapAdminCL -o <output directory>
                    -mod <module_name>
   ```

   where,

   <output directory>
   specifies the output directory where the module handler stub files will be generated.

   <module_name>
   specifies the name of the generated module.

   For example:

   ```
   OSS> SoapAdminCL –o /home/user1/mynssoap –mod logging
   ```

3. Verify that the following directory structure is created in the *<output directory>*:

   ```
   modules
     /mod_logging
        module.xml
        /src
          mod_logging.h
          mod_logging.c
          logging_in_handler.c
          logging_out_handler.c
          Makefile
   ```

   where,

   module.xml
   is the configuration file for the logging module.

   mod_logging.h
   is the header file for the mod_logging.c source.

   mod_logging.c
   is the C stub file that implements the interface between the NonStop SOAP 4 server and the logging module.

   logging_in_handler.c
   is the C stub file where you must implement the loggingInHandler business logic for the handler. For the logging module, the business logic is designed to log the input XML message.

`logging_out_handler.c`
is the C stub file where you must implement the `loggingOutHandler` business logic for the handler. For the `logging` module, the business logic is designed to log the output XML message.

`Makefile`
is the Makefile for the `logging` module.

NOTE: If the specified directory is not created with all the listed files, repeat Step 1 through Step 3. Do not attempt to manually create the directory structure or the files in it.

Implementing the Business Logic in the Module Handler Stub Files

After generating the module handler stub files using the `SoapAdminCL` tool, you must implement the business logic in the `invoke` methods as described in:

- "Implementing the `invoke()` Method for `loggingInHandler`" (page 134)
- "Implementing the `invoke()` Method for `loggingOutHandler`" (page 134)

Implementing the `invoke()` Method for `loggingInHandler`

Each handler in a module has an `invoke()` method associated with it. The application logic of the handler must be coded in this method:

```
axis2_status_t AXIS2_CALL axutil_logging_in_handler_invoke
               (axis2_handler_t *handler,
               const axis2_env_t *env,
               struct axis2_msg_ctx *msg_ctx);
```

where,

`axis2_handler_t *handler`
is the address of a handler object.

`const axis2_env_t *env`
is the address of an environment structure.

`struct axis2_msg_ctx *msg_ctx`
is the address of a message context structure. The message context structure includes information about the XML message and the message context structure.

Implement the following business logic for the `logging` module:

1. Get the SOAP envelope from the message context structure.
2. Get the SOAP message body from the SOAP envelope.
3. Get the root node of the SOAP body.
4. Get the XML message by serializing the root node.
5. Log the XML message to the log file.

For the sample source code for `invoke()` method of `loggingInHandler`, see the `<NonStop SOAP 4 Installation Directory>/sample_services/modules/mod_logging/src/logging_in_handler.c` file.

Implementing the `invoke()` Method for `loggingOutHandler`

Each handler in a module has an `invoke()` method associated with it. The application logic of the handler must be coded in this method:

```
axis2_status_t AXIS2_CALL axutil_logging_out_handler_invoke(
    struct axis2_handler * handler,
    const axutil_env_t * env,
    struct axis2_msg_ctx * msg_ctx)
```

where,

`axis2_handler *handler`
is the address of a handler object.

`const axutil_env_t *env`
is the address of the environment structure.

`struct axis2_msg_ctx *msg_ctx`
is the address of a message context structure. The message context structure includes information about the XML message and the message context.

Implement the following business logic for the `logging` module:

1. Get the SOAP envelope from the message context structure.
2. Get the SOAP message body from the SOAP envelope.
3. Get the root node of the SOAP body.
4. Get the XML message by serializing the root node.
5. Log the XML message to the log file.

For the sample source code for `invoke()` method of `loggingOutHandler`, see the `<NonStop SOAP 4 Installation Directory >/sample_services/modules/mod_logging/src/logging_out_handler.c` file.

The `logging_in_handler.c` and `logging_out_handler.c` files contain the complete implementation of the handlers and are located in the `<NonStop SOAP 4 Installation Directory>/sample_services/modules/mod_logging/src` directory.

To build your `logging` module, complete the following steps:

1. Copy the `logging_in_handler.c` and `logging_out_handler.c` files to the `<NonStop SOAP 4 Deployment Directory>/modules/logging/src` directory.
2. Set the environment variable AXIS2C_HOME to `<NonStop SOAP 4 Deployment Directory>` using the OSS command:

   ```
   OSS> export AXIS2C_HOME=<NonStop SOAP 4 Deployment Directory>
   ```

3. Build the `logging` module using the following command:

   ```
   OSS>cd <NonStop SOAP 4 Deployment Directory>/modules/logging/src
   OSS>make
   ```

   The `make` command builds the `libaxis2_mod_logging.so` DLL.

   NOTE: NonStop SOAP 4 obtains the name of the DLL file by adding `lib` to the class name and adding the `.so` file extension. For example, if the class name of the module is `axis2_logging`, the DLL name must be `libaxis2_mod_logging.so`.

4. Edit the `module.xml` file as follows:
   - Set the value of the `phase` attribute of the `order` element in the `inflow` to `PostDispatch`. The `loggingInHandler` is invoked in the `PostDispatch` phase.
   - Set the value of the `phase` attribute of the `order` element in `outflow` to `MessageOut`. The `loggingOutHandler` is invoked in the `MessageOut` phase.

```
<module name="mod_logging" class="axis2_mod_logging">
        <inflow>
                <handler name="loggingInHandler" class="axis2_mod_logging">
                        <order phase="PostDispatch"/>
                </handler>
        </inflow>
        <outflow>
                <handler name="loggingOutHandler" class="axis2_mod_logging">
                        <order phase="MessageOut"/>
                </handler>
        </outflow>
</module>
```

NOTE: If you encounter an error while building the `logging` module, repeat the steps listed in "Running the SoapAdminCL Tool to Generate the Module Handler Stub Files" (page 133) and "Implementing the Business Logic in the Module Handler Stub Files" (page 134).

Engaging the Module Handler at the Service Level

After the DLL for the `libaxis2_mod_logging.so` module is built, you must engage the DLL with NonStop SOAP 4. The module can be engaged in NonStop SOAP 4 only if the `module.xml` and DLL files are present in *<NonStop SOAP 4 Deployment Directory>*.

NOTE: If you do not generate the module handler stub files in *<NonStop SOAP 4 Deployment Directory>*, you must:

1. Create the *<NonStop SOAP 4 Deployment Directory>*/modules/*<module_name>* directory.
2. Copy the `module.xml` file and the generated DLL file in the *<NonStop SOAP 4 Deployment Directory>*/modules/*<module_name>* directory. For information about generating DLL files, see "Implementing the Business Logic in the Module Handler Stub Files" (page 134).

If the `module.xml` and the `libaxis2_mod_logging.so` DLL files are not present in *<NonStop SOAP 4 Deployment Directory>*/modules/mod_logging, complete the following steps:

1. Create the *<module_name>* directory in the *<NonStop SOAP 4 Deployment Directory>*/modules directory.

   ```
   OSS> mkdir <NonStop SOAP 4 deployment location>/modules/mod_logging
   ```

2. Copy the `libaxis2_mod_logging.so` DLL file and the `module.xml` file from the *<output directory>* to the *<NonStop SOAP 4 Deployment Directory>*.

   ```
   OSS> cp <output directory>/modules/mod_logging/libaxis2_mod_logging.so <NonStop SOAP 4 deployment
   location>/modules/mod_logging/.

   OSS> cp <output directory>/modules/mod_logging/module.xml <NonStop SOAP 4 deployment
   location>/modules/mod_logging/.
   ```

   The `module.xml` file and the `libaxis2_mod_logging.so` DLL file is now present in the *<NonStop SOAP 4 Deployment Directory>*.

To engage the module handler at the service level, add the following entry to the `services.xml` file located in the *<NonStop SOAP 4 Deployment Directory>*/services/empdb directory:

```
<module ref ="mod_logging"/>
```

The value of the `ref` attribute of the module element must refer to the name of a directory in the modules folder of the NonStop SOAP 4 deployment location.

NOTE: You can engage the module at the global level, which makes the module intercept messages for every service. To engage the module at the global level, add the following entry in the `axis2.xml` file.

```
<module ref ="mod_logging"/>
```

Verifying the Module Handler Output

To verify the module handler output, complete the following steps:

1. Remove the log files using the following OSS command:

   ```
   OSS> cd <NonStop SOAP 4 Deployment Directory>/logs
   OSS> rm -rf *
   ```

2. Set the `LOG_MODE` environment variable to `4` in the `itp_axis2.config` file to enable the debug level log setting:

   ```
   Server  $Axis2c {
           CWD $AXIS2_DEPLOYMENT_ROOT/bin
           Env AXIS2C_HOME=$AXIS2_DEPLOYMENT_ROOT
           Env LOG_MODE=4
   ```

3. Restart the iTP WebServer on which NonStop SOAP 4 is deployed, using the following command:

   ```
   OSS> ./<iTP WebServer Deployment Directory>/conf/restart
   ```

4. Send a request to the `empdb` service using the `empdb` HTML client.

   The `empdb` HTML client can be accessed using the following Web address:

   ```
   http://<ip address>:<port>/<url_pattern>/client/empdb/SoapPW_EmpInfo.html
   ```

   where,

   `<ip address>:<port>`
   is the location of the iTP WebServer where NonStop SOAP 4 is deployed.

   `url_pattern`
   is the pattern string specified while installing NonStop SOAP 4.

5. Verify that the following output is present in the `soaperror.log` file to confirm that the module handler is attached to the service:

   ```
   Input Message: <SOAP request XML>
   Output Message: <SOAP response XML>
   ```

Customizing the NonStop SOAP 4 Message Process Using Message Receiver User Functions

The Message Receiver phase of NonStop SOAP 4 uses the information in the `services.xml` file to convert the incoming SOAP message into a Pathsend buffer and send it to the `$RECEIVE` queue of the designated server class. Message Receiver User Functions (MRUFs) serve the purpose of accessing and modifying data that are not part of the SOAP message. This includes the destination `PATHMON` and the server class name or the destination process name. MRUFs also have access to the Pathsend buffer and only MRUFs can be attached to the Message Receiver phase of NonStop SOAP 4.

Customizing the NonStop SOAP 4 message process using Message Receiver User Functions, involves the following tasks:

1. "Configuring NonStop SOAP 4 Message Receiver User Functions" (page 138)
2. "Modifying the Data Buffer Passed to the Service using NonStop SOAP 4 Message Receiver User Functions" (page 139)
3. "Modifying the Pathway or Process Attributes using NonStop SOAP 4 Message Receiver User Functions" (page 143)

4. "Modifying the Message Flow in the Pathway Message Receiver using NonStop SOAP 4 Message Receiver User Functions" (page 147)

Configuring NonStop SOAP 4 Message Receiver User Functions

Message Receiver User Functions can be configured in NonStop SOAP 4 at the service level and at the global level.

This section includes the following topics:

- "Configuring NonStop SOAP 4 Message Receiver User Functions at the service level" (page 138)
- "Configuring NonStop SOAP 4 Message Receiver User Functions at the global level" (page 138)

Configuring NonStop SOAP 4 Message Receiver User Functions at the service level

To configure Message Receiver User Functions at the service level, complete the following steps:

1. Add the parameter name `MessageReceiverUserFunctions` under the `service` element in the `services.xml` configuration file. The value of the `MessageReceiverUserFunctions` parameter must be set to the DLL (`.so` file) that implements the Message Receiver User Functions:

   ```
   <parameter name="MessageReceiverUserFunctions">[Message Receiver User Functions DLL]</parameter>
   ```

   For example:

   ```
   <parameter name="MessageReceiverUserFunctions">axis2_MRUF_empdb</parameter>
   ```

   NOTE: The `parameter` element that defines the `MessageReceiverUserFunctions` must always be written on a single line.

2. Place the DLL that implements the Message Receiver User Functions in the *<NonStop SOAP 4 Deployment Directory>*`/MRUserFunctions` directory.

   NonStop SOAP 4 generates the name of the DLL by adding `lib` as the prefix and `.so` as the suffix to the Message Receiver User Functions DLL class defined in the `services.xml` configuration file.

   For example:

   If `axis2_MRUF_empdb` is the Message Receiver User Functions class name, the name of the DLL that implements the Message Receiver User Functions must be `libaxis2_MRUF_empdb.so`.

   NOTE: Message Receiver User Functions are different from the handlers that are used to customize the default NonStop SOAP 4 phases or user-defined phases, in the following ways:

   - Message Receiver User Functions are called from the Message Receiver.
   - Message Receiver User Functions do not have a module configuration file.
   - Message Receiver User Functions DLLs must be placed in the *<NonStop SOAP 4 Deployment Directory>*`/MRUserFunctions` directory. Handlers must be placed in the *<NonStop SOAP 4 Deployment Directory>*`/Modules/`*<module_name>* directory.

Configuring NonStop SOAP 4 Message Receiver User Functions at the global level

To configure Message Receiver User Functions at the global level, complete the following steps:

1. Add the parameter name `MessageReceiverUserFunctions` under the `axisconfig` element in the `axis2.xml` file. The value of the `MessageReceiverUserFunctions` parameter must be set to the DLL (`.so` file) class that implements the Message Receiver User Functions:

   ```
   <parameter name="MessageReceiverUserFunctions">[Message Receiver User Functions DLL]</parameter>
   ```

   For example:

   ```
   <parameter name="MessageReceiverUserFunctions">axis2_MRUF_empdb</parameter>
   ```

2. Place the DLL that implements the Message Receiver User Functions in the *<NonStop SOAP Deployment Directory>*`/MRUserFunctions` directory.

   NonStop SOAP 4 generates the name of the DLL by adding `lib` as the prefix and `.so` as the suffix to the Message Receiver User Functions DLL class defined in the `axis2.xml` configuration file.

NOTE: If Message Receiver User Functions are configured both at the service level and the global level, NonStop SOAP 4 calls the Message Receiver User Functions at the global level before the Message Receiver User Functions at the service level. For example, if the global Message Receiver User Functions change the `PATHMON` name from `$pmon1` to `$pmon2`, the service level Message Receiver User Functions will receive the `PATHMON` value as `$pmon2`, not `$pmon1`.

Modifying the Data Buffer Passed to the Service using NonStop SOAP 4 Message Receiver User Functions

Message Receiver User Functions enable you to modify the data buffer before it is passed to the Pathway or NonStop process-based service.

Modifying the data buffer involves the following steps:

- "Setting the `pre_service` and `pre_marshal` Message Receiver User Functions" (page 139)
- "Implementing the `pre_service` and `pre_marshal` Message Receiver User Functions" (page 141)

Setting the `pre_service` and `pre_marshal` Message Receiver User Functions

After loading the Message Receiver User Function DLL, the Pathway Message Receiver invokes the `get_instance()` function in the Message Receiver User Function.

The following tasks are performed in the `get_instance()` function that is implemented as a part of the Message Receiver User Function:

1. "Creating an instance of the Message Receiver User Function structure" (page 139)
2. "Setting `pre_service` and `pre_marshal` function names" (page 140)

NOTE: The `SoapAdminCL` tool generates the stub files along with integration functions, such as `get_instance()`, `create()`, and `remove_instance()`. It also sets the `pre_service` and `pre_marshal` Message Receiver User Functions names and generates the skeleton code for these functions. You must implement the business logic in the `pre_service` and `pre_marshal` Message Receiver User Functions skeleton code generated by the `SoapAdminCL` tool.

Creating an instance of the Message Receiver User Function structure

You must invoke the `MessageReceiverUserFunctions_create()` API from the `axis2_get_instance()` function to create an instance of the Message Receiver User Functions structure call.

Synopsis:

```
AXIS2_EXPORT axis2_MessageReciverUserFunctions_t *MessageReceiverUserFunctions_create (
    const axutil_env_t * env)
```

where,

env
: is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

Return Values:

Returns a pointer to the axis2_MessageReviverUserFunction_t structure. If an error occurs, it returns NULL.

Setting pre_service and pre_marshal function names

After you create a pointer of the Message Receiver User Functions structure, the callback functions that implement the pre_service and pre_marshal functions must be set in the Message Receiver User Functions structure pointer.

This section includes the following topics:

- "Setting the name of the pre_service callback function in the Message Receiver User Functions structure" (page 140)
- "Setting the name of the pre_marshal callback function in the Message Receiver User Functions" (page 140)

Setting the name of the pre_service callback function in the Message Receiver User Functions structure

To set the name of the pre_service callback function in the Message Receiver User Functions structure, invoke the axis2_msg_recv_set_pre_service_function() API from the get_instance() function.

Synopsis:

```
AXIS2_EXPORT axis2_status_t axis2_msg_recv_set_pre_service_function (
    struct axis2_MessageReceiverUserFunctions *inst,
    const axutil_env_t * env
    PRE_SERVICE_FUNCTION func)
```

Parameters:

inst
: is an input parameter and is a pointer to the Message Receiver User Functions structure created using the MessageReceiverUserFunctions_create() function.

env
: is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

func
: is an input parameter that holds the name of function that implements the pre_service functionality.

Return Values:

- AXIS2_SUCCESS on success.
- AXIS2_FAILURE on failure.

Setting the name of the pre_marshal callback function in the Message Receiver User Functions

To set the name of the pre_marshal callback function in the Message Receiver User Functions structure, invoke the axis2_msg_recv_set_pre_marshal_function() API from the get_instance() function.

Synopsis:

```
AXIS2_EXPORT axis2_status_t axis2_msg_recv_set_pre_marshal_function (
    struct axis2_MessageReceiverUserFunctions *inst,
```

```
    const axutil_env_t * env
    PRE_MARSHAL_FUNCTION func)
```

Parameters:

`inst`

is an input parameter and is a pointer to the Message Receiver User Functions structure created using the `MessageReceiverUserFunctions_create()` function.

`env`

is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`func`

is an input parameter that holds the name of function that implements the `pre_marshal` functionality.

Return Values:

- `AXIS2_SUCCESS` on success.
- `AXIS2_FAILURE` on failure.

Implementing the `pre_service` and `pre_marshal` Message Receiver User Functions

The `pre_service` and `pre_marshal` user functions, implemented in the Message Receiver User Functions, can be used to modify the request and response buffers, and Pathway and NonStop process-specific parameters, such as `PATHMON` name, `Process` name, and `serverclass` name. The `pre_service` and `pre_marshal` functions also enable you to modify the default Pathway message receiver flow by setting the `skip_service` and `skip_marshal` flags.

The `pre_service` Message Receiver User Function

The `pre_service` Message Receiver User Function is called after the request buffer is created and before the request buffer is sent to the Pathway or NonStop process-based service.

You can use the `pre_service` Message Receiver User Function to change the following parameters:

- Request message buffer
- `PATHMON` and server class name (for Pathway-based services)
- `Process name` (for process-based service)

You can modify the default process of NonStop SOAP 4 using the following flags:

- Skip service invocation
- Skip response creation

NOTE: For more information on customizing the NonStop SOAP 4 default message process, see "Modifying the Message Flow in the Pathway Message Receiver using NonStop SOAP 4 Message Receiver User Functions" (page 147).

Synopsis:

```
int <service name>_pre_service_function(
                  axis2_MessageReceiverUserFunctions_t *MessageReceiverUserFunctions,
                  axis2_char_t * payload,
                  const axutil_env_t * env,
                  int ReqLen,
                  struct axis2_msg_ctx * in_msg_ctx,
                  struct axis2_msg_ctx * out_msg_ctx)
```

Parameters:

`MessageReceiverUserFunctions`
is an input parameter and is a pointer to the address of the `axis2_MessageReceiverUserFunctions_t` structure.

`payload`
is an input parameter and is a pointer to the request message buffer.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`ReqLen`
is an input parameter and is a pointer to the request length of the buffer.

`in_msg_ctx`
is an input parameter and is a pointer to the message context structure for the `inflow`.

`out_msg_ctx`
is an input parameter and is a pointer to the message context structure for the `outflow`.

Return value:

Returns the modified length of the request message buffer as `int`.

If you want to modify the request buffer in the `pre_service` function, you must modify the length of the request buffer (`ReqLen`) before the returning statement of the `pre_service` function.

NOTE: For more information on message context structure, see the `msg_ctx.h` header file located in the *<NonStop SOAP Installation Directory>*`/include` directory.

The `pre_marshal` Message Receiver User Function

The `pre_marshal` Message Receiver User Function is called after the response is received from the Pathway or NonStop process-based service. You can use the `pre_marshal` Message Receiver User Function to modify the response message buffer and to skip the response creation.

In the `pre_marshal` Message Receiver User Function, you can access the service name and the operation name to perform the service-specific processing and/or operation-specific processing.

Synopsis:

```
int <service name>_pre_marshal_function(
                    axis2_MessageReceiverUserFunctions_t *MessageReceiverUserFunctions,
                    axis2_char_t * payload,
                   const axutil_env_t * env,
                   int ResLen,
                   struct axis2_msg_ctx * in_msg_ctx,
                   struct axis2_msg_ctx * out_msg_ctx)
```

Parameters:

`MessageReceiverUserFunctions`
is an input parameter and is a pointer to the address of the `axis2_MessageReceiverUserFunctions_t` structure.

`payload`
is an input parameter and is a pointer to the response message buffer.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`ResLen`
is an input parameter and is a pointer to the response length of the payload.

`in_msg_ctx`
is an input parameter and is a pointer to the message context structure for `inflow`.

`out_msg_ctx`
is an input parameter and is a pointer to the message context structure for `outflow`.

Return value:

Returns the length of the response message buffer as `int`.

If you are modifying the response buffer in the `pre_marshal` function, you must modify the length of the response buffer (`ResLen`) before the returning statement of the `pre_marshal` function.

Modifying the Pathway or Process Attributes using NonStop SOAP 4 Message Receiver User Functions

The `axis2_MessageReceiverUserFunctions_t` structure provides the following functions that can be used to modify the Pathway or process attributes:

- "The `axis2_msg_recv_get_pathmonName()` Function" (page 143)
- "The `axis2_msg_recv_get_serverclassName()` Function" (page 143)
- "The `axis2_msg_recv_get_serviceName()` Function" (page 144)
- "The `axis2_msg_recv_get_operationName()` Function" (page 144)
- "The `axis2_msg_recv_get_processName()` Function" (page 145)
- "The `axis2_msg_recv_set_pathmonName()` Function" (page 145)
- "The `axis2_msg_recv_set_serverclassName()` Function" (page 145)
- "The `axis2_msg_recv_set_serviceName()` Function" (page 146)
- "The `axis2_msg_recv_set_operationName()` Function" (page 146)
- "The `axis2_msg_recv_set_processName()` Function" (page 147)

The `axis2_msg_recv_get_pathmonName()` Function

The `axis2_msg_recv_get_pathmonName()` function returns the value of the PATHMON name for the service.

Synopsis:

```
AXIS2_EXPORT axis2_char_t *AXIS2_CALL
axis2_msg_recv_get_pathmonName(
axis2_MessageReceiverUserFunctions_t * MessageReceiverUserFunctions,
const axutil_env_t * env)
```

Parameters:

`MessageReceiverUserFunctions`
is an input parameter and is a pointer to the address of the `axis2_MessageReceiverUserFunctions_t` structure.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return Value:

Returns the value of the `pathmon` name as an `axis2_char_t` pointer.

The `axis2_msg_recv_get_serverclassName()` Function

The `axis2_msg_recv_get_serverclass()` function returns the value of the server class name for the service.

Synopsis:

```
AXIS2_EXPORT axis2_char_t *AXIS2_CALL
axis2_msg_recv_get_serverclassName(
    axis2_MessageReceiverUserFunctions_t * MessageReceiverUserFunctions,
    const axutil_env_t * env)
```

Parameters:

`MessageReceiverUserFunctions`
is an input parameter and is a pointer to the address of the `axis2_MessageReceiverUserFunctions_t` structure.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return Value:

Returns the value of the server class name as an `axis2_char_t` pointer.

The `axis2_msg_recv_get_serviceName()` Function

The `axis2_msg_recv_get_serviceName()` function returns the value of the service name for the service.

Synopsis:

```
AXIS2_EXPORT axis2_char_t *AXIS2_CALL
axis2_msg_recv_get_serviceName(
    axis2_MessageReceiverUserFunctions_t * MessageReceiverUserFunctions,
    const axutil_env_t * env)
```

Parameters:

`MessageReceiverUserFunctions`
is an input parameter and is a pointer to the address of the `axis2_MessageReceiverUserFunctions_t` structure.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return Value:

Returns the value of the service name as an `axis2_char_t` pointer.

The `axis2_msg_recv_get_operationName()` Function

The `axis2_msg_recv_get_operationName()` function returns the value of the operation name for the service.

Synopsis:

```
AXIS2_EXPORT axis2_char_t *AXIS2_CALL
axis2_msg_recv_get_ operationName (
    axis2_MessageReceiverUserFunctions_t * MessageReceiverUserFunctions,
    const axutil_env_t * env)
```

Parameters:

`MessageReceiverUserFunctions`
is an input parameter and is a pointer to the address of the `axis2_MessageReceiverUserFunctions_t` structure.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return Value:

Returns the value of the operation name as an `axis2_char_t` pointer.

The `axis2_msg_recv_get_processName()` Function

The `axis2_msg_recv_get_processName()` function returns the value of the `process` name for the service.

Synopsis:

```
AXIS2_EXPORT axis2_char_t *AXIS2_CALL
axis2_msg_recv_get_processName(
axis2_MessageReceiverUserFunctions_t * MessageReceiverUserFunctions,
const axutil_env_t * env)
```

Parameters:

`MessageReceiverUserFunctions`
: is an input parameter and is a pointer to the address of the `axis2_MessageReceiverUserFunctions_t` structure.

`env`
: is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return Value:

Returns the value of the `process` name as an `axis2_char_t` pointer.

The `axis2_msg_recv_set_pathmonName()` Function

The `axis2_msg_recv_set_pathmonName()` function sets the `pathmon` name for the service.

Synopsis:

```
AXIS2_EXPORT axis2_status_t AXIS2_CALL
axis2_msg_recv_set_pathmonName (
axis2_MessageReceiverUserFunctions_t * MessageReceiverUserFunctions,
const axutil_env_t * env,
const axis2_char_t * pathmon)
```

Parameters:

`MessageReceiverUserFunctions`
: is an input parameter and is a pointer to the address of the `axis2_MessageReceiverUserFunctions_t` structure.

`env`
: is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`pathmon`
: is the `pathmon` you want to set for the service.

Return Value:

- `AXIS2_SUCCESS` on success
- `AXIS2_FAILURE` on failure

The `axis2_msg_recv_set_serverclassName()` Function

The `axis2_msg_recv_set_serverclass()` function sets the `serverclass` name for the service.

Synopsis:

```
AXIS2_EXPORT axis2_status_t AXIS2_CALL
axis2_msg_recv_set_serverclassName (
    axis2_MessageReceiverUserFunctions_t * MessageReceiverUserFunctions,
```

```
    const axutil_env_t * env,
    const axis2_char_t * serverclass)
```

Parameters:

`MessageReceiverUserFunctions`
is an input parameter and is a pointer to the address of the `axis2_MessageReceiverUserFunctions_t` structure.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`serverclass`
is the name of the server class you want to set for the service.

Return Value:

- `AXIS2_SUCCESS` on success
- `AXIS2_FAILURE` on failure

The `axis2_msg_recv_set_serviceName()` Function

The `axis2_msg_recv_set_serviceName()` function sets the service name for the service.

Synopsis:

```
AXIS2_EXPORT axis2_status_t AXIS2_CALL
axis2_msg_recv_set_serviceName (
    axis2_MessageReceiverUserFunctions_t * MessageReceiverUserFunctions,
    const axutil_env_t * env,
    const axis2_char_t * serviceName)
```

Parameters:

`MessageReceiverUserFunctions`
is an input parameter and is a pointer to the address of the `axis2_MessageReceiverUserFunctions_t` structure.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`serviceName`
is the service name you want to set for the service.

Return Value:

- `AXIS2_SUCCESS` on success
- `AXIS2_FAILURE` on failure

The `axis2_msg_recv_set_operationName()` Function

The `axis2_msg_recv_set_operationName()` function sets the operation name for the service.

Synopsis:

```
AXIS2_EXPORT axis2_status_t AXIS2_CALL
axis2_msg_recv_set_operationName (
    axis2_MessageReceiverUserFunctions_t * MessageReceiverUserFunctions,
    const axutil_env_t * env,
    const axis2_char_t * operationName)
```

Parameters:

`MessageReceiverUserFunctions`
is an input parameter and is a pointer to the address of the `axis2_MessageReceiverUserFunctions_t` structure.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`operationName`
is the operation name you want to set for the service.

Return Value:

- `AXIS2_SUCCESS` on success
- `AXIS2_FAILURE` on failure

The `axis2_msg_recv_set_processName()` Function

The `axis2_msg_recv_set_processName()` function sets the `process` name for the service.

Synopsis:

```
AXIS2_EXPORT axis2_status_t AXIS2_CALL
axis2_msg_recv_set processName (
axis2_MessageReceiverUserFunctions_t * MessageReceiverUserFunctions,
const axutil_env_t * env,
const axis2_char_t * process)
```

Parameters:

`MessageReceiverUserFunctions`
is an input parameter and is a pointer to the address of the `axis2_MessageReceiverUserFunctions_t` structure.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`process`
is the `process` you want to set for the service.

Return Value:

- `AXIS2_SUCCESS` on success
- `AXIS2_FAILURE` on failure

Modifying the Message Flow in the Pathway Message Receiver using NonStop SOAP 4 Message Receiver User Functions

The `axis2_MessageReceiverUserFunctions_t` structure provides the following functions to modify the message flow in the Pathway Message Receiver:

- "The `axis2_msg_recv_get_skipService()` Function" (page 147)
- "The `axis2_msg_recv_get_skipMarshal()` Function" (page 148)
- "The `axis2_msg_recv_set_skipService()` Function" (page 148)
- "The `axis2_msg_recv_set_skipMarshal()` Function" (page 149)

The `axis2_msg_recv_get_skipService()` Function

The `axis2_msg_recv_get_skipService()` function returns the value of the `skipService` flag.

Synopsis:

```
AXIS2_EXPORT axis2_bool_t AXIS2_CALL
axis2_msg_recv_get_skipService(
    axis2_MessageReceiverUserFunctions_t * MessageReceiverUserFunctions,
    const axutil_env_t * env)
```

Parameters:

`MessageReceiverUserFunctions`
is an input parameter and is a pointer to the address of the `axis2_MessageReceiverUserFunctions_t` structure.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return Value:

Returns the value of the `skipService` flag.

- `AXIS2_TRUE` if the `skipService` flag is set
- `AXIS2_FALSE` if the `skipService` flag is not set

NOTE: When the `skipService` flag is set, NonStop SOAP 4 will not perform the service invocation. To skip the service invocation, this flag must be set to `AXIS2_TRUE`.

The `axis2_msg_recv_get_skipMarshal()` Function

The `axis2_msg_recv_get_skipMarshal()` function returns the value of the `skipMarshal` flag.

Synopsis:

```
AXIS2_EXPORT axis2_bool_t AXIS2_CALL
axis2_msg_recv_get_ skipMarshal (
    axis2_MessageReceiverUserFunctions_t * MessageReceiverUserFunctions,
    const axutil_env_t * env)
```

Parameters:

`MessageReceiverUserFunctions`
is an input parameter and is a pointer to the address of the `axis2_MessageReceiverUserFunctions_t` structure.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return Value:

Returns the value of the `skipMarshal` flag.

- `AXIS2_TRUE` if the `skipMarshal` flag is set
- `AXIS2_FALSE` if the `skipMarshal` flag is not set

NOTE: When the `skipMarshal` flag is set, NonStop SOAP 4 will not perform the service invocation. To skip the service invocation, this flag must be set to `AXIS2_TRUE`.

The `axis2_msg_recv_set_skipService()` Function

The `axis2_msg_recv_set_skipService()` function sets the value of the `skipService` flag.

Synopsis:

```
AXIS2_EXPORT axis2_status_t AXIS2_CALL
axis2_msg_recv_set_skipService(
    axis2_MessageReceiverUserFunctions_t * MessageReceiverUserFunctions,
    const axutil_env_t * env,
     const axis2_bool_t * skipService )
```

Parameters:

`MessageReceiverUserFunctions`
is an input parameter and is a pointer to the address of the `axis2_MessageReceiverUserFunctions_t` structure.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`skipService`
is a flag that skips the service.

Return Value:

- `AXIS2_SUCCESS` on success
- `AXIS2_FAILURE` on failure

The `axis2_msg_recv_set_skipMarshal()` Function

The `axis2_msg_recv_set_skipMarshal()` function sets the value of the `skipMarshal` flag.

If the `skipMarshal` flag is set to `AXIS2_TRUE`, the NonStop SOAP 4 server will not generate the response XML from the response buffer. To modify the response XML, you can skip the NonStop SOAP 4 marshalling and write your own logic to generate the response XML in the `pre_marshal` function.

Synopsis:

```
AXIS2_EXPORT axis2_status_t AXIS2_CALL
axis2_msg_recv_set_skipMarshal (
    axis2_MessageReceiverUserFunctions_t * MessageReceiverUserFunctions,
    const axutil_env_t * env,
    const axis2_bool_t * skipMarshal)
```

Parameters:

`MessageReceiverUserFunctions`
is an input parameter and is a pointer to the address of the `axis2_MessageReceiverUserFunctions_t` structure.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`skipMarshal`
is a flag that skips the marshalling (response XML generation) process.

Return Value:

- `AXIS2_SUCCESS` on success
- `AXIS2_FAILURE` on failure

Developing Sample Message Receiver User Functions for NonStop SOAP 4

NonStop SOAP 4 Message Receiver User Functions use the C programming language to modify the message process in the NonStop SOAP 4 server.

Developing the Message Receiver User Functions involves the following tasks:

1. "Running the SoapAdminCL Tool to Generate the Message Receiver User Functions Stub Files" (page 150)
2. "Implementing the Business Logic in the Message Receiver User Functions Stub Files" (page 150)
3. "Engaging the Message Receiver User Functions at the Service Level" (page 151)
4. "Verifying the Message Receiver User Functions Output" (page 152)

Running the SoapAdminCL Tool to Generate the Message Receiver User Functions Stub Files

To run the SoapAdminCL tool to generate the Message Receiver User Functions stub files, complete the following steps:

1. Add the directory that includes the SoapAdminCL executable image to the OSS PATH variable.

   ```
   OSS>export PATH=<NonStop SOAP 4 Installation Directory>/tools:$PATH
   ```

   For example:

   ```
   OSS>export PATH=/usr/tandem/nssoap/t0865h01/tools:$PATH
   ```

2. Run the SoapAdminCL tool to generate the Message Receiver User Functions stub files:

   ```
   OSS> SoapAdminCL -MRUF <MRUF_name> -o <output directory>
   ```

   where,

   <output directory>
   specifies the output directory where the Message Receiver User Functions stub files will be generated.

   <MRUF_name>
   specifies the name of the generated Message Receiver User Functions.

   For example:

   ```
   OSS> SoapAdminCL –MRUF logging –o /home/usr/mynssoap
   ```

3. Verify that the following directory structure is created in the *<output directory>*:

   ```
   MRUserFunctions
       /logging
         /MRUF_src
           logging_MRUF.h
           logging_MRUF.c
           Makefile
   ```

   where,

   logging_MRUF.h
   is the header file for the Message Receiver User Functions.

   logging_MRUF.c
   is the C stub file where you can implement the business logic for the pre_service and pre_marshal Message Receiver User Functions.

   Makefile
   is the Makefile to build the Message Receiver User Functions DLL.

Implementing the Business Logic in the Message Receiver User Functions Stub Files

After generating the Message Receiver User Functions stub files using the SoapAdminCL tool, you must implement the business logic in the pre_service and pre_marshal Message Receiver User Functions as described in:

- "Implementing the pre_service Message Receiver User Function" (page 151)
- "Implementing the pre_marshal Message Receiver User Function" (page 151)

Implementing the pre_service Message Receiver User Function

For the implementation of business logic for the pre_service Message Receiver User Function, see the sample code in the <NonStop SOAP 4 Installation Directory>/sample_services/modules/logging_MRUF/src/logging_MRUF.c file.

Implementing the pre_marshal Message Receiver User Function

For the implementation of business logic for the pre_marshal Message Receiver User Function, see the sample code in the <NonStop SOAP 4 Installation Directory>/sample_services/modules/logging_MRUF/src/logging_MRUF.c file.

The logging_MRUF.c file contains the source code and is located in <NonStop SOAP 4 Installation Directory>/sample_services/modules/logging_MRUF/src.

To build your logging module, complete the following steps:

1. Copy the MRUserFunctions_logging directory to the <NonStop SOAP 4 Deployment Directory> directory.

   ```
   OSS>cp -rf
       <NonStop SOAP 4 Installation Directory>/sample_services/modules/logging_MRUF/src
       <NonStop SOAP 4 Deployment Directory>/MRUserFunctions/logging/MRUF_src
   ```

2. Set the environment variable AXIS2C_HOME to <NonStop SOAP 4 Deployment Directory> using the OSS command:

   ```
   OSS> export AXIS2C_HOME=<NonStop SOAP 4 Deployment Directory>
   ```

3. Run the following OSS command to build the logging module DLL:

   ```
   OSS>cd <NonStop SOAP 4 Deployment Directory>/MRUserFunctions/logging/MRUF_src

   OSS>make
   ```

   The make command builds the libaxis2_MRUF_logging.so DLL and places it in the <NonStop SOAP 4 Deployment Directory>/MRUserFunctions directory.

   NOTE: If you encounter an error while building the logging module, repeat the steps listed in "Running the SoapAdminCL Tool to Generate the Message Receiver User Functions Stub Files" (page 150) and "Implementing the Business Logic in the Module Handler Stub Files" (page 134).

Engaging the Message Receiver User Functions at the Service Level

After the DLL Message Receiver User Functions (libaxis2_MRUF_logging.so) are built, you must engage the DLL with NonStop SOAP 4.

The Message Receiver User Functions can be engaged in NonStop SOAP 4 only if the libaxis2_MRUF_logging.so DLL file is present in the <NonStop SOAP 4 Deployment Location>/MRUserFunctions.

NOTE: If you do not generate the Message Receiver User Functions stub files in the <NonStop SOAP 4 Deployment Directory>, you must copy the generated DLL file in the <NonStop SOAP 4 Deployment Directory>/MRUserFunctions directory. For information about generating DLL files, see "Implementing the Business Logic in the Message Receiver User Functions Stub Files" (page 150).

Copy the libaxis2_MRUF_logging.so file in the <NonStop SOAP 4 Deployment Directory>/MRUserFunctions directory, using the following command:

```
OSS> cp <output directory>/MRUserFunctions/libaxis2_MRUF_logging.so <NonStop SOAP 4 deployment location>/MRUserFunctions/.
```

To engage the Message Receiver User Functions at the service level, add the following entry in the services.xml file located in the <NonStop SOAP 4 Deployment Directory>/services/empdb directory:

```
<parameter name="MessageReceiverUserFunctions">axis2_MRUF_logging</parameter>
```

where,

`axis2_MRUF_logging`
is the name of the DLL containing the Message Receiver User Functions.

NOTE: You can also engage the module at the global level, which makes the module intercept messages for every service. To engage the module at the global level, add the following entry in the `axis2.xml` file.

```
<parameter name="MessageReceiverUserFunctions">axis2_MRUF_logging</parameter>
```

Verifying the Message Receiver User Functions Output

To verify the Message Receiver User Functions output, complete the following steps:

1. Remove the log files using the following OSS command:

   ```
   OSS> cd <NonStop SOAP 4 Deployment Directory>/logs
   OSS> rm -rf *
   ```

2. Set the `LOG_MODE` environment variable to `4` in the `itp_axis2.config` file to enable the debug level log setting.

   ```
   Server  $Axis2c {
           CWD $AXIS2_DEPLOYMENT_ROOT/bin
           Env AXIS2C_HOME=$AXIS2_DEPLOYMENT_ROOT
           Env LOG_MODE=4
   ```

3. Restart the iTP WebServer on which NonStop SOAP 4 is deployed, using the following command:

   ```
   OSS> <iTP WebServer Deployment Directory>/conf/restart
   ```

4. Send a request to the `empdb` service using the `empdb` HTML client.

   The `empdb` HTML client can be accessed using the following Web address:

   ```
   http://<ip address>:<port>/<url_pattern>/client/empdb/SoapPW_EmpInfo.html
   ```

   where,

   `<ip address>:<port>`
   is the location of the iTP WebServer in which NonStop SOAP 4 is deployed.

   `url_pattern`
   is the pattern string specified while installing NonStop SOAP 4.

5. Verify that the following output is present in the `soaperror.log` file to confirm that the Message Receiver User Functions are attached to the service:

   ```
   Input buffer: <request buffer>
   Output Buffer: <response buffer>
   ```

NonStop SOAP 4 is now customized to log input and output buffers using MRUFs.

8 NonStop SOAP 4 Service Description Language

The NonStop SOAP 4 SDL file is an XML file that describes the Web services deployed in NonStop SOAP 4. The SDL file along with the data definition language file for the Web service is used as an input by the `SoapAdminCL` tool to generate the WSDL file, HTML clients, XML schema files, SOAP request and response XML files, and the `services.xml` file for NonStop SOAP 4 services. NonStop SOAP 4 does not use the SDL file during runtime.

You can generate an SDL file for NonStop SOAP 4 services by:

- Using the NonStop SOAP 3 service definition repository file with the NonStop SOAP 4 `SoapAdminCL` tool
- Copying and updating the SDL files distributed with the NonStop SOAP 4 distribution
- Using the NonStop SOAP 4 Administration Utility

This chapter describes the SDL file and its elements:

- "SDL File Elements and Attributes" (page 153)
- "SDL Service Types" (page 154)

SDL File Elements and Attributes

The SDL file describes the target service to be deployed in NonStop SOAP 4. The hierarchy of the elements in the SDL document type definition (DTD) file is as follows:

```
<sdl>
  <Pathway | Process | ServerAPI>
    <PathwayEnvironment | ProcessEnvironment>
       <Service | ProcessDetails>
          <Operation>
             <OperationDescription>
             <RequestInfo>
                <DDLDefinitionName>
             <ResponseInfo>
                <DDLDefinitionName>
                <ResponseSelection>
```

NOTE: All attribute values are case-sensitive. Use these values as detailed in this manual.

The following sections describe the SDL file elements and their attributes.

The `sdl` Element

The `sdl` element is the base element of the SDL file. This element is mandatory and its definition appears in the SDL file as:

```
<sdl Url="/<url_pattern>"  ServerAddress="http://<ip address>:<port>">
```

The `sdl` element has the following attributes:

- ServerAddress
- Url

The `ServerAddress` attribute

The `ServerAddress` attribute specifies the location (the protocol, ip address, and port) of the iTP WebServer where the NonStop SOAP 4 server is deployed. The default location is http://www.nonstopsoap.com.

For example:

SDL file:

```
ServerAddress = "http://www.nonstopsoap.com"
```

Generated WSDL file:

```
<soap:address location="http://www.nonstopsoap.com/<url_pattern>/
                        services/<service_name>"/>
```

The Url attribute

The optional Url attribute specifies the Web address of the NonStop SOAP 4 deployment location. The value of this attribute is the *<url_pattern>* or the Filemap value specified in the itp_axis2.config file. The default value of this attribute is nssoap.

For example:

SDL file:

```
Url = "/nssoap"
```

Generated HTML file:

```
ACTION =/nssoap/services/<service_name>
```

Generated WSDL file:

```
<soap:address location="http://<ip address>:<port>/
                        nssoap/services/<service_name>"/>
```

SDL Service Types

The child element of the sdl element describes the service type. NonStop SOAP 4 supports the following service types:

- "The Pathway Element" (page 154)
- "The Process Element" (page 174)
- "The ServerAPI Element" (page 176)

The SoapAdminCL tool, when run with the import option, checks the name of the child element of the sdl element and generates the HTML files, WSDL files, XML schema files, SOAP request and response XML files, and the services.xml file.

The Pathway Element

The Pathway element indicates that the service is a Pathway application. The SDL file uses the Pathway element to define services in multiple Pathway environments. The Pathway element definition appears in the SDL file as:

```
<Pathway>
```

The Pathway element has a child element named PathwayEnvironment.

The PathwayEnvironment Element

The PathwayEnvironment element specifies the Pathway environment of the target service. This element is mandatory for the Pathway application. The element definition appears in the SDL file as follows:

```
<PathwayEnvironment PathmonName="$<PATHMON Name>">
```

The PathwayEnvironment element includes the PathmonName attribute.

The PathmonName attribute

The PathmonName attribute specifies the name of the PATHMON under which the target service is running as a serverclass. The value of this element is stored in the services.xml file. The NonStop SOAP 4 server uses this attribute value during runtime.

For example:

SDL file:

```
PathmonName="$PMON"
```

Generated `services.xml` file:

```
<parameter name="pathmon">$PMON</parameter>
```

The `PathwayEnvironment` has a child element named `Service`.

The `Service` Element

The `Service` element includes information about the services to be deployed in the NonStop SOAP 4 server. The element definition appears in the SDL file as:

```
<Service
        Name="<service_name>"
        ServerClassName="<ServerClassName>"
        DDLDictionaryLocation="$<volumename>.<subvolumename>"
        language="[C | COBOL]"
        SrvrEncoding="UTF-8"
        GenerateSessionHeader="[yes | no]"
        GenerateTransactionHeader="[yes | no]"
        GenerateModuleHandlerFiles="[yes | no]"
        GenerateMessageReceiverUserFunctionsFiles="[yes | no]"
        stringTermination="[NonNull | NullTerminated]">
```

The `Service` element includes the following attributes:

- `Name`
- `ServerClassName`
- `DDLDictionaryLocation`
- `language`
- `SrvrEncoding`
- `GenerateSessionHeader`
- `GenerateTransactionHeader`
- `GenerateModuleHandlerFiles`
- `GenerateMessageReceiverUserFunctionsFiles`
- `stringTermination`

The `Service` element contains a child element named `Operation`. For more information about the `Operation` element, see "The Operation Element" (page).

The `Service` element has the following attributes:

`Name`

The `Name` attribute specifies the name of the target service to deploy in NonStop SOAP 4. The name is translated to the target service name for NonStop SOAP 4 and the value is the endpoint of the target service. The `Name` attribute can take alphanumeric characters and underscores (_). The `Name` attribute is mandatory and the value must be unique for a NonStop SOAP 4 deployment.

Using the value of this attribute, the `SoapAdminCL` tool generates the directory structure in the client and services directory of the NonStop SOAP 4 deployment location.

NOTE: The `Name` attribute is included in NonStop SOAP 4 to support multiple services pointing to the same `serverclass` in a single NonStop SOAP 4 deployment.

For example:

SDL file:

```
Name="MyService"

The generated directory structure is as follows:
```

```
<NonStop SOAP 4 Deployment Directory>
  Client
    MyService
      SoapPW_<OperationName>.html
  Services
    MyService
      SoapPW_MyService.wsdl
      services.xml
```

Generated WSDL file:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MyService-interface"
  targetNamespace="urn:MyService-interface"
  xmlns:def="urn:MyService-interface"
  .
  .>
.
.
<portType
   name="portMyService">
.
.
</portType>
<binding
   name="MyServiceBinding"
   type="def:portMyService">
.
.
</binding>
<service_name="MyServiceService">
  <port name="portMyService"
        binding="def:MyServiceBinding">
    <soap:address location="http://www.nonstopsoap.com/nssoap/services/MyService"/>
  </port>
</service>
</definitions>
```

Generated `services.xml` file:

```
<service_name="MyService">

<parameter name="ServiceClass" locked="xsd:false">MyService</parameter>
<parameter name="wsdl_path">
<NonStop SOAP deployment dir>/services/reflector/SoapPW_MyService.wsdl
</parameter>
.
.
</service>
```

`ServerClassName`

The `ServerClassName` attribute specifies the `serverclass` name running in `PATHMON`. The value of this element is stored in the `services.xml` file for the NonStop SOAP 4 server to use during runtime. This is a mandatory attribute for the Pathway application service.

For example:

SDL file:

```
ServerClassName="MYSERVERCLASS"
```

Generated `services.xml` file:

```
<parameter name="serverclass">MYSERVERCLASS</parameter>
```

`DDLDictionaryLocation`

The `DDLDictionaryLocation` attribute specifies the Guardian location of the DDL dictionary files. The `SoapAdminCL` tool uses the DDL definitions from the specified location to generate elements for the HTML clients and the XML schema definitions for XSD files and WSDL files. This is a mandatory attribute for all types of services.

For example:

SDL file:

```
DDLDictionaryLocation="$volume.subvolume"
```

NOTE: The DDL dictionary includes the SOAP request and response structures for the particular service. If the dictionary is not available at the specified location, the `SoapAdminCL` tool reports the following error and ends the operation.

```
SOAPADMIN ERROR >> Error generating files for <request/response> of service <service_name>
```

`language`

The `language` attribute specifies the language in which the service is written. The following languages are available in NonStop SOAP 4:

- `COBOL`
- `C`

For example:

SDL file:

```
language="COBOL"
```

Generated `services.xml` file:

```
<parameter name="language">COBOL</parameter>
```

`SrvrEncoding`

The `SrvrEncoding` attribute specifies the character encoding used by the target `PATHWAY` server. The default value is `UTF-8`. This attribute is optional.

For example:

SDL file:

```
SrvrEncoding=”UTF-16“
```

Generated WSDL file:

```
<xsd:complexType name="InEncodingType">
<xsd:element name="InEncoding">
<xsd:complexType>
<xsd:attribute name="InputEncoding" default="UTF-16" type="xsd:string" use="optional" />
</xsd:complexType>
</xsd:element>
</xsd:complexType>
```

`GenerateSessionHeader`

The `GenerateSessionHeader` attribute specifies that the WSDL file schema should be generated with the `Session` element definition in the SOAP header namespace.

For more information about the `Session` element, see “Transaction Management” (page 236).

For example:

SDL file:

```
GenerateSessionHeader="yes"
```

Generated WSDL file:

```
<xsd:schema
     elementFormDefault="qualified"
     targetNamespace="urn:compaq_nsk_oss_SoapHeader"
     xmlns="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Session">
  <xsd:complexType>
         <xsd:attribute use="optional"
                        name="SessionID"
                        type="xsd:string" />
         <xsd:attribute use="optional"
                        name="BeginNewTransaction"
                        type="xsd:string" />
         <xsd:attribute use="optional"
                        name="CurrentTransactionCommand"
```

```
                    type="xsd:string" />
          <xsd:attribute use="optional"
                    name="SessionCommand"
                    type="xsd:string" />
          <xsd:attribute use="optional"
                    name="Subsession"
                    type="xsd:string" />
```

`GenerateTransactionHeader`

The `GenerateTransactionHeader` attribute specifies that the WSDL file schema should be generated with the `Transaction` element definition in the SOAP header namespace.

For more information about the `Transaction` element, see "Transaction Management" (page 236) .

For example:

SDL file:

```
GenerateTransactionHeader="yes"
```

Generated WSDL file:

```
<xsd:schema
     elementFormDefault="qualified"
     targetNamespace="urn:compaq_nsk_oss_SoapHeader"
     xmlns="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Transaction">
  <xsd:complexType>
         <xsd:attribute use="optional"
                    name="Command"
                    type="xsd:string" />
         <xsd:attribute use="optional"
                    name="TransactionID"
                    type="xsd:string" />
         <xsd:attribute use="optional"
                    name="Timeout"
                    type="xsd:string" />
  </xsd:complexType>
  </xsd:element>
```

NOTE: When the `GenerateSessionHeader` and the `GenerateTransactionHeader` attributes are set to `yes`, the `SoapAdminCL` tool returns the following error:

```
SOAPADMIN ERROR >> Error in definition of service "<service_name>".
Both attributes "GenerateSessionHeader" and "GenerateTransactionHeader" cannot have a value "yes"
```

`GenerateModuleHandlerFiles`

The `GenerateModuleHandlerFiles` attribute specifies whether the module handler files must be generated. The default value is `no`. This attribute is optional.

For example:

SDL file:

```
GenerateModuleHandlerFiles ="yes"
```

Generated `services.xml` file:

```
<!-- uncomment the following parameter to engage the module -->
<!-- module ref="mod_<service_name>"/-->
```

Generated directory structure:

```
    <NonStop SOAP 4 Deployment Directory>
      /modules
        /mod_<service_name>
          module.xml
          /src
            Makefile
```

```
mod_<service_name>.h
mod_<service_name>.c
<service_name>_pre_process_handler.c
<service_name>_post_process_handler.c
```

GenerateMessageReceiverUserFunctionsFiles

The GenerateMessageReceiverUserFunctionsFiles attribute specifies whether the Message Receiver User Functions skeletons stubs must be generated. The default value is no. This attribute is optional.

For example:

SDL file:

```
GenerateMessageReceiverUserFunctionsFiles="yes"
```

Generated services.xml file:

```
<!-- uncomment the following parameter to engage Message Receiver User Functions -->
<!-- parameter name="MessageReceiverUserFunctions">axis2_MRUF_<service_name></parameter -->
```

Generated directory structure:

```
<NonStop SOAP 4 Deployment Directory>
 /services
  /<service_name>
   /MRUserFunction_src
    Makefile
    <service_name>_MRUF.c
    <service_name>_MRUF.h
```

stringTermination

The stringTermination attribute denotes the type of string that the target Pathway server expects. It specifies whether the string is NullTerminated or NonNull. The value of this element is stored in the services.xml file for the NonStop SOAP 4 server to use during runtime. The default value is NullTerminated. This attribute is optional.

For example:

SDL file:

```
stringTermination="NullTerminated"
```

Generated services.xml file:

```
<parameter name="stringTermination">NullTerminated</parameter>
```

The Operation Element

The Operation element describes the operations supported by each service. In NonStop SOAP 4, the SoapAdminCL tool translates this element into operations in the WSDL file. The SoapAdminCL tool generates HTML files for each of these operations. An SDL file may contain only one operation of a given name, even if the operations apply to different services.

NOTE: When the -m|-migration option is used, the SoapAdminCL tool generates a separate WSDL file for each operation in the WSDL directory.

Following is a sample Operation element definition in the SDL file:

```
<Operation
            Name="<operation Name>"
            TMFTransactionSupport="[Required | Supports | Never]"
            AbortTransactionOnFault="[yes | no]"
            NameSpaceQualified="[yes | no]"
            SoapMessageType="[document | rpc]"
            ProcessSoapDDLComments="[yes | no]"
            SoapDDLAttribute="[yes | no]"
            OutputFileNamePrefix="<Character string value>"
            UseDDLDefaultValue="[yes | no]"
            EnableOutputSensitive="[yes | no]"
            RspEncoding="UTF-8"
```

The Operation element includes the following attributes:

- Name
- TMFTransactionSupport
- AbortTransactionOnFault
- NameSpaceQualified
- SoapMessageType
- ProcessSoapDDLComments
- SoapDDLAttribute
- OutputFileNamePrefix
- UseDDLDefaultValue
- EnableOutputSensitive
- RspEncoding

The Operation element includes the following attributes:

Name

The Name attribute specifies the name of the operation. The value of this attribute should be unique in a single NonStop SOAP 4 deployment. This is a mandatory attribute and can have alphanumeric characters and an underscore (_).

The value of this element is reflected in the WSDL file and the services.xml file.

For example:

SDL file:

```
Name="MyOperation"
```

Generated directory structure:

```
<NonStop SOAP 4 Deployment Directory>
      Client
          MyService
             SoapPW_MyOperation.html
             wsdl
                 SoapPW_MyOperation.wsdl
      Services
          MyService
             SoapPW_MyService.wsdl
             services.xml
```

Generated WSDL file:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
        .
        .
  xmlns:tns1="urn:cpq_tns_MyOperation"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<types>
        .
        .
<xsd:schema
     elementFormDefault="qualified"
     targetNamespace="urn:cpq_tns_MyOperation"
     xmlns="http://www.w3.org/2001/XMLSchema">

<xsd:complexType name="MyOperationResponse0">
 <xsd:sequence>
  <xsd:element name="request" minOccurs="1" maxOccurs="1">
   <xsd:simpleType>
```

```
    <xsd:restriction base="xsd:string">
     <xsd:maxLength value="12"/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MyOperation">
 <xsd:sequence>
  <xsd:element name="request" minOccurs="1" maxOccurs="1">
   <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:maxLength value="12"/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
</xsd:schema>
</types>

<message name="outMyOperationResponse0">
  <part name="MyOperationResponse0"
     type="tns1:MyOperationResponse0"/>
</message>

<message name="inMyOperation">
  <part name="MyOperation"
     type="tns1:MyOperation"/>
</message>
   .
   .

<portType ...>

<operation
   name="MyOperation">

    <input
      name="inMyOperation"
      message="def:inMyOperation"/>
    <output
      name="outMyOperationResponse0"
      message="def:outMyOperationResponse0"/>
  </operation>

</portType>

<binding ...>
  .
  .
    <operation
     name="MyOperation">

    <soap:operation
      soapAction="MyOperation"/>

    <input name="inMyOperation">
   .
 .
    </input>
    <output name="outMyOperationResponse0">
 .
 .
```

```
    </output>
   </operation>
 </binding>
  .
  .
</definitions>
```

Generated `services.xml` file:

```
<service ...>
  .
  .
        <operation name="MyOperation">
     .
     .
        </operation>
</service>
```

`TMFTransactionSupport`

The `TMFTransactionSupport` attribute specifies if the service will interact with the TMF subsystem. This attribute has three values: `Required`, `Supports`, and `Never`. Based on these values, the NonStop SOAP 4 server decides on the state of the transaction before invoking the service. This value is reflected in the `services.xml` file.

If you set the value of the `TMFTransactionSupport` attribute to `Required`, the service will always be accessed under a transaction. If the client does not explicitly start a transaction, the NonStop SOAP 4 server will start a transaction, invoke the service, and end the transaction.

For example:

SDL file:

```
TMFTransactionSupport ="Required"
```

Generated `services.xml` file:

```
<parameter name="TMFTransactionSupport">Required</parameter>
```

If you set the value of the attribute to `Supports`, the service can be accessed with or without a transaction. If the client starts a transaction, NonStop SOAP 4 will start a transaction. If the client does not start a transaction, NonStop SOAP 4 will serve the request but will not start a transaction.

For example:

SDL file:

```
TMFTransactionSupport ="Supports"
```

Generated `services.xml` file:

```
<parameter name="TMFTransactionSupport">Supports</parameter>
```

If you set the value of the attribute to `Never`, the NonStop SOAP 4 server will not support a transaction. If the client starts a transaction, the NonStop SOAP 4 server will return an error.

For example:

SDL file:

```
TMFTransactionSupport ="Never"
```

Generated `services.xml` file:

```
<parameter name="TMFTransactionSupport">Never</parameter>
```

The default value is `Never`. This attribute is optional.

`AbortTransactionOnFault`

The `AbortTransactionOnFault` attribute specifies that the associated TMF transaction must be aborted if the NonStop Soap 4 server generates a fault. The default value is `no`. This attribute is optional.

For example:

SDL file:

```
AbortTransactionOnFault ="yes"
```

Generated `services.xml` file:

```
<parameter name="AbortTransactionOnFault">yes</parameter>
```

`NameSpaceQualified`

The `NameSpaceQualified` attribute specifies whether the element names in the WSDL file generated must be namespace qualified. The default value is `no`. This attribute is optional.

For example:

SDL file: `NameSpaceQualified ="yes"`	SDL file: `NameSpaceQualified ="no"`
Generated WSDL file: `<xsd:schema elementFormDefault="qualified" targetNamespace="urn:cpq_tns_MyOperation" xmlns="http://www.w3.org/2001/XMLSchema">`	Generated WSDL file: `<xsd:schema targetNamespace="urn:cpq_tns_MyOperation" xmlns="http://www.w3.org/2001/XMLSchema">`

`SoapMessageType`

The `SoapMessageType` attribute specifies whether the operation is remote procedure call (RPC) oriented (messages containing parameters and return values) or document-oriented (messages containing documents). When the value of the attribute is `rpc`, the `OperationName` element (the immediate child of the `Operation` element) must contain only one immediate child element. If the `SoapMessageType` attribute is not specified, its value is assumed to be `document`. This attribute is optional.

For example:

SDL file: `SoapMessageType="rpc"`	SDL file: `SoapMessageType="document"`
Schema in the generated WSDL file: `<xsd:complexType name="MyOperationResponse0"> <xsd:sequence> <xsd:element name="request" minOccurs="1" maxOccurs="1"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="12"/> </xsd:restriction> </xsd:simpleType> </xsd:sequence> </xsd:complexType>`	Schema in the generated WSDL file: `<xsd:element name="MyOperationResponse0" type="tns1:MyOperationResponse0"/> <xsd:complexType name="MyOperationResponse0"> <xsd:sequence> <xsd:element name="request" minOccurs="1" maxOccurs="1"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="12"/> </xsd:restriction> </xsd:simpleType> </xsd:element> </xsd:sequence> </xsd:complexType>`
Message definition in the generated WSDL file: `<message name="inMyOperation"> <part name="MyOperation" type="tns1:MyOperation"/> </message>`	Message definition in the generated WSDL file: `<message name="inMyOperation"> <part name="MyOperation" element="tns1:MyOperation"/> </message>`
Binding style in the generated WSDL file:	Binding style in the generated WSDL file:

`<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>`	`<soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>`

In the `services.xml` file, the following parameter is added to support different SOAP message types:

```
<parameter name="SoapMessageType">document</parameter>
```

The value of this parameter can be either `document` or `rpc` based on the value in SDL.

NOTE: If `SOAPMessageType` is set to `rpc`, NonStop SOAP 4 does not validate the incoming request with schema using its validation module, and even if it is engaged it will be skipped.

`ProcessSoapDDLComments`

The `ProcessSoapDDLComments` option processes the comments in the DDL files. If the `ProcessSoapDDLComments` option is set to `yes`, the `SoapAdminCL` tool processes the comments in the DDL files and takes the appropriate action while generating the files. The default value is `no`.

NOTE: To enable DDL comments, you must specify `?comments` in the DDL file.

Table 4 lists the various actions the `SoapAdminCL` tool performs for different DDL comment values.

Table 4 The DDL Comments values

DDL Comment	Description

`@SOAP_OPTIONAL`	This specifies an element as optional. It sets the `minOccurs` value of the element as 0 and `maxOccurs` value to 1 in the generated WSDL file.
	For example:
	DDL file:
	`* @SOAP_OPTIONAL`
	`input type binary 32.`
	WSDL file translation:
	`<input type="xsd:long" minOccurs="0" maxOccurs="1">`
	`</input>`

`@SOAP_SUPPRESS_IN`	This specifies that a field is optional in the SOAP request message because the DDL field might be a group field or a leaf field. This comment is activated only when it appears in combination with `@SOAP_OPTIONAL`. In the WSDL file, this comment is translated as `minOccurs=0` and `maxOccurs` depending on the selected DDL comment.
	For example:
	DDL file:
	`* @SOAP_OPTIONAL @SOAP_SUPPRESS_IN`
	`20 Bin3 type binary 32.`
	WSDL file translation:
	`<xsd:element name="bin3" type="xsd:int" minOccurs="0" maxOccurs="1" />`

`@SOAP_SUPPRESS_OUT`	This specifies that a field is optional in the SOAP response message because the DDL field might be a group field or a leaf field. This comment is activated only when it appears in combination with `@SOAP_OPTIONAL`. In the WSDL file, this comment is translated as `minOccurs=0` and `maxOccurs` depending on the selected DDL comment. For example: DDL file:

Table 4 The DDL Comments values *(continued)*

DDL Comment	Description
	* @**SOAP_OPTIONAL** @**SOAP_SUPPRESS_OUT** input type binary 32. WSDL file translation: <xsd:element name="input" type="xsd:int" minOccurs="0" maxOccurs="1" />
@SOAP_SUPPRESS_INOUT	This specifies that a field is optional in the SOAP request and the SOAP response message because the DDL field might be a group field or a leaf field. This comment is activated only when it appears in combination with @SOAP_OPTIONAL. In the WSDL file, this comment is translated as minOccurs=0 and maxOccurs depending on the selected DDL comment. For example: DDL file: * @**SOAP_OPTIONAL** @**SOAP_SUPPRESS_INOUT** input type binary 32 WSDL file translation: <xsd:element name="input" type="xsd:int" minOccurs="0" maxOccurs="1" />
@SOAP_OCCURS_DEP_ON	The SoapAdminCL tool adds the DDL comment in the generated WSDL file and the HTML client files to indicate if the occurrences of the current field are dependent on the value of the field mentioned in the comment. This comment must appear with the element that has more than one Occurs value. In the SOAP request, the element must appear as many times as the value of the dependent element occurs at runtime. For example: DDL file: * @**SOAP_OCCURS_DEP_ON Bin2** 20 Bin3 type binary 32 OCCURS 5 times WSDL file translation: <!-- The following element occurs 0 to 5 times, depending upon the value of the element: bin2:urn:cpq_tns_reflector--> <xsd:element name="bin3" type="xsd:int" minOccurs="0" maxOccurs="5" /> **NOTE:** For the OCCURS .. DEPENDING ON .. clause, the SoapAdminCL tool generates the following output: DDL file: 20 Bin3 type binary 32 OCCURS 2 to 5 times DEPENDING ON Bin2 WSDL file translation: <!-- The following element occurs 2 to 5 times, depending upon the value of the element: bin2:urn:cpq_tns_reflector--> <xsd:element name="bin3" type="xsd:int" minOccurs="2" maxOccurs="5" /> You must define the depending on element one level above where it is used in the DDL. To overcome this limitation, use the @SOAP_OCCURS_DEP_ON comment tag.

Table 4 The DDL Comments values *(continued)*

DDL Comment	Description
@SOAP_BASE64	This generates the <xsd:base64binary> element in the WSDL file. For example: DDL file: `* @SOAP_BASE64 15 CH2 type character 13` WSDL file translation: `<!-- The following xsd:base64Binary type, represents a xsd:type="xsd:string" type --> <xsd:element name="ch2base64Binary" type="xsd:base64Binary" minOccurs="1" maxOccurs="1"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="13" /> </xsd:restriction> </xsd:simpleType> </xsd:element>`
@SOAP_ELEMENT	This marks the field as an element in the WSDL file. All the DDL fields appear as elements in the WSDL file. If the SoapDDLAttribute value is set to yes, this comment forces the particular field to appear as an element. For example: DDL file: `* @SOAP_ELEMENT 10 mystr TYPE character 10` WSDL file translation: `<xsd:sequence> <xsd:element name="mystr" minOccurs="1" maxOccurs="1"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="10"/> </xsd:restriction> </xsd:simpleType> </xsd:element> </xsd:sequence>`
@SOAP_ATTRIBUTE	This exposes the DDL field as an attribute in the WSDL file. For example: DDL file: `* @SOAP_ATTRIBUTE 10 mystr TYPE character 10` WSDL file translation: `<xsd:attribute name="mystr" use="required"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="10"/> </xsd:restriction> </xsd:simpleType> </xsd:attribute>`

`SoapDDLAttribute`

The `SoapDDLAttribute` specifies whether all the leaf fields of the request and response DDL definitions of the particular service must be represented as XML attributes in the generated WSDL file. The specified value is `yes` or `no`; the default value is `no`. This attribute is optional.

For example:

SDL file: `SoapDDLAttribute ="yes"`	SDL file: `SoapDDLAttribute ="no"`

Generated schema in the WSDL file:
`<xsd:complexType name="request">`
`<xsd:attribute name="req1"`
`use="required">`
`<xsd:simpleType>`
`<xsd:restriction base="xsd:string">`
`<xsd:maxLength value="5"/>`
`</xsd:restriction>`
`</xsd:simpleType>`
`</xsd:attribute>`
`<xsd:attribute name="req2"`
`use="required">`
`<xsd:simpleType>`
`<xsd:restriction base="xsd:string">`
`<xsd:maxLength value="5"/>`
`</xsd:restriction>`
`</xsd:simpleType>`
`</xsd:attribute>`
`<xsd:attribute name="req3"`
`use="required">`
`<xsd:simpleType>`
`<xsd:restriction base="xsd:string">`
`<xsd:maxLength value="2"/>`
`</xsd:restriction>`
`</xsd:simpleType>`
`</xsd:attribute>`
`</xsd:complexType>`
	Generated schema in the WSDL file:
	`<xsd:complexType name="request">`
	`<xsd:sequence>`
	`<xsd:element name="req1"`
	`minOccurs="1" maxOccurs="1">`
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:string">`
	`<xsd:maxLength value="5"/>`
	`</xsd:restriction>`
	`</xsd:simpleType>`
	`</xsd:element>`
	`<xsd:element name="req2"`
	`minOccurs="1" maxOccurs="1">`
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:string">`
	`<xsd:maxLength value="5"/>`
	`</xsd:restriction>`
	`</xsd:simpleType>`
	`</xsd:element>`
	`<xsd:element name="req3"`
	`minOccurs="1" maxOccurs="1">`
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:string">`
	`<xsd:maxLength value="2"/>`
	`</xsd:restriction>`
	`</xsd:simpleType>`
	`</xsd:element>`
	`</xsd:sequence>`
	`</xsd:complexType>`

`OutputFileNamePrefix`

The `OutputFileNamePrefix` attribute specifies the prefix of the generated files. The default value is "". This attribute is optional and supports alphanumeric characters and an underscore(_).

For example:

SDL file:

`OutputFileNamePrefix = "`**`John`**`_"`

Generated directory structure:

```
<NonStop SOAP 4 Deployment Directory>
   Client
      MyService
         John_SoapPW_MyOperation.html
   Services
      MyService
         John_SoapPW_MyService.wsdl
         services.xml
```

Generated `services.xml` file:

```
<parameter name="wsdl_path"><NonStop SOAP Deployment Directory>
/services/MyService/John_SoapPW_reflector.wsdl</parameter>
```

`UseDDLDefaultValue`

The `UseDDLDefaultValue` attribute specifies whether the default value of the element or attribute must be used. The default value is `no`. This attribute is optional.

For example:

SDL file:	SDL file:

UseDDLDefaultValue="**yes**"	UseDDLDefaultValue="**no**"

DDL File:
`DEFINITION REQ.`
`10 Request.`
`20 req1 type character 5 Value "R".`
`20 req2 type character 5 value "B".`
`20 req3 type character 2 Value "R".`
`END`
	DDL file:
	`DEFINITION REQ.`
	`10 Request.`
	`20 req1 type character 5 Value "R".`
	`20 req2 type character 5 value "B".`
	`20 req3 type character 2 Value "R".`
	`END`

Schema in the generated WSDL file:
`<xsd:complexType name="request">`
`<xsd:sequence>`
`<xsd:element name="req1" default="R" minOccurs="1" maxOccurs="1">`
`<xsd:simpleType>`
`<xsd:restriction base="xsd:string">`
`<xsd:maxLength value="5"/>`
`</xsd:restriction>`
`</xsd:simpleType>`
`</xsd:element>`
`<xsd:element name="req2" default="B" minOccurs="1" maxOccurs="1">`
`<xsd:simpleType>`
`<xsd:restriction base="xsd:string">`
`<xsd:maxLength value="5"/>`
`</xsd:restriction>`
`</xsd:simpleType>`
`</xsd:element>`
`<xsd:element name="req3" default="R" minOccurs="1" maxOccurs="1">`
`<xsd:simpleType>`
`<xsd:restriction base="xsd:string">`
`<xsd:maxLength value="2"/>`
`</xsd:restriction>`
`</xsd:simpleType>`
`</xsd:element>`
`</xsd:sequence>`
`</xsd:complexType>`
	Schema in the generated WSDL file:
	`<xsd:complexType name="request">`
	`<xsd:sequence>`
	`<xsd:element name="req1" minOccurs="1" maxOccurs="1">`
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:string">`
	`<xsd:maxLength value="5"/>`
	`</xsd:restriction>`
	`</xsd:simpleType>`
	`</xsd:element>`
	`<xsd:element name="req2" minOccurs="1" maxOccurs="1">`
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:string">`
	`<xsd:maxLength value="5"/>`
	`</xsd:restriction>`
	`</xsd:simpleType>`
	`</xsd:element>`
	`<xsd:element name="req3" minOccurs="1" maxOccurs="1">`
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:string">`
	`<xsd:maxLength value="2"/>`
	`</xsd:restriction>`
	`</xsd:simpleType>`
	`</xsd:element>`
	`</xsd:sequence>`
	`</xsd:complexType>`

`EnableOutputSensitive`

The `EnableOutputSensitive` attribute determines whether the output files generated by `SoapAdminCL` must be case-sensitive. The default value of the attribute is `no`. Therefore, the output files generated by `SoapAdminCL` would have all the field names in lowercase.

If the `EnableOutputSensitive` attribute is set to `yes` in the SDL file, then:

- In the DDL, only the structure of the DDL definition must be compiled using the `output_sensitive` option and not the definition name.
- The DDL must not be compiled with the `output_sensitive` flag in the command line.

For example:

DDL file:

```
Definition TestN.
?output_sensitive
02 NaMeS.
03 First Pic X(10).
03 Middle Pic X(10).
03 Last Pic X(10).
```

```
?nooutput_sensitive
End.
```

Now, compile the DDL using the following command:

```
gtacl -p ddl < "ddl file-name"
```

CAUTION: The DDL must not be compiled with the `output_sensitive` flag in the command line.

`RspEncoding`

The `RspEncoding` attribute specifies any default output encoding for a Pathway service other than `UTF-8`. The SOAP request and response XML file reflects the value of this attribute. The default value is `UTF-8`. This attribute is optional.

For example:

SDL file:

```
RspEncoding="Shift-JIS"
```

SOAP Message(SOAP Request/Response):

```
<?xml version="1.0" encoding="Shift_JIS"?>
```

The `Operation` element contains the following child elements:

- "The `OperationDescription` Element" (page 169)
- "The `RequestInfo` Element" (page 169)
- "The `ResponseInfo` Element" (page 170)

The `OperationDescription` Element

The `OperationDescription` element specifies the text description of the operation. The maximum size of the text description is 256 bytes. If the length specified is greater than the maximum length, the first 256 bytes are used. This is a mandatory element. The element definition appears in the SDL file as:

```
<OperationDescription>Reflect the Response</OperationDescription>
```

The `RequestInfo` Element

The `RequestInfo` element specifies the SOAP request information for the particular operation. This is a mandatory element. The element definition appears in the SDL file as:

```
<RequestInfo> </RequestInfo>
```

The RequestInfo element has a child element named DDLDefinitionName.

- The DDLDefinitionName Element

  The DDLDefinitionName element specifies the DDL definition of the NonStop SOAP 4 request that is accepted by the Pathway service. This is a mandatory element. The element definition appears in the SDL file as:

  For example:

  DDL file:

```
DEFINITION REQ.
      10 request TYPE character 12.
END
```

  SDL file:

```
<DDLDefinitionName>REQ</DDLDefinitionName>
```

  NOTE: You can use multiple <DDLDefinitionName> element. For more information, see "Support for Multiple DDL Definitions" (page 234)

  Generated WSDL file:

```
<xsd:element name="MyOperation" type="tns1:MyOperation"/>
<xsd:complexType name="MyOperation">
 <xsd:sequence>
  <xsd:element name="request" minOccurs="1" maxOccurs="1">
   <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:maxLength value="12"/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
```

The ResponseInfo Element

The ResponseInfo element specifies the SOAP response information for the particular operation. This is a mandatory element. The element definition appears in the SDL file as:

```
<ResponseInfo> </ResponseInfo>
```

The ResponseInfo element has the following child elements:

- "The DDLDefinitionName Element" (page 170)
- "The ResponseSelection Element" (page 171)

The DDLDefinitionName Element

The DDLDefinitionName element specifies the DDL definition of the NonStop SOAP 4 response that is accepted by the Pathway service. This is a mandatory element. The element definition appears in the SDL file as:

```
DDLDefinitionName>[DDL response definition]</DDLDefinitionName>
```

For example:

DDL file:

```
DEFINITION RES.
     10 response TYPE character 12.
END
```

SDL file:

```
<DDLDefinitionName>RES</DDLDefinitionName>
```

NOTE: You can use multiple `<DDLDefinitionName>` element. For more information about support for multiple DDL elements, see "Support for Multiple DDL Definitions" (page 234)

Generated WSDL file:

```
<xsd:element name="MyOperationResponse0" type="tns1:MyOperationResponse0"/>
<xsd:complexType name="MyOperationResponse0">
 <xsd:sequence>
  <xsd:element name="response" minOccurs="1" maxOccurs="1">
   <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:maxLength value="12"/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
```

The `ResponseSelection` Element

The `ResponseSelection` element specifies the selection criteria of a NonStop SOAP 4 response. This element supports the multiple non-fault response feature.

NOTE: NonStop SOAP supports multiple non-fault messages per request. However, wsdl standard mandates only one non-fault message and maximum one fault message per request. If multiple messages are marked as non-fault messages, the generated wsdl is not parsed by wsdl2c and wsdl2pwy tools and may result in error.

The value of the attributes guide the NonStop SOAP 4 server to select a particular response. The information is stored in the `services.xml` file, which the server uses during runtime. This is a mandatory element. The element definition appears in the SDL file as:

```
<ResponseSelection
       Start="0"
       End="2"
       BufVal="00"
       ConversionOp="[string | decimal]"
       ComparisonOp="[eq | gt | lt | ne]"
       Fault="[yes | no]"
       defaultResp="[yes | no]"/>
```

The `ResponseSelection` element includes the following attributes:

- `Start`
- `End`
- `BufVal`
- `ConversionOp`
- `ComparisonOp`
- `Fault`
- `defaultResp`
-

`Start`

The `Start` attribute specifies the start index of the response buffer for the response selection criteria. There is no default value for this attribute. This attribute is optional.

For example:

SDL file:

```
Start="0"
```

Generated `services.xml` file:

```
<parameter name="Response0_StartIndex">0</parameter>
```

`End`

The `End` attribute specifies the end index of the response buffer for the response selection criteria. There is no default value for this attribute. This attribute is optional. This attribute is mandatory if the `Start` attribute is defined.

For example:

SDL file:

```
End="2"
```

Generated `services.xml` file:

```
<parameter name="Response0_EndIndex">2</parameter>
```

`BufVal`

The `BufVal` attribute specifies the response buffer value that must be checked between the start index and the end index for the response selection criteria (`Response[Start:End-1]`). There is no default value for this attribute. This attribute is optional. It is mandatory if the `Start` and `End` attributes are defined.

For example:

SDL file:

```
BufVal="12"
```

Generated `services.xml` file:

```
<parameter name="Response0_BufferValue">12</parameter>
```

`ConversionOp`

The `ConversionOp` attribute specifies the conversion operation for the response selection criteria. Based on the value of this attribute, the response buffer value is converted to the string or decimal data type and compared with the value of the `BufVal` parameter. Table 5 lists the values for the `ConversionOp` attribute.

Table 5 Conversion Operator Values Table

Value	Effect
string	Converts the response buffer value between the start and end index to string datatype before comparing them for the response selection.
decimal	Converts the response buffer value between the start and end index to decimal datatype before comparing them for the response selection.

For example:

SDL file:

```
ConversionOp="string"
```

Generated `services.xml` file:

```
<parameter name="Response0_ConversionOp">string</parameter>
```

`ComparisonOp`

The `ComparisonOp` attribute specifies the comparison operation for the response selection criteria. Based on the value of this attribute, the response buffer value is compared with the value mentioned in the `BufferValue` parameter of the `services.xml` file. Table 6 lists the values for the `ComparisonOp` attribute.

Table 6 Comparison Operator Values Table

Value	Effect
eq	Response is selected if the Response buffer value between the start and end index matches the value mentioned in the `BufVal` attribute.
gt	Response is selected if the Response buffer value between the start and end index is greater than the value mentioned in the `BufVal` attribute.
lt	Response is selected if the Response buffer value between the start and end index is less than the value mentioned in the `BufVal` attribute.
ne	Response is selected if the Response buffer value between the start and end index does not match the value mentioned in the `BufVal` attribute.

For example:

SDL file:

```
ComparisonOp="eq"
```

Generated `services.xml` file:

```
<parameter name="Response0_ComparisonOp">eq</parameter>
```

`Fault`

The `Fault` attribute specifies the fault response selection criteria. If the comparison is `true` and this attribute value is `yes`, it returns the message as a Fault Message.

For example:

SDL file:

```
Fault="yes"
```

Generated `services.xml` file:

```
<parameter name="Response0_Fault">yes</parameter>
```

`defaultResp`

The `defaultResp` attribute specifies the selection of the default response. If the value of this parameter is set to `yes`, then it overrides the response selection criterion specified for the current element, and selects the current response when no other matching response element is found.

For example:

SDL file:

```
defaultResp="yes"
```

Generated `services.xml` file:

```
<parameter name="Response0_Default">yes</parameter>
```

For example, a sample SDL file for Pathway service will appear as follows:

```
<!--NonStop SOAP 4 SDL file -->
<sdl Url="/axis2c" ServerAddress="http://www.nonstopsoap.com">
  <Pathway>
    <PathwayEnvironment PathmonName="$jae">
      <Service Name="reflector"
         ServerClassName="reflector"
         DDLDictionaryLocation="$FC3.SOL3379"
         SrvrEncoding="UTF-8"
         language="C"
         GenerateSessionHeader="no"
         GenerateTransactionHeader="no"
         GenerateModuleHandlerFiles="no"
         GenerateMessageReceiverUserFunctionsFiles="no"
         stringTermination="NonNull">
        <Operation  Name="REFLECTOR"
           TMFTransactionSupport="Never"
```

```
            AbortTransactionOnFault="no"
            NameSpaceQualified="no"
            SoapMessageType="document"
            ProcessSoapDDLComments="yes"
            SoapDDLAttribute="yes"
            UseDDLDefaultValue="no"
            EnableOutputSensitive="no">
            <OperationDescription>Retrieve user and model profiles</OperationDescription>
            <RequestInfo>
               <DDLDefinitionName>INPUT</DDLDefinitionName>
            </RequestInfo>
            <ResponseInfo>
              <DDLDefinitionName>OUTPUT</DDLDefinitionName>
              <ResponseSelection  Start="0" End="2" BufVal="11"
                ComparisonOp="eq" defaultResp="yes" Fault="no"/>
            </ResponseInfo>
          </Operation>
        </Service>
      </PathwayEnvironment>
    </Pathway>
</sdl>
```

The Process Element

The Process element specifies that the service is running as a process on a NonStop system. The SDL file can define multiple process environments. Multiple environments are separated by the Process element. The element definition appears in the SDL file as:

```
<Process> </Process>
```

The Process element has a child element named ProcessEnvironment.

NOTE: The CreateDDLDefFile attribute of the <Process> element is not included in NonStop SOAP 2.4 and later releases. It appears in the SDL DTD for backward compatibility. The NonStop SOAP 4 SoapAdminCL tool ignores the occurrence of this attribute.

The ProcessEnvironment Element

The ProcessEnvironment element specifies the process environment of the target service. This is a mandatory element for the NonStop process. The element definition appears in the SDL file as:

```
<ProcessEnvironment Name="$<Process Name>">
```

The ProcessEnvironment element contains the <Name> attribute.

Name

The Name attribute specifies the value of the NonStop process name in which the target service is running. The value of this element is stored in the services.xml file for the NonStop SOAP 4 server to use during runtime.

For example:

SDL file:

```
Name="$PROC"
```

Generated services.xml file:

```
<parameter name="process">$PROC</parameter>
```

The ProcessEnvironment has a child element named ProcessDetails.

The ProcessDetails Element

The ProcessDetails element specifies the details of the services to be deployed in the NonStop SOAP 4 server. The element definition appears in the SDL file as:

```
<ProcessDetails
        Name="<Service Name>"
        DDLDictionaryLocation="$<volumename>.<subvolumename>"
        language="[C | COBOL]"
```

```
        SrvrEncoding="UTF-8"
        GenerateSessionHeader="[yes | no]"
        GenerateTransactionHeader="[yes | no]"
        GenerateModuleHandlerFiles="[yes | no]"
        GenerateMessageReceiverUserFunctionsFiles="[yes | no]"
        stringTermination="[NonNull | NullTerminated]">
```

The `ProcessDetails` element includes the following attributes:

- `Name`
- `DDLDictionaryLocation`
- `language`
- `SrvrEncoding`
- `GenerateSessionHeader`
- `GenerateTransactionHeader`
- `GenerateModuleHandlerFiles`
- `GenerateMessageReceiverUserFunctionsFiles`
- `stringTermination`

For example:

```
<sdl>
  <Process>
      <ProcessEnvironment Name="$REFL">
        <ProcessDetails Name="Reflector"
           DDLDictionaryLocation="$FC3.AXTE1"
           SrvrEncoding="UTF-8"
           language="C"
           stringTermination="NullTerminated">
          <Operation Name="REFLECTOR"
             TMFTransactionSupport="Required"
             AbortTransactionOnFault="yes"
             NameSpaceQualified="yes"
             SoapMessageType="document"
             ProcessSoapDDLComments="no"
             SoapDDLAttribute="no"
             UseDDLDefaultValue="no">
             <OperationDescription>Reflector</OperationDescription>
             <RequestInfo>
                <DDLDefinitionName>INPUT</DDLDefinitionName>
             </RequestInfo>
             <ResponseInfo>
               <DDLDefinitionName>OUTPUT</DDLDefinitionName>
               <ResponseSelection defaultResp="yes"/>
             </ResponseInfo>
          </Operation>
        </ProcessDetails>
      </ProcessEnvironment>
  </Process>
</sdl>
```

The `ProcessDetails` element includes the `Operation` element as a child element. For information about the `Operation` element and its attributes, see "The `Service` Element" (page 155).

NOTE: The `ServerClassName` attribute is not applicable for the `ProcessDetails` element. It returns an error when used in the `ProcessDetails` element.

The ServerAPI Element

The ServerAPI element specifies that the service is a server API service. The SDL file can define multiple ServerAPI environments. Multiple environments are separated by the ServerAPI element. The element definition appears in the SDL file as:

```
<ServerAPI>
```

The ServerAPI element has one child element named Service. The ServerAPI element contains attributes that are same as the Service element. Except for the ServerClass attribute in the Service element, the ServerAPI elements support all the attributes in the Service element.

For example:

```
<sdl Url="/nssoap" ServerAddress="http://www.nonstopsoap.com">
    <ServerAPI>
       <Service
           Name="math"
           DDLDictionaryLocation="$DSMSCM.API">
        <Operation
              Name="add"
              NameSpaceQualified="yes"
              SoapMessageType="document"
              ProcessSoapDDLComments="no"
              SoapDDLAttribute="no"
              UseDDLDefaultValue="no">
           <OperationDescription>
                Add two numbers
           </OperationDescription>
           <RequestInfo>
              <DDLDefinitionName>PARAMETER</DDLDefinitionName>
           </RequestInfo>
           <ResponseInfo>
              <DDLDefinitionName>RESULT</DDLDefinitionName>
              <ResponseSelection defaultResp="yes" Fault="no"/>
           </ResponseInfo>
        </Operation>
       </Service>
    </ServerAPI>
</sdl>
```

9 NonStop SOAP 4 Configuration Files

The NonStop SOAP 4 distribution includes configuration files that can be used to configure the NonStop SOAP 4 server, its modules, and the services running under it.

This chapter describes the following NonStop SOAP 4 configuration files:

- "The `itp_axis2.config` File" (page 177)
- "The `axis2.xml` File" (page 180)
- "The `services.xml` File" (page 182)
- "The `module.xml` File" (page 190)
- "Defining the Log File Size" (page 180)
- "Defining Separate Log and Trace Files for NonStop SOAP 4 Servers" (page 180)

The `itp_axis2.config` File

After installing NonStop SOAP 4, run the NonStop SOAP 4 deployment script (*<NonStop SOAP 4 Installation Directory>*`/bin/deploy.sh`) to create the default `itp_axis2.config` file in the *<NonStop SOAP 4 Deployment Directory>*. The configurations set in the `itp_axis2.config` file are loaded by the iTP WebServer during startup.

You must specify the location of the `itp_axis2.config` file in the *<iTP WebServer Deployment directory>*`/conf/local.config` file.

The `itp_axis2.config` file is used to perform the following tasks:

- "Linking iTP WebServer to the NonStop SOAP 4 Deployment" (page 177)
- "Specifying iTP WebServer Filemap for NonStop SOAP 4" (page 177)
- "Defining the Log Levels of the NonStop SOAP 4 Server" (page 178)
- "Defining the Log File Size" (page 180)
- "Defining Separate Log and Trace Files for NonStop SOAP 4 Servers" (page 180)

Linking iTP WebServer to the NonStop SOAP 4 Deployment

The itp_axis2.config file is read by the iTP WebServer during startup. The AXIS2_DEPLOYMENT_ROOT variable is set to the location of the *<NonStop SOAP 4 Deployment Directory>* that is entered while running the NonStop SOAP 4 deployment script, `deploy.sh`.

For information on setting up the location of the NonStop SOAP 4 deployment directory, see "Setting up the Deployment Environment" (page 38).

For example:

```
set AXIS2_DEPLOYMENT_ROOT /home/usr1/t0865h01
```

This command instructs iTP WebServer to run the NonStop SOAP 4 deployment located at `/home/usr1/t0865h01`.

Specifying iTP WebServer Filemap for NonStop SOAP 4

The iTP WebServer identifies the NonStop SOAP 4 requests by locating a unique string specified in the `url_pattern` of the request to a `FileMap` variable. The `FileMap` variable is specified in the `itp_axis2.config` file.

NOTE: You must specify the location of the `itp_axis2.config` file in the `<iTP WebServer Deployment directory>/conf/local.config` file.

The unique string specified as the <*url_pattern*> is the string entered while setting the deployment environment for NonStop SOAP 4. For more information, see Step 6 in "Setting up the Deployment Environment" (page 38).

For example:

`http://<ip address>:<port>/<url_pattern>/services/<service_name>`

where,

`<ip address>:<port>`
: is the IP address and port of iTP WebServer integrated with NonStop SOAP 4.

`<url_pattern>`
: is the value of the `<url_pattern>`. By default, this value is set to `axis2c`.

`services`
: is the name of the directory in `<NonStop SOAP 4 Deployment Directory>` where the services are present.

  NOTE: This directory must be named `services`.

`<service_name>`
: is the name of the service to be accessed.

For information about deploying additional SOAP servers in a given iTP WebServer, see "Setting up Multiple NonStop SOAP Deployment Instances in a Single iTP WebServer" (page 43).

Defining the Log Levels of the NonStop SOAP 4 Server

NonStop SOAP 4 supports seven logging levels that define the extent of information to be logged in the NonStop SOAP 4 log files. The logging level is specified by setting the `LOG_MODE` OSS environment variable in the `itp_axis2.config` file.

The NonStop SOAP 4 log files (`soaperror.log`) are located at `<NonStop SOAP 4 Deployment Directory>/logs`. You can read the log files using an editor of your choice.

For example, to log only critical errors, set the `LOG_MODE` variable to `0`. The default `LOG_MODE` value in NonStop SOAP 4 is `2`.

Table 7 lists the different logging levels and the corresponding `LOG_MODE` values:

Table 7 Logging Levels

`LOG_MODE` Value	Description
0	Logs only critical errors.
1	Logs errors and critical errors.
2	Logs warnings, errors, and critical errors.
3	Logs information, warnings, errors, and critical errors.
4	Logs debug messages, information, warnings, errors, and critical errors.
5	Logs user-level log messages, errors, and critical errors.
6	Logs traces, debug messages, information, warnings, errors, and critical errors.

Example 1 shows a sample `itp_axis2.config` configuration file.

Example 1 A Sample itp_axis2.config Configuration File

```
#
#   i t p _ a x i s 2 . c o n f i g
#
#  This file resides in / and contains
#  information for the deployment described at /home/usr1/t0865h01
#
#  This file was generated by /usr/tandem/nssoap/t0865h01/bin/deploy.sh

#########################################################################
# Set environment variables used by this deployment
#
#

# ----------------------------------------------------------------------
# Axis2_DEPLOYMENT_ROOT is the root directory for the
# deployment/operational files for this particular NSSOAP 4 SOAP
# SERVER deployment

set AXIS2_DEPLOYMENT_ROOT /home/usr1/t0865h01

# ----------------------------------------------------------------------
# Axis2c is the full pathname of the axis2cgi.pway file

set Axis2c $AXIS2_DEPLOYMENT_ROOT/bin/axis2cgi.pway

#########################################################################
#
#   Add a Filemap for a "nssoap" region. The URL to this new region
#   would be http://www.yourserver.com:port/axis
#
Filemap /axis2c $Axis2c
Filemap /axis2c/client $AXIS2_DEPLOYMENT_ROOT/client
#########################################################################
# Force decoding of form encoded data and Query String
# directly into environment variables without user
# intervention.
#########################################################################

Region  /axis2c/* {
    AddCGI AUTOMATIC_FORM_DECODING ON
}

Server  $Axis2c {
        CWD $AXIS2_DEPLOYMENT_ROOT/bin
        Env AXIS2C_HOME=$AXIS2_DEPLOYMENT_ROOT
        Env ICU_DATA=/usr/tandem/xml/T0563H01/lib/icu/data/
        Env LOG_MODE=2
        Priority 160
        Numstatic 1
        Maxlinks 1
        Maxservers 5
        set env(HOMETERM) [exec tty]
        Stdin  $AXIS2_DEPLOYMENT_ROOT/logs/ss_stdin.log
        Stdout $AXIS2_DEPLOYMENT_ROOT/logs/ss_stdout.log
        Stderr $AXIS2_DEPLOYMENT_ROOT/logs/ss_stderr.log
}
```

In the `itp_axis2.config` file sample, the `Stdout`, `Stdin` and `Stderr` parameters define the filenames for standard output, input streams, and error streams, respectively.

NOTE: NonStop SOAP 4 logs output to the `soaperror.log` file. If the log level is set to tracing, then NonStop SOAP 4 logs trace messages to the `soaperror.log.trc` file.

NOTE: The location specified for ICU_DATA must be a valid, existing location containing the ICU files to be used for encoding. If the location does not exist, SOAP server logs an error message. If the files in the given location are missing or corrupt, then encoding features will work only for the UTF-8 encoding type.

Defining the Log File Size

The NonStop SOAP 4 server supports log and trace file rollover after the files reach a specific size. The default file size is set to 20 MB, after which the logging and tracing rolls over for `soaperror.log` and `soaperror.log.trc` files.

For specifying a different rollover size, set the `LOG_SIZE` OSS environment variable in the `itp_axis2.config` file. The value must be numeric and the base unit is in MB. The minimum value is 1, which indicates that the file size of 1 MB triggers a rollover.

CAUTION: A negative or zero value indicates that rollover must not be attempted. In this case the file sizes can reach the maximum file size supported by the underlying file system. HP does not recommend a negative or zero value for `LOG_SIZE`.

Defining Separate Log and Trace Files for NonStop SOAP 4 Servers

By default all the static and dynamic SOAP servers running in a specific iTP WebServer PATHMON logs information to the same `soaperror.log` or `soaperror.log.trc` files.

To create separate log and trace files for individual NonStop SOAP 4 servers, in the `itp_axis2.config` file, set the `LOG_PER_PROCESS` OSS environment variable to `Yes`. With this setting, different files are created with the process name as a suffix to the log or trace file name.

The `axis2.xml` File

The `axis2.xml` file in the *<NonStop SOAP 4 Installation Directory>* directory is used to configure a specific instance of a NonStop SOAP 4 deployment. The `axis2.xml` file is created when the *<NonStop SOAP 4 Deployment Directory>*`/bin/deploy.sh` script is run to deploy NonStop SOAP 4.

The configurations set in the `axis2.xml` configuration file are loaded during the NonStop SOAP 4 server startup, and applied to all the services deployed under the specific NonStop SOAP 4 deployment.

The `axis2.xml` file is used to perform the following tasks:

- "Setting the Message Receiver and the Message Exchange Pattern" (page 180)
- "Attaching a Module at the Global Level" (page 181)
- "Attaching Message Receiver User Functions at the Global Level" (page 181)
- "Specifying the Order of Phase Invocation in NonStop SOAP 4 Message Processing" (page 182)

Setting the Message Receiver and the Message Exchange Pattern

During startup, NonStop SOAP 4 reads the name of the Message Receiver class and the default message exchange pattern (MEP) from the `axis2.xml` file.

NonStop SOAP 4 enables you to set the Message Receiver and the MEP to communicate with the service. For more information about Message Receiver and Message Exchange Pattern, see "Introduction to NonStop SOAP" (page 27).

You can set the Message Receiver and MEP using the `class` and `mep` attribute of the Message Receiver elements in the `axis2.xml` configuration file as follows:

```
<messageReceiver class="message-receiver-class" mep="name of the mep"/>
```

where,

`class [axis2_pway_receiver | axis2_receivers ]`

is the Message Receiver class to be attached with NonStop SOAP 4. This is a mandatory attribute.

The NonStop SOAP 4 distribution supports the following Message Receivers:

`axis2_pway_receiver`

specifies the Pathway Message Receiver to be used for communication with NonStop processes or Pathway services. The default value is `axis2_pway_receiver`.

`axis2_receivers`

specifies the XML Message Receiver to be used for services developed using server APIs.

`mep [IN-OUT | IN-ONLY ]`

specifies the message exchange pattern to be used when communicating with the client. This is a mandatory attribute. The default value is `IN-OUT`.

Attaching a Module at the Global Level

A module can be attached at the global level so that its handlers are invoked every time a service is called in the NonStop SOAP 4 deployment. For more information about modules, see Chapter 7: "Customizing NonStop SOAP 4 Message Processing" (page 124).

The `module` element in the `axis2.xml` file is optional and is used to attach a particular module with NonStop SOAP 4. This module is attached globally and will be invoked on every service request.

For example, to globally add a new module, `mod_logging`, add the following entry in the `axis2.xml` file.

```
<module ref="mod_logging"/>
```

where,

`ref`

is the attribute that accepts the module name of the module being referenced.

NOTE: To attach a module, the module DLL must be added to the `/modules` directory in the NonStop SOAP 4 deployment directory. In the `mod_logging` example, the DLL object of `mod_logging` is located in the *<NonStop SOAP 4 Installation Directory>*`/sample_services/modules/mod_logging` directory.

Attaching Message Receiver User Functions at the Global Level

A Message Receiver User Function can be attached at the global level so that it is invoked every time a service is called in the NonStop SOAP 4 deployment. For more information about the Message Receiver User Function, see Chapter 3: "Migrating NonStop SOAP 3 Services to NonStop SOAP 4 or Higher Versions" (page 52).

For example, to add a new Message Receiver User Function `axis2_MRUF_log` globally, add the following entry in the `axis2.xml` file.

```
<parameter name="MessageReceiverUserFunctions">axis2_MRUF_log</parameter>
```

where,

`axis2_MRUF_log`

is the class name of the Global Message Receiver User Function.

The `parameter` element in the `axis2.xml` file is optional and is used to define the custom parameters in NonStop SOAP 4. The `MessageReceiverUserFunctions` parameter is used to define DLLs, which are attached as Message Receiver User Functions. If the parameter is defined in `axis2.xml` file, then the Message Receiver User Functions are attached globally and invoked on every service request.

Specifying the Order of Phase Invocation in NonStop SOAP 4 Message Processing

NonStop SOAP 4 enables you to specify the order in which the pre-defined and user-defined phases can be invoked during a message processing cycle. To specify the invocation order of the phases, modify the `phaseorder` attribute to reflect the desired order of invocation. You can also customize user–defined phases using the `phaseOrder` attribute.

NOTE: For more information about the `phaseOrder` element attribute, see Chapter 7: "Customizing NonStop SOAP 4 Message Processing" (page 124).

For example, to add a phase named `UserPhase` to the `inflow` after the pre-defined phases, add the highlighted entry in the `axis2c.xml` file:

```
<phaseOrder type = "inflow">
     <!-- System pre defined phases       -->
     <phase name="Transport"/>
     <phase name="PreDispatch"/>
     <phase name="Dispatch"/>
     <phase name="PostDispatch"/>
     <!-- End system pre defined phases       -->
     <!-- User defined phases could be added here -->
     <phase name="UserPhase"/>
</phaseOrder>
```

The `services.xml` File

The `services.xml` file describes a particular service deployed in a NonStop SOAP 4 deployment and enables you to configure the service.

The `services.xml` file is located in the service directory of a particular NonStop SOAP deployment. For example, if the deployment directory for NonStop SOAP 4 is `/home/usr1/t0865h01`, and a service named `test` is deployed in it, the `services.xml` file for the `test` service will be located in the `/home/usr1/t0865h01/services/test` directory.

```
OSS> cd /home/usr1/t0865h01/services/test
OSS> ls
          services.xml       test.wsdl
```

To be a valid service, each service must have a `services.xml` file in the service directory. NonStop SOAP 4 provides service-level customization by allowing the `services.xml` file to override the values defined globally using the `axis2.xml` file. Sample `services.xml` files are available with the sample services distributed with NonStop SOAP 4.

The `services.xml` file can be generated for Pathway services using the `SoapAdminCL` tool. For more information about the `SoapAdminCL` tool, see Chapter 10: "NonStop SOAP Tools" (page 194).

The `services.xml` file is used to perform the following tasks:

- "Updating the Service Parameters" (page 183)
- "Defining Multiple SOAP Response Selection Criteria" (page 186)
- "Controlling TMF Transaction Support" (page 188)
- "Engaging a Module at the Service Level" (page 189)
- "Setting the Operation-Specific MEP" (page 189)
- "Setting the Operation-Specific Message Receiver" (page 190)

NOTE: The `services.xml` file is an output file produced by the `SoapAdminCL` tool in response to a particular SDL file. Modifications to `services.xml` should be undertaken carefully, as they will be overwritten when `SoapAdminCL` is subsequently run on the input SDL file.

Updating the Service Parameters

The `services.xml` file is used to set the service parameters to configure a service as a Pathway application, a process, or as a DLL.

NOTE: The service parameters are set using the `parameter` tag and are defined as name-value pairs. You need not modify the service parameters because the `services.xml` file is generated by the `SoapAdminCL` tool.

This section describes the following topics:

- "Configuring a service as a Pathway application" (page 183)
- "Configuring a service as a process" (page 183)
- "Configuring a service as a DLL" (page 183)
- "Other Service Parameters" (page 184)

Configuring a service as a Pathway application

To configure a service as a Pathway application, use the `axis2_pway_receiver` Message Receiver and set the following parameters in the `services.xml` file:

- `wsdl_path`: specifies the location of the WSDL file of the service.
- `pathmon`: specifies the name of the PATHMON on which the Pathway service is running.
- `serverclass`: specifies name of the server class for the Pathway service.
- `serverLanguage`: specifies the language in which the service is implemented.

For example, to update the parameters, add the following entries in the `services.xml` file:

```
<parameter name="wsdl_path">/home/t0865h01/services/
 empdb/SoapPW_empdb.wsdl</parameter>
<parameter name="pathmon">$AXIS</parameter>
<parameter name="serverclass">EMPDB</parameter>
<parameter name="serverLanguage">C</parameter>
```

Configuring a service as a process

To configure a service as a process, use the `axis2_pway_receiver` Message Receiver and set the following parameters in the `services.xml` file:

- `wsdl_path`: specifies the location of the WSDL file of the service.
- `process`: specifies name of the process that is running.
- `serverLanguage`: specifies the language in which the service is implemented.

For example, to update the parameters, add the following entries in the `services.xml` file:

```
<parameter name="wsdl_path">/home/t0865h01/services/
 empdb/SoapPW_empdb.wsdl</parameter>
<parameter name="process">$AXIS</parameter>
<parameter name="serverLanguage">C</parameter>
```

Configuring a service as a DLL

To configure a service as a DLL, use the `axis2_receivers` Message Receiver and set the following parameters in the `services.xml` file:

- `wsdl_path`: specifies the location of the WSDL file of the service.
- `serviceClass`: specifies the class name of the service DLL. If the service DLL name is `libecho.so`, the class value will be `echo`.

For example, to update the parameters, add the following entries in the `services.xml` file:

```
<parameter name="wsdl_path">/home/t0865h01/services/
 empdb/SoapPW_empdb.wsdl</parameter>
<parameter name="serviceClass">echo</parameter>
```

Other Service Parameters

- The *DDLMapping* parameter setting determines the use of the DDLMapping.xml file for processing a SOAP message. If the parameter is set to Yes, then the NonStop SOAP 4 server uses the DDLMapping.xml file for processing the SOAP message, otherwise, it uses the WSDL file for processing the SOAP message. For example, when the NonStop SOAP 4 server finds the following entry in the services.xml file, it reads the DDLMapping.xml file:

  ```
  < parameter name="DDLMapping">Yes< /parameter>
  ```

  The SoapAdminCL tool generates the DDLMapping.xml file for specific conditions that cannot be processed by the NonStop SOAP 4 server with the information in the WSDL file. The SoapAdminCL tool sets this parameter in the generated services.xml file.

- The *AlignmentRule* parameter setting determines the alignment rule. For information about the valid parameter settings, see the following tables. The default is even for COBOL services and datatype for C services. For example, the NonStop SOAP 4 server aligns the data elements according to the element datatype when it finds the following entry in the services.xml file.

  ```
  < parameter name="AlignmentRule">datatype< /parameter>
  ```

  NOTE: The COBOL services cannot have specific elements of a structure marked as SYNC. All elements within a structure must be marked as SYNC. If all the elements of a structure are marked as SYNC, use the datatype setting.

  The following tables list the valid settings for this parameter for C and COBOL services.

Table 8 Valid AlignmentRule setting for existing C services that use DDL generated header files

C option	DDL setting	AlignmentRule setting
CSHARED2	C00CALIGN	even
SHARED2	CFIELDALIGN_MATCHED2	shared2
SHARED2	C_MATCH_HISTORIC_TAL	shared2
SHARED8	SHARED8	shared8
AUTO	SHARED8	datatype
PLATFORM	SHARED8	datatype

Table 9 Valid AlignmentRule setting for existing C services that does not use DDL generated header files

C option	AlignmentRule setting
CSHARED2	even
SHARED2	shared2
SHARED8	shared8
AUTO	datatype
PLATFORM	datatype

Table 10 Valid AlignmentRule setting for existing COBOL services that use DDL generated header files

COBOL option	DDL setting	AlignmentRule setting
Default	C00CALIGN	even
Default	CFIELDALIGN_MATCHED2	shared2
Default	C_MATCH_HISTORIC_TAL	shared2
Default	SHARED8	shared8

Table 11 Valid AlignmentRule setting for existing COBOL services that does not use DDL generated header files

COBOL option	AlignmentRule setting
Default	even
All elements marked as SYNC	datatype

NOTE: For C and COBOL services that do not include DDL generated header files, use the `SOAPAdminCL` tool with `-fs` option.

- The *`responseType`* parameter setting determines the processing of `BLANK` response elements. If the parameter is set to `strict`, then the NonStop SOAP 4 server builds the XML response, according to the service definition, even if `BLANK` elements are present. If the parameter is set to `lenient`, then the NonStop SOAP 4 server does not create the `BLANK` elements and builds the minimal size XML response.

  In C services, all bytes of an element filled with the null character (`\0`) denote a `BLANK` element. In COBOL services, all bytes of an alphabetic, alpha-edited, or alphanumeric element filled with spaces, or all bytes of a numeric or numeric-edited element filled with zero denote a `BLANK` element.

  An entity is considered optional if:

  - The corresponding DDL field is marked with a @SOAP_OPTIONAL comment tag
  - It is an element in WSDL, `minOccurs` value is set to 0
  - It is an attribute in WSDL, `use="optional"` is set

  The following table lists the display behavior for different *`responseType`* parameter and `BLANK` element settings.

Table 12 Displaying `BLANK` element in response XML

`BLANK` element is	*`responseType`* set to `strict`	*`responseType`* set to `lenient`
`OPTIONAL`	No	No
`MANDATORY`	Yes	No
Greater than `minOccurs` and less than `maxOccurs`	No	No
Greater than `minOccurs` and less than `OccDependON`	Yes	No
Less than `minOccurs`	Yes	No

NOTE: If an optional element or attribute is omitted in XML request, BLANK value is sent.

Defining Multiple SOAP Response Selection Criteria

NonStop SOAP 4 enables you to define multiple response messages for each operation in a Web service based on a multiple response selection criterion. The response selection criterion can be defined using the following name attributes of the parameter element in the `services.xml` configuration file.

- `TotalResponse`

  This element defines the total number of responses that must be considered while selecting a single response for a given operation. This is a mandatory attribute.

  ```
  <parameter name="TotalResponse"><total response></parameter>
  ```

  For example, to define two responses for an operation:

  ```
  <parameter name="TotalResponse">2</parameter>
  ```

- `Response(x)`

  An operation can expect multiple responses for a given request. Each response must mention the corresponding element name from the WSDL file. In the `Response(x)` parameter, x is an integer, and its value varies from 0 to `total response` -1. Therefore, if the `total response` parameter value is 3, there will be three `Response(x)` parameters with the following sample syntax:

  ```
  <parameter name="Response0">Element Name</parameter>
  <parameter name="Response1">Element Name 1</parameter>
  <parameter name="Response2">Element Name 2</parameter>
  ```

  This is a mandatory attribute.

- `Response(x)_BufferValue`

  This element defines a buffer value. The buffer value is compared with the response returned from the service.

  ```
  <parameter name="Response0_BufferValue">00</parameter>
  ```

  This is an optional element. It is mandatory only if the `Response(x)_StartIndex` and `Response(x)_EndIndex` parameters are specified for a response selection criterion.

- `Response(x)_ComparisonOp [ eq | gt | lt | ne ]`

  This element defines the comparison operator that compares the `Response(x)_BufferValue` parameter specified with the response buffer [`start index : endIndex-1`] received from the service. The values defined for the comparison operator are:

  `eq`
  : Buffer value must be equal to `response[start-index: end-index-1]`.

  `gt`
  : Buffer value must be greater than `response[start-index: end-index-1]`.

  `lt`
  : Buffer value must be less than `response[start-index: end-index-1]`.

  `ne`
  : Buffer value must not be equal to `response[start-index: end-index-1]`.

  ```
  <parameter name="Response0_ComparisonOp">ne</parameter>
  ```

  This is an optional parameter. If not specified, the default comparison operator used by NonStop SOAP 4 is `eq`.

- `Response(x)_StartIndex`

  This element defines the offset in the response buffer for the data to be evaluated using the `ComparisonOP` and `BufferValue` defined for a particular `Response`.

For example, if the StartIndex is specified for the first response selection criterion, the sample syntax will be:

```
<parameter name="Response0">Element Name</parameter>
<parameter name="Response0_StartIndex">0</parameter>
```

This is an optional element.

- Response(x)_EndIndex

  This element defines the offset in the response buffer up to which the data must be evaluated against the response selection criterion. The data selected from the response buffer ranges from the Response(x)_StartIndex parameter to the Response(x)_EndIndex parameter.

  For example:

  ```
  <parameter name="Response0">Element Name</parameter>
  <parameter name="Response0_StartIndex">0</parameter>
  <parameter name="Response0_EndIndex">2</parameter>
  ```

  This is an optional element. It is mandatory only if the Response(x)_StartIndex parameter is specified for the response selection criterion.

- Response(x)_ConversionOp [string | decimal]

  This tag defines the conversion that must be done on the Response(x)_BufferValue parameter before comparing it with the response[start-index:end-index-1] buffer. The permissible values for this parameter are:

  string
  : performs the string conversion on the Response(x)_BufferValue parameter before comparing it with response[start-index:end-index-1].

  decimal
  : performs the decimal conversion on the Response(x)_BufferValue parameter before comparing it with response[start-index:end-index-1].

  For example:

  ```
  <parameter name="Response0_ConversionOp">string</parameter>
  ```

  This is an optional parameter. If not specified, the default conversion value is string.

- Response(x)_Default [true | false]

  If this parameter is set to true, it indicates that this is the default response element to be selected if no matching response selection criteria specified for the operation is found. This is an optional parameter.

  ```
  <parameter name="Response0_Default">true</parameter>
  ```

- Response(x)_Fault [true | false]

  If this parameter is set to true and the response received from the service matches the response selection criterion, NonStop SOAP 4 considers the response received to be a fault response. This parameter is optional. For example:

  ```
  <parameter name="Response0_Fault">true</parameter>
  ```

For example:

```
<parameter name= "Response0">TestElementResponse</parameter>
<parameter name= "Response0_StartIndex">0</parameter>
<parameter name= "Response0_EndIndex">2</parameter>
<parameter name= "Response0_BufferValue">00</parameter>
```

In the example, on receiving a response from the service, NonStop SOAP 4 compares bytes 0 (StartIndex) to 1 (EndIndex – 1) received in the response with the specified Response0_BufferValue. If a comparison operator is not specified, the default value is "equal

to" and the default conversion operator is "string". If the buffers match, the response structure in Response0 is selected to return the response to the client.

Controlling TMF Transaction Support

NonStop SOAP 4 enables you to specify whether a TMF transaction must be started for a given operation. To achieve this, specify the TMFTransactionSupport parameter in the name attribute of the parameter element in the services.xml file. This is an optional element. For example:

```
<parameter name="TMFTransactionSupport"><parameter-value></parameter>
```

where,

<parameter-value> [Required | Supports | Never]

is the parameter value. The options are Required, Supports,or Never. The default value is Supports.

Required

indicates that the operation in the SOAP request will always be performed within a TMF transaction.

- If the client request does not have the Transaction element as a part of the SOAP request header, NonStop SOAP server will begin a TMF transaction before invoking the service and commit or abort the transaction.
- If the client request has a Transaction element in the SOAP request header, NonStop SOAP will process the transaction element.
- For server initiated transactions, NonStop SOAP 4 will set the transaction timeout equal to the value of the TransactionTimeOut parameter in the services.xml file (effective timeout will be lower than the timeout set in the services.xml file and TMFAutoAbort timeout system configuration).

Supports

indicates that the operation in the SOAP request will be performed within a client-initiated TMF transaction. If the client request does not have the transaction or session element in the SOAP request header, NonStop SOAP 4 will perform the operation in the SOAP request without starting any transaction.

Never

indicates that NonStop SOAP 4 will not initiate a transaction.

NOTE: For more information about transaction initiated by the client request, see Transaction Management (page 236) .

Example 2 shows a sample services.xml configuration file.

Example 2 A Sample `services.xml` Configuration File

```
<service name="EMPDB">

        <parameter name="ServiceClass" locked="xsd:false">EMPDB</parameter>


        <!--Uncomment to specify static WSDL path-->
        <parameter name="wsdl_path">/home/usr1/t0865h01/services/empdb_p
athway/SoapPW_EMPSVR.wsdl</parameter>
        <parameter name="pathmon">$AXIS</parameter>
        <parameter name="serverclass">EMPDB</parameter>
        <description>
                Service template file
        </description>



        <operation name="EmpInfo">
                <messageReceiver class="axis2_pway_receiver" />

                <parameter name="TMFTransactionSupport">Supports</parameter>
                <parameter name="TotalResponse">2</parameter>

                <parameter name="Response0">EmpInfoResponse0</parameter>
                <parameter name="Response0_Default">yes</parameter>

                <parameter name="Response1">EmpInfoResponse1</parameter>
                <parameter name="Response1_StartIndex">0</parameter>
                <parameter name="Response1_EndIndex">2</parameter>
                <parameter name="Response1_BufferValue">00</parameter>
                <parameter name="Response1_ComparisonOp">eq</parameter>
                <parameter name="Response1_ConversionOp">string</parameter>
                <parameter name="Response1_Fault">yes</parameter>

        </operation>
</service>
```

Engaging a Module at the Service Level

A module can be attached at the service level so that its handlers are invoked every time a service is called. A module can be attached with a service using the `module` element. This element specifies the module that is referenced by the service. The sample syntax for the `module` element is:

```
<module ref="module class name"/>
```

where,

`ref`

is the attribute that accepts the module name of the module being referenced. This is a mandatory attribute.

Setting the Operation-Specific MEP

The operation-specific MEP information for a service can be specified using the `operation` element. The sample syntax for the `operation` element is:

```
<operation name="operation-name" mep="mep-uri">
</operation>
```

where,

`name`

is the operation name specified in the WSDL file for the service. This is a mandatory attribute.

mep
: is the MEP Web address to be used for this operation. If not specified, the default MEP Web address is http://www.w3.org/2004/08/wsdl/in-out, which maps to the IN-OUT MEP.

Setting the Operation-Specific Message Receiver

NonStop SOAP 4 enables you to override the Message Receiver configurations set in the `axis2.xml` file at the operation level. You can specify the Message Receiver to be invoked by NonStop SOAP 4 for an operation.

For example, to specify the Message Receiver to be invoked for the `EmpInfo` operation of the `empdb` service, include the following entry in the `services.xml` file for the `empdb` service.

```
<operation name="EmpInfo">
     <messageReceiver class="axis2_pway_receiver" />
</operation>
```

where,

`class [axis2_pway_receiver | axis2_receiver ]`
: is the attribute for the Message Receiver that is used for request processing. This is a mandatory attribute. The NonStop SOAP 4 distribution supports the following:

  `axis2_pway_receiver`
  : specifies the Pathway Message Receiver to be used for communication with NonStop processes or Pathway services.

  `axis2_receiver`
  : specifies the XML Message Receiver to be used for services developed using server APIs.

In addition to `axis2_pway_receiver` and `axis2_receiver`, you can also customize Message Receivers.

The `module.xml` File

The `module.xml` file is a module descriptor that contains configuration contexts for the modules configured in NonStop SOAP 4. Each module directory must have a `module.xml` file in the module directory.

NOTE: For more information about modules, see the "Customizing NonStop SOAP 4 Message Processing" (page 124).

Sample `module.xml` files are available in the NonStop SOAP 4 distribution. For example, if the installation directory of NonStop SOAP 4 is `/usr/tandem/nssoap/t0865h01`, the `module.xml` file for the addressing module will be located at:

```
/usr/tandem/nssoap/t0865h01/modules/addressing
```

Example 3 shows a sample `module.xml` configuration file structure.

Example 3 A Sample `module.xml` File Structure

```
<module>
      <inflow>
            <handler>
                  <order/>
            </handler>
      </inflow>
      <outflow>
            <handler>
                  <order/>
            </handler>
      </outflow>
      <INfaultflow>
            <handler>
```

```
                    <order/>
                </handler>
        </INfaultflow>
        <Outfaultflow>
                <handler>
                        <order/>
                </handler>
        </Outfaultflow>
</module>
```

where,

`<module>`
: represents the entire module to be configured.

`<inflow>`
: represents a collection of one or more `<handler>` elements that must be invoked while processing the input flow for a particular message.

`<handler>`
: represents the unit or class that must be invoked by the NonStop SOAP 4 server. Multiple handlers can be used.

`<order>`
: represents the order and phase in which its parent handler must be invoked.

The `module.xml` file is used to perform the following tasks:

- "Specifying the Module Name and Module Class File Name" (page 191)
- "Defining a New Handler and Specifying its Invocation Order" (page 191)

Specifying the Module Name and Module Class File Name

The `module` element must be updated with the module name and the class name. The sample syntax to specify the module name and class file name in the `module.xml` file is:

```
<module name="module-name" class="module-class">
```

where,

`name`
: is the name of the module to be configured. This is a mandatory attribute.

`class`
: is the module class name. If the module DLL name is `libaxis2_mod_log.so`, the class value will be `axis2_mod_log`. This is a mandatory attribute.

Defining a New Handler and Specifying its Invocation Order

A module is a collection of handlers. To add a handler, specify the handler name and the handler class name in the `<handler>` element of the `module.xml` file as:

```
<handler name="handler-name" class="handler-class">
```

where,

`name`
: is the name of the handler. This is a mandatory attribute.

`class`
: specifies the DLL that implements the handler. If the name of the DLL is `libaxis2_loghandler.so`, the class value will be `loghandler`.

The `module.xml` file provides the flexibility to decide when a particular handler must be invoked within a given phase. The `<order>` element has attributes that can help invoke handlers depending

on when another handler is invoked. To specify the order of invoking handlers, include the following entry with the required information in the `module.xml` file:

```
<order phase="phase-name" phaseLast="true" phaseFirst="first"
before="handler-name" after="handler-name"/>
```

where,

`phase`
: indicates the phase in which this handler must be invoked. This is a mandatory attribute.

`phaseLast [true | false]`
: indicates that the parent handler is the last handler that must be invoked for the given phase. This is an optional attribute.

`phaseFirst [true | false]`
: indicates that the parent handler is the first handler that must be invoked for the given phase. This is an optional attribute.

`before`
: is the attribute value that takes the name of a valid handler. The valid handler is invoked before the handler specified in the `before` attribute. This is an optional attribute.

`after`
: is the attribute value that takes the name of a valid handler. The valid handler is invoked after the handler specified in the `after` attribute. This is an optional attribute.

Defining the Module-Specific Parameters

You can define the module-specific parameters by using the `<parameter>` element. This element is a child of the `<module>` element. To add module-specific parameters, include the following entry with the required information in the `module.xml` file:

```
<parameter name="parameter-name" locked="false"><parameter-value></parameter>
```

where,

`name`
: is the parameter name. This is a mandatory attribute for the parameter tag.

`locked [true | false]`
: specifies whether the parameter can be modified within the user code. If this attribute is set to `true`, the parameters can be modified. The default value is `false`.

`<parameter-value>`
: is the parameter value associated with the parameter name.

Example 4 shows a sample `module.xml` configuration file.

Example 4 A Sample `module.xml` Configuration File

```
<module name="addressing" class="axis2_mod_addr">
    <inflow>
        <handler name="AddressingInHandler" class="axis2_mod_addr">
            <order phase="Transport" before="addressing_based_dispatcher"/>
        </handler>
    </inflow>

    <outflow>
        <handler name="AddressingOutHandler" class="axis2_mod_addr">
            <order phase="MessageOut"/>
        </handler>
    </outflow>

    <Outfaultflow>
        <handler name="AddressingOutHandler" class="axis2_mod_addr">
```

```
            <order phase="MessageOut"/>
        </handler>
    </Outfaultflow>
</module>
```

For more information about modules and handlers, see “Customizing NonStop SOAP 4 Message Processing” (page 124).

10 NonStop SOAP Tools

The NonStop SOAP 4 distribution includes tools that help perform important tasks, such as:

- Generating the WSDL file, the `services.xml` file, HTML clients, and SOAP request and response XML files for NonStop SOAP 4
- Migrating services from NonStop SOAP 3 to NonStop SOAP 4
- Generating client stubs and service skeleton files using WSDL files as input

This chapter describes the following tools that are available with the NonStop SOAP 4 distribution:

- "The `SoapAdminCL` Tool" (page 194)
- "The `WSDL2PWY` Tool" (page 203)
- "The `WSDL2C` Tool" (page 211)

This chapter also describes the "The NonStop SOAP 4 Administration Utility" (page 215).

The `SoapAdminCL` Tool

The `SoapAdminCL` tool is a standalone command-line tool that runs in the OSS environment on NonStop systems. You can use the `SoapAdminCL` tool to perform the following tasks:

- Generate the WSDL file, the `services.xml` file, HTML clients, XML schema files, and SOAP request and response XML files using the SDL file as an input
- Generate module and handler stubs
- Generate Message Receiver User Functions stubs
- Generate the SDL file from the NonStop SOAP 3 SDR file

NOTE: All the options are case sensitive.

This section describes the following topics:

- "Generating NonStop SOAP 4 Files using the SDL File" (page 194)
- "Generating Module and Handler Stubs" (page 199)
- "Generating Message Receiver User Functions Stubs" (page 200)
- "Generating the NonStop SOAP 4 SDL File from the NonStop SOAP 3 SDR File" (page 201)
- "Additional Options" (page 202)

Generating NonStop SOAP 4 Files using the SDL File

The `SoapAdminCL` tool uses the SDL file and the DDL dictionaries to generate the following:

- WSDL file: used by NonStop SOAP 4 during runtime
- The `services.xml` file: used by NonStop SOAP 4 during runtime
- HTML client files: used to test your services
- XML schema files: used to provide request and response structure in XML schema
- SOAP request and response XML files: used to provide request and response structure in XML
- Migration WSDL files: used to migrate your applications from NonStop SOAP 3 to NonStop SOAP 4

NOTE:

- The migration WSDL files are generated only when the `-m` migration flag is used.
- You can generate an SDL file for NonStop SOAP 4 services using the following:
  - NonStop SOAP 4 Administration Utility
  - NonStop SOAP 3 SDR file with the NonStop SOAP 4 `SoapAdminCL` tool
  - Copying and updating the SDL files distributed with the NonStop SOAP 4 distribution

  For more information about the SDL file, see "NonStop SOAP 4 Service Description Language" (page 153).
- To use request encoding or response encoding, set the `ICU_DATA` environment variable to the location of the ICU data files before executing SoapAdminCL command:

  ```
  OSS> export ICU_DATA=/usr/tandem/xml/T0563H01/lib/icu/data/
  ```

  You can set the request encoding using the `SrvrEncoding` attribute of the `Service` element and response encoding using the `RspEncoding` attribute of the `Operation` element in the SDL file. For more information on using the encoding attributes in the SDL file, see "Configuring the response encoding" (page 230).

Figure 14 shows the files generated by the `SoapAdminCL` tool.

Figure 14 Files Generated by the `SoapAdminCL` Tool

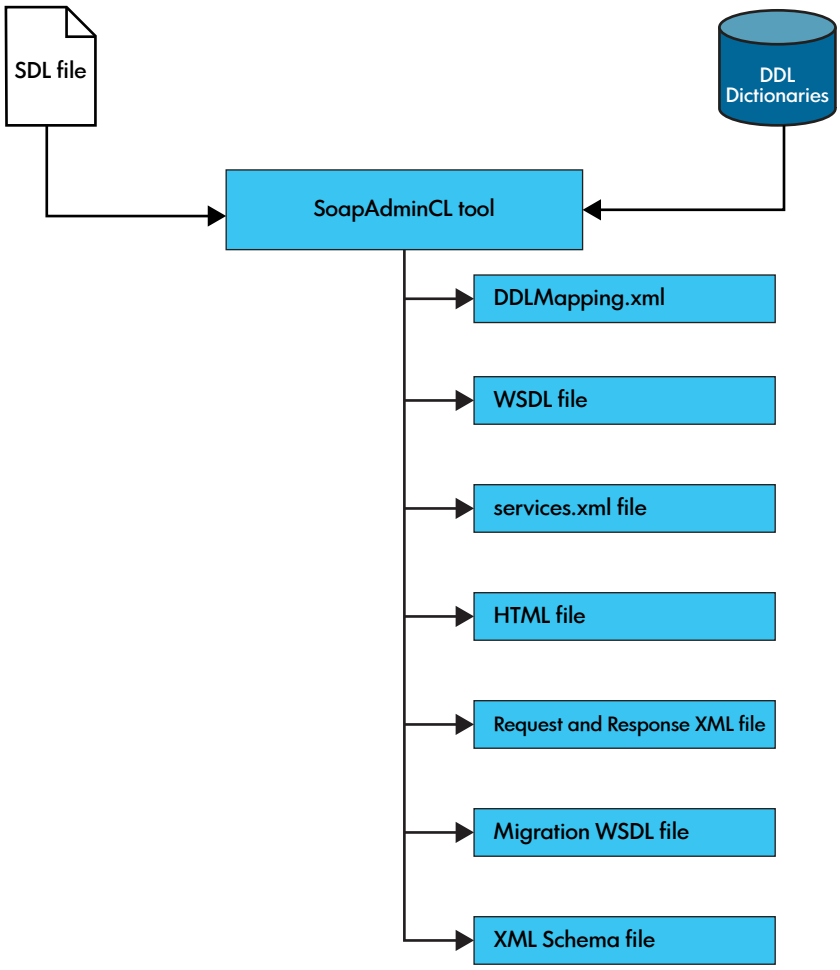


Table 13 lists the location of the files generated by the `SoapAdminCL` tool.

Table 13 Location of Files Generated by the SoapAdminCL Tool

File Name	Location
WSDL	*<output directory>*/services/*<service_name>*/SoapPW_*<service_name>*.wsdl
DDLMapping.xml	*<output directory>*/services/*<service_name>*/DDLMapping.xml
services.xml	*<output directory>*/services/*<service_name>*/services.xml
HTML	*<output directory>*/client/*<service_name>*/SoapPW_*<operation_name>*.html
Request and Response XML file	*<output directory>*/client/*<service_name>*/pway-xml/SoapPW_*<operation_name>*.xml *<output directory>*/client/*<service_name>*/pway-xml/SoapPW_*<operation_name>*Response0.xml
Migration WSDL file	*<output directory>*/client/*<service_name>*/wsdl/SoapPW_*<operation_name>*.wsdl
XML Schema file	*<output directory>*/client/*<service_name>*/pway-xsd/SoapPW_*<operation_name>*.xsd *<output directory>*/client/*<service_name>*/pway-xsd/SoapPW_*<operation_name>*Response0.xsd

To generate WSDL files, the services.xml file, HTML clients, XML schema files, and SOAP request and response XML files using the SoapAdminCL tool, complete the following steps:

1. Ensure that the DDL dictionaries are available at the location specified in the DDLDictionaryLocation attribute of the SDL file.

   NOTE: If the DDL dictionaries are not available in the specified location, the SoapAdminCL command fails.

2. Generate HTML clients, XML schema files, SOAP request and response XML files, the WSDL file, and the services.xml file using the SoapAdminCL command:

```
OSS> SoapAdminCL -i <SDL file name>
       [-o | -outdir] <output directory>
       [-m | -migration]
       [-f | -force]
       [-w]
```

   where,

   `-i <sdl file name>`
   specifies the name of the SDL file to be processed.

   `[-o | -outdir] <output directory>`
   specifies the directory path, where the HTML clients, XML schema files, SOAP request and response XML files, the WSDL file, and the services.xml file is generated.

   NOTE: If the -o option is not used, the files are placed in the current working directory.

   The directory structure appears as:

```
<output directory>
   /client
     /<service_name>
        SoapPW_<operation_name>.html
        /pway-xml
           SoapPW_<operation_name>.xml
```

```
            SoapPW_<operation_name>Response0.xml
         /pway-xsd
            SoapPW_<operation_name>.xsd
            SoapPW_<operation_name>Response0.xsd

   /services
     /<service_name>
        SoapPW_<service_name>.wsdl
        services.xml
```

where,

<service_name>

is the name of the service specified in the `name` attribute of the `service` element in the SDL file.

`[-m | -migration]`

specifies that a separate file for each operation will be generated.

The `[-m |-migration]` option is useful when you need to ensure that the WSDL files generated in NonStop SOAP 4 match as closely as possible with the WSDL files generated by NonStop SOAP 3.

The generated file structure appears as:

```
<output directory>
   /client
      /<service_name>
         SoapPW_<operation_name>.html
         /pway-xml
            SoapPW_<operation_name>.xml
            SoapPW_<operation_name>Response0.xml
         /pway-xsd
            SoapPW_<operation_name>.xsd
            SoapPW_<operation_name>Response0.xsd
         /wsdl
            SoapPW_<operation_name>.wsdl
   /services
      /<service_name>
         SoapPW_<service_name>.wsdl
         services.xml
```

NOTE: NonStop SOAP 3 generates the WSDL file based on each `<operation>` element present in the SDL file. In NonStop SOAP 4, you can generate the WSDL file in a similar manner by using the `SoapAdminCL` tool with the `-m` option. This helps you to migrate a NonStop SOAP 3 application to NonStop SOAP 4.

`[-f | -force]`

specifies that the existing files (if any) created by the `SoapAdminCL` tool will be overwritten.

Ensure that you use this option only when customizing an existing service. If the `[-f | -force]` option is not used, the existing output files for the service will not be overwritten.

[-w]

This option validates the NonStop SOAP server and PORT that are mentioned in SDL file. This option works when SDL is provided by using –i option. This option checks if iTP WebServer is running on a given PORT (For example, default PORT is 80). If yes, It also checks if the NonStop SOAP 4 server is running under this iTP Webserver.

For example:

- The following command generates the HTML client files, XML schema files, the WSDL file, the `services.xml` file, and the SOAP request and response XML files in the current directory (when the SDL file contains definitions for a single service with two operations):

```
OSS> SoapAdminCL -i mysdl.xml
```

The generated directory structure appears in the current directory as:

```
<current directory>
  /client
     /myservice
        SoapPW_myoperation1.html
        SoapPW_myoperation2.html
        /pway-xml
           SoapPW_myoperation1.xml
           SoapPW_myoperation1Response0.xml
           SoapPW_myoperation2.xml
           SoapPW_myoperation2Response0.xml
        /pway-xsd
           SoapPW_myoperation1.xsd
           SoapPW_myoperation1Response0.xsd
           SoapPW_myoperation2.xsd
           SoapPW_myoperation2Response0.xsd
  /services
     /myservice
        SoapPW_myservice.wsdl
        services.xml
```

- The following command generates the HTML client files, XML schema files, the WSDL file, the `services.xml` file, and the SOAP request and response XML files at the location mentioned using the `-o` option (when the SDL file contains definitions for a single service with two operations):

  ```
  OSS> SoapAdminCL -i mysdl -o /home/usr/my_nssoap
  ```

  The generated directory structure appears at the location mentioned using the `-o` option as:

  ```
  /home/usr/my_nssoap
    /client
       /myservice
          SoapPW_myoperation1.html
          SoapPW_myoperation2.html
          /pway-xml
             SoapPW_myoperation1.xml
             SoapPW_myoperation1Response0.xml
             SoapPW_myoperation2.xml
             SoapPW_myoperation2Response0.xml
          /pway-xsd
             SoapPW_myoperation1.xsd
             SoapPW_myoperation1Response0.xsd
             SoapPW_myoperation2.xsd
             SoapPW_myoperation2Response0.xsd
    /services
       /myservice
          SoapPW_myservice.wsdl
          services.xml
  ```

- The following command generates the HTML client files, XML schema files, WSDL files, the `services.xml` file, and the SOAP request and response XML files at the location mentioned with the `-o` option. The `-m` option generates the operation-based WSDL file to facilitate migration of applications from NonStop SOAP 3 to NonStop SOAP 4 (when the SDL file includes definitions for a single service with two operations):

  ```
   SoapAdminCL -i mysdl -o /home/usr/my_nssoap -m
  /home/usr/my_nssoap
    /client
       /myservice
          SoapPW_myoperation1.html
          SoapPW_myoperation2.html
          /pway-xml
  ```

```
                  SoapPW_myoperation1.xml
                  SoapPW_myoperation1Response0.xml
                  SoapPW_myoperation2.xml
                  SoapPW_myoperation2Response0.xml
               /pway-xsd
                  SoapPW_myoperation1.xsd
                  SoapPW_myoperation1Response0.xsd
                  SoapPW_myoperation2.xsd
                  SoapPW_myoperation2Response0.xsd
               /wsdl
                  SoapPW_myoperation1.wsdl
                  SoapPW_myoperation2.wsdl
         /services
            /myservice
               SoapPW_myservice.wsdl
               services.xml
```

Generating Module and Handler Stubs

To generate the module and handler stubs, use the `SoapAdminCL` tool with the `–mod` command line option.

Use the following command to generate the module and handler stub files:

```
OSS> SoapAdminCL  [-mod | -module] <module_name>
                  [-o | -outdir] <output directory>
                  [-f | -force]
```

where,

`[-mod | -module] <module_name>`
specifies the name of the global module handler.

`[-o | -outdir] <output directory>`
specifies the directory where the module handler stub files are generated.

`[-f | -force]`
specifies that the existing files (if any) created by the `SoapAdminCL` tool will be overwritten.

The directory structure appears as:

```
<output directory>
      /modules
           /<module_name>
               module.xml
               /src
                 <module_name>.c
                 <module_name>.h
                 <module_name>_in_handler.c
                 <module_name>_out_handler.c
                 Makefile
```

For example:

```
SoapAdminCL -mod logging -o /home/usr/my_nssoap -f
```

where,

`/home/usr/my_nssoap`
is the directory where the global module handler stub files are generated.

The directory structure appears as:

```
/home/usr/my_nssoap
       /modules
           /logging
                module.xml
                /src
                  logging.c
```

```
logging.h
logging_in_handler.c
logging_out_handler.c
Makefile
```

Generating Message Receiver User Functions Stubs

Message Receiver User Functions (MRUF) enable you to modify the message buffer, Pathway or service attributes, and customize the message flow in the Message Receiver phase. For more information about MRUF, see "Customizing NonStop SOAP 4 Message Processing" (page 124).

NOTE: The `SoapAdminCL` tool can generate MRUF stubs that can be customized as required.

Use the following command to generate the MRUF stubs:

```
OSS> SoapAdminCL  -MRUF <Message Receiver User Functions name>
                  [-o | -outdir] <output directory>
                  [-f | -force]
```

where,

`-MRUF <Message Receiver User Functions name>`
specifies the name of the Message Receiver User Functions.

`-o <output directory>`
specifies the directory where the MRUF stub files are generated.

`-f | -force`
specifies that the existing files (if any) created by the `SoapAdminCL` tool will be overwritten.

The directory structure appears as:

```
<output directory>
        /MRUserFunctions
           /<Message Receiver User Functions Name>
                /src
                   <Message Receiver User Functions Name>_MRUF.c
                   <Message Receiver User Functions Name>_MRUF.h
                   Makefile
```

For example:

```
SoapAdminCL -MRUF sample_MRUF -o /home/usr/my_nssoap -f
```

where,

`sample_MRUF`
is the name of the MRUF.

`/home/usr/my_nssoap`
is the directory where the MRUF stub files are generated.

The directory structure appears as:

```
/home/usr/my_nssoap
         /MRUserFunctions
             /myapplication
                  /src
                     myapplication_MRUF.c
                     myapplication_MRUF.h
                     Makefile
```

NOTE: The `SoapAdminCL` tool also generates the stub files for service-level module handlers and Message Receiver User Functions if the `GenerateModuleHandlerFiles` and `GenerateMessageReceiverUserFunctionFiles` attributes of the `service` element of the SDL file are set to `yes`. For more information about these attributes, see "NonStop SOAP 4 Service Description Language" (page 153).

Generating the NonStop SOAP 4 SDL File from the NonStop SOAP 3 SDR File

The SoapAdminCL tool enables you to generate the NonStop SOAP 4 SDL file from the NonStop SOAP 3 SDR file to migrate services from NonStop SOAP 3 to NonStop SOAP 4.

For information about the SDL file, see "NonStop SOAP 4 Service Description Language" (page 153).

To generate the NonStop SOAP 4 SDL file from the NonStop SOAP 3 SDR file using the SoapAdminCL tool, complete the following steps:

1. Set the OSS environment variable SOAP_SDLDB_LOC to the Guardian location where the SDR file is available:

   ```
   OSS> export SOAP_SDLDB_LOC=\<vol-name>.<subvol-name>
   ```

   For example:

   ```
   OSS> export SOAP_SDLDB_LOC=\$data.sdrdb
   ```

   where,

   $data
   : is the volume name.

   sdrdb
   : is the subvolume name.

2. Go to the OSS location where you want to generate the SDL file.
3. To generate the SDL file, run the SoapAdminCL tool with the following options:

   ```
   OSS> SoapAdminCL -e <SDL file name>
        [-pathway [-env <space separated Pathmon environment list>]]
        [-process [-env <space separated Process environment list>]]
        [-casesensitive]
   ```

   where,

   -e
   : specifies the SDL file you want to generate from an existing NonStop SOAP 3 SDR file.

   <SDL file name>
   : specifies the name of the SDL file. This is a mandatory parameter when the –e flag is set.

   -pathway
   : specifies that the attributes for the Pathway services in the SDR file must be extracted to the SDL file. This is an optional parameter.

     To export the attributes for specific Pathway services, use the –env option with the –pathway option:

     -env *<space separated Pathmon environment list>*
     : specifies that the attributes for Pathway services, entered in the *<space separated Pathmon environment list>* in the SDR file, must be extracted to the SDL file only.

   -process
   : specifies that the attributes for the NonStop processes in the SDR file must be extracted to the SDL file. This is an optional parameter.

     To export the attributes for specific NonStop processes, use the –env option with the –process option:

     -env *<space separated Process environment list>*
     : specifies that the attributes for NonStop processes, entered in the *<space separated Process environment list>* in the SDR file, must be extracted to the SDL file only.

`-casesensitive`

specifies the case in which the service name is generated.

- when the `-casesensitive` option is used:

  If the SDR file defines a service name in mixed case, the `SoapAdminCL -e <SDL file name> -casesensitive` command generates the service names in mixed case.

  For example, if the SDR file has an employee database service defined thrice in mixed cases in the following order:

  EmpDB ... Empdb .... EMPDB ... , the `SoapAdminCL -e <SDL file name> -casesensitive` command will generate the service names as EmpDB, Empdb, and EMPDB.

- when the `-casesensitive` option is not used:

  If the SDR file defines a service name in mixed case, the `SoapAdminCL -e <SDL file name>` command generates the service names in the same case as that used in the first instance of the service definition in the SDR file.

  For example, if the SDR file has an employee database service defined thrice in mixed cases in the following order:

  EmpDB ... Empdb .... EMPDB ... , the `SoapAdminCL -e <SDL file name>` command will generate the service name as EmpDB, that is, the first employee database service definition in the SDR file. The other two definitions of the service name (Empdb and EMPDB) in the SDR file will be converted to EmpDB.

For example:

- The following command generates the `mysdl.xml` file that includes information about the Pathway services and NonStop processes deployed in the SDR file:

  NOTE: The `mysdl.xml` file is a NonStop SOAP 4 SDL file.

  ```
  OSS> SoapAdminCL -e mysdl.xml
  ```

- The following command generates the `mysdl.xml` file that includes information about the Pathway services deployed in the SDR file:

  ```
  OSS> SoapAdminCL -e mysdl.xml -pathway
  ```

- The following command generates the `mysdl.xml` file that includes information about the Pathway services running in the `$pmon1` and `$pmon2` `PATHMON`s deployed in the SDR file:

  ```
  OSS> SoapAdminCL -e mysdl.xml -pathway -env $pmon1 $pmon2
  ```

- The following command generates the `mysdl.xml` file that includes information about the NonStop processes deployed in the SDR file:

  ```
  SoapAdminCL -e mysdl.xml -process
  ```

- The following command generates the `mysdl.xml` file that includes information about the `$proc1` and `$proc2` processes deployed in the SDR file:

  ```
  OSS> SoapAdminCL -e mysdl.xml -process -env $proc1 $proc2
  ```

Additional Options

The `SoapAdminCL` tool provides additional options to perform the following activities:

- View the DTD of the SDL file using the following command:

```
OSS>SoapAdminCL -dtd
```

- Check the version of the SoapAdminCL tool using the following command:

```
OSS>SoapAdminCL -v | -version
```

- Invoke the command-line help of the SoapAdminCL tool using the following command:

```
OSS>SoapAdminCL -h | -help | -?
```

- Convert a transaction identifier from an external ASCII form to an internal form (\system-name(tm-flags).cpu.sequence).

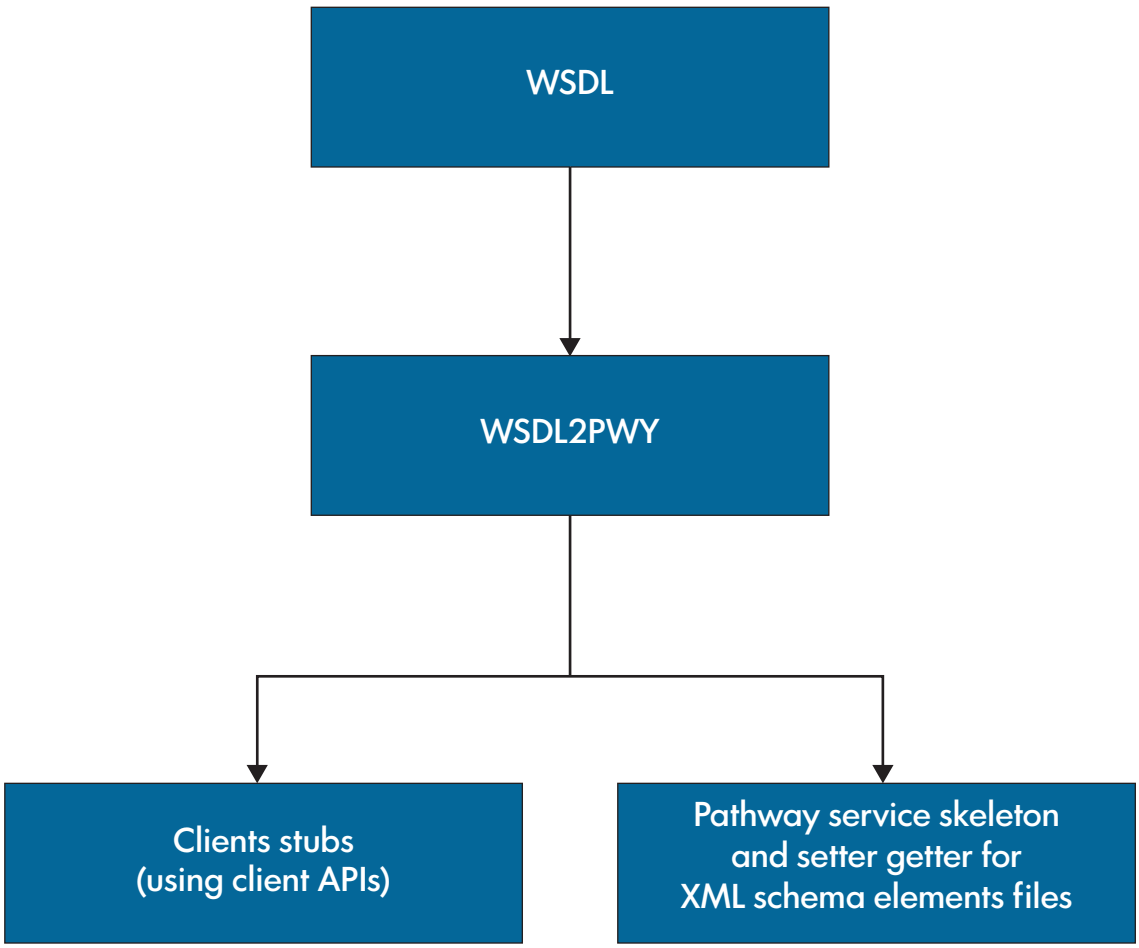
```
OSS>SoapAdminCL -t | -TMFTransactionId <Transaction identified in external ASCII format>
```

The WSDL2PWY Tool

The WSDL2PWY tool is a standalone Java-based command-line tool that runs in the OSS environment on NonStop systems. This tool uses a WSDL file as an input to generate client stubs and Pathway service skeleton files in the C programming language. The Pathway service skeleton files are used to create a Pathway application that can be deployed as a Web service in NonStop SOAP 4.

Figure 15 shows the files generated by the WSDL2PWY tool.

Figure 15 Files Generated by the WSDL2PWY Tool



This section describes the following topics:

- "Generating NonStop SOAP 4 Client Stubs" (page 204)
- "Generating Service Stubs" (page 205)
- "XML Schema and C Data Types Mapping" (page 206)

- "Migrating to Contract-First Services" (page 208)
- "WSDL2PWY Limitations" (page 210)

Generating NonStop SOAP 4 Client Stubs

The WSDL2PWY tool generates the NonStop SOAP 4 client stubs in the C programming language. The generated client stubs contain the interface code to be implemented when developing a client using NonStop SOAP 4 client APIs.

To generate the NonStop SOAP 4 client stubs using the WSDL2PWY tool, complete the following steps:

1. Set the OSS environment variable NSSOAP_HOME to the OSS location where the NonStop SOAP 4 installation directory is located:

   ```
   OSS> export NSSOAP_HOME=<NonStop SOAP 4 Installation Directory>
   ```

   For example:

   ```
   OSS> export NSSOAP_HOME=/usr/tandem/nssoap/t0865h01
   ```

   where,

   /usr/tandem/nssoap/t0865h01 is the NonStop SOAP 4 installation directory.

2. Add the directory containing the WSDL2PWY executable image to the OSS PATH variable.

   ```
   OSS> export PATH=<NonStop SOAP 4 Installation Directory>/tools:$PATH
   ```

   For example:

   ```
   OSS> export PATH=/usr/tandem/nssoap/t0865h01/tools:$PATH
   ```

3. Add the <*Java Installation Directory*>/bin directory to the PATH environment variable, using the command:

   ```
   OSS> export PATH=<Java Installation Directory>/bin:$PATH
   ```

   For example:

   ```
   OSS> export PATH=/usr/tandem/java/bin:$PATH
   ```

   where,

   /usr/tandem/java/
   is the Java installation directory.

4. Generate the client stubs using the WSDL2PWY command:

   ```
   OSS> WSDL2PWY [options] -uri [wsdl_path]
   ```

   where,

   [options] includes the following:

   -o <*output location*>
   specifies the output location where the client stubs and the Makefile (Makefile_client) must be generated.

   -a
   generates the client stubs to invoke a Web service in asynchronous or non-blocking mode. In this mode, the client sends a request to the Web service and continues processing without waiting for a response.

   -s
   generates the client stubs to invoke a Web service in synchronous mode or blocking mode. In this mode, the client sends a request to the Web service and waits for a response.

   -uri <*wsdl_path*>
   specifies the location of the WSDL file.

For example, use the following command to generate synchronous client stubs using the WSDL2PWY tool:

```
OSS> WSDL2PWY -o "/home/nssoap/test/client" -s
     -uri "/home/nssoap/test/services/test_service/SoapPW_test.wsdl"
```

where,

`/home/nssoap/test/client`
is the output location where the client stubs and the Makefile (`Makefile_client`) are generated. The client stubs are placed in the `src` directory in the output location. If the `-o` option is not specified, then the files are generated in the `src` directory under the current directory.

`-uri /home/nssoap/test/services/test_service/SoapPW_test.wsdl`
is a WSDL Web address provided to the WSDL2PWY tool.

5. On successful execution, the following files will be generated:
   - Client source stub file which needs to implement the main function called when an executable file is run.
   - The header file for the client source stub file that holds the declarations of the functions that are implemented in the client stub source file.

Generating Service Stubs

The WSDL2PWY tool generates the service stubs based on the WSDL definition. The generated service stubs are:

- Request and response structure definitions in C programming language.
- Code for TS/MP communication.
- Placeholders for the business logic function for every operation.

To generate the NonStop SOAP 4 service stubs using the WSDL2PWY tool, complete the following steps:

1. Set the OSS environment variable NSSOAP_HOME to the OSS location where the NonStop SOAP 4 installation directory is located:

   ```
   OSS> export NSSOAP_HOME=<NonStop SOAP 4 Installation Directory>
   ```

   For example:

   ```
   OSS> export NSSOAP_HOME=/usr/tandem/nssoap/t0865h01
   ```

   where,

   `/usr/tandem/nssoap/t0865h01`
   is the NonStop SOAP 4 installation directory.

2. Add the directory containing the WSDL2PWY executable image to the OSS PATH variable.

   ```
   OSS> export PATH=<NonStop SOAP 4 Installation Directory>/tools:$PATH
   ```

   For example:

   ```
   OSS> export PATH=/usr/tandem/nssoap/t0865h01/tools:$PATH
   ```

3. Add the <*Java Installation Directory*>/bin directory to the PATH environment variable, using the command:

   ```
   OSS> export PATH=<Java Installation Directory>/bin:$PATH
   ```

   For example:

   ```
   OSS> export PATH=/usr/tandem/java/bin:$PATH
   ```

   where,

   `/usr/tandem/java/`
   is the Java installation directory.

4. Generate the Pathway service skeleton files using the WSDL2PWY command:

   ```
   OSS> WSDL2PWY [options] -ss -uri [wsdl_path]
   ```

   where,

   [options] includes the following:

   -o <*output location*>
   specifies the output location where the Pathway service skeleton files must be generated.

   -ss
   generates the Pathway service skeleton files.

   -uri <*wsdl_path*>
   specifies the location of the WSDL file.

   For example, use the following command to generate the Pathway service skeleton files using the WSDL2PWY tool:

   ```
   OSS> WSDL2PWY -o "/home/nssoap/test/client" -ss
        -uri "/home/nssoap/test/services/test_service/SoapPW_test.wsdl"
   ```

   where,

   /home/nssoap/test/client
   is the output location where the Pathway service skeleton files are generated.

   -uri /home/nssoap/test/services/test_service/SoapPW_test.wsdl
   is a WSDL Web address provided to the WSDL2PWY tool.

5. On successful execution, The WSDL2PWY tool generates the following files for the service stubs::

   - DataConverter.c and DataConverter.h – include the utility functions to convert the XML value to C structure and vice-versa.
   - Msg<*messagename*>SetterGetter.c and Msg<*messagename*>SetterGetter.h – include the functions to set and get the value from C structure. The .h and .c files are generated for each message defined in the WSDL.
   - Svc<*servicename*>SetterGetter.c and Svc<*servicename*>SetterGetter.h – implement the interface functions, which identify the request and response structures for the operation.
   - pway<*servicename*>.c – includes the code to communicate with the NonStop SOAP 4 server.
   - pway<*servicename*>.h – includes the C structure definitions for the messages.
   - pway<*servicename*>Impl.c – includes the skeletons to implement the business logic. For each WSDL operation, you must add the business logic code in this file.

NOTE: If the -o option is not specified, the files are generated in the src directory under the current directory.

XML Schema and C Data Types Mapping

The following table lists the WSDL2PWY tool mapping of XML schema and C data types.

Table 14 XML schema to C data type mapping

Xml schema type	Sample schema	C data type mapping
anyType	Not Applicable	char *
anySimpleType	Not Applicable	char *

Table 14 XML schema to C data type mapping *(continued)*

Xml schema type	Sample schema	C data type mapping
Duration	<xs:element name="elapsedTime" type="xs:duration”/>	char *
dateTime	<xs:element name="orderDate" type="xs:datetime”/>	char *
time	<xs:element name="now" type="xs:time”/>	char *
date	<xs:element name="shipdate" type="xs:date”/>	char *
gYearMonth	<xs:element name="winterStart" type="xs:gYearMonth”/>	char *
gYear	<xs:element name="yy" type="xs:gYear”/>	int
gMonthDay	<xs:element name="salaryDay" type="xs:gMonthDay”/>	char *
gDay	<xs:element name="secSat" type="xs:gDay”/>	char *
gMonth	<xs:element name="fyResults" type="xs:gMonth”/>	char *
string	<part name="greeting" type="xsd:string" />	char *
normalizedString	<part name="greeting" type="xsd:normalizedString" />	char *
token	Not Applicable	char *
NMTOKEN	Not Applicable	char *
NMTOKENS	Not Applicable	char *
Name	Not Applicable	char *
NCName	Not Applicable	char *
ID	Not Applicable	char *
IDREF	Not Applicable	char *
IDREFS	Not Applicable	char *
ENTITY	Not Applicable	char *
ENTITIES	Not Applicable	char *
language	Not Applicable	char *
Boolean	<part name="say" type="xsd:boolean" />	short
base64Binary	<part name="noOfItemInbase64" type="xsd:base64Binary" />	char *
hexBinary	<part name="noOfItemInHex" type="xsd:hexBinary" />	char *

Table 14 XML schema to C data type mapping *(continued)*

Xml schema type	Sample schema	C data type mapping
float	<part name="cost" type="xsd:float" />	float
Decimal	<part name="numberOfItems" type="xsd:float" />	float
integer	<part name="numberOfItems" type="xsd:integer" />	int
nonPositiveInteger	<part name="numberOfItems" type="xsd:nonPositiveInteger" />	int
negativeInteger	<part name="numberOfItems" type="xsd:negativeInteger" />	int
nonNegativeInteger	<part name="numberOfItems" type="xsd:nonNegativeInteger" />	unsigned int
positiveInteger	<part name="numberOfItems" type="xsd:positiveInteger" />	unsigned int
unsignedLong	<part name="numberOfItems" type="xsd:unsignedLong" />	unsigned long long
unsignedInt	<part name="numberOfItems" type="xsd:unsignedInt" />	unsigned int
unsignedShort	<part name="numberOfItems" type="xsd:unsignedShort" />	unsigned short
long	<part name="numberOfItems" type="xsd:long" />	long long
int	<part name="numberOfItems" type="xsd:int" />	int
short	<part name="numberOfItems" type="xsd:short" />	short
byte	<part name="numberOfItems" type="xsd:byte" />	short
Double	<part name="precision" type="xsd:double" />	double
anyURI	<part name="myURI" type="xsd:anyURI" />	char *
QName	<part name="name" type="xsd:QName" />	char *
NOTATION	<part name="not" type="xsd:NOTATION" />	char *

Migrating to Contract-First Services

The existing service stubs must be regenerated to enable the unbounded data elements feature. The business logic and server configurations must be updated. The migration procedure is listed as follows:

1. For every operation, add the `axis2_pway_xml_receiver` message receiver in the `services.xml` file. For example:

```
<operation name="<operation name>">
          <messageReceiver class="axis2_pway_xml_receiver"/>
          ......
        ......
          </operation>
```

2. Run the `WSDL2PWY` tool to regenerate the service skeletons.

```
WSDL2PWY -o <output_directory> –ss –uri <wsdl_file_name>
```

3. Move the existing business logic code to the newly generated service skeletons. The `WSDL2PWY` tool generates a function for each operation to add the business logic. The signature of this function is changed and the earlier versions have the following signature:

```
void operation_pway_<operation_name>  (char *request_buffer,
char **response_buffer,
long *reply_count,
unsigned short count_read)
{
}
```

   The new function signature is:

```
void operation_pway_<operation_name>  (<req_structure_name> *reqStruct,
<response_structure_name> **respStruct,
<fault_struct_name> **faultStruct)
{
}
```

   The function `operation_pway_<operation_name>` is in `pway<ServiceName>Impl.c` file. In the earlier versions, this function is in `pway<ServiceName>.c` file.

   Move the `operation_pway_<operation_name>` implementation code from `pway<ServiceName>.c` file to the newly generated skeletons in `pway<ServiceName>Impl.c` file.

   NOTE: The newly generated skeletons have the sample implementation code, and you must remove them completely before you move the existing business logic code.

4. Modify the following in the business logic.
   - The field names in the newly generated structure definitions will not have `_type<numeric_number>` at the end. The field name change has considerable impact on the business logic, and the name must be changed wherever it is used in the code.
   - Dynamic arrays will not have an impact on how the data is accessed, but to find out the length of the array, use the `_size<array_name>` field. If there are any dynamic arrays in the response structure, explicitly allocate the memory and then set the value. Also, populate `_size<array_name>` field with the number of elements in the array.
   - Nested structures are removed and independent structures are created for each complex type in the WSDL. These independent structures are used as reference in other structures. Removal of nested structures does not have any impact on data access from the structure but to find the length of dynamic array, use `_size<array_name>` field.
   - For the function signature, see Step 3.
5. Compile the modified code.
6. Deploy the service in Pathway environment.

WSDL2PWY Limitations

The NonStop SOAP 4 WSDL2PWY tool is used to generate the Pathway service artifacts for contract-first development approach. This tool has some deviations from the WSDL specification and some limitations, which are discussed in this section.

Table 26 lists the XML schema definitions, which deviate from WSDL specification and WSDL2PWY tool behavior for these XML schema definitions.

Table 15 XML schema definitions, which deviate from WSDL specification

XML schema definitions	WSDL specification	WSDL2PWY behavior
Elements having complexContent and simpleContent	Elements cannot have simpleContent or complexContent	Tool parses the WSDL and generates element with type axiom_node_t *, which results in a compilation error
maxOccurs != 1 or minOccurs >1 for elements within <all>	maxOccurs must be 1 and minOccurs must be either 0 or 1 for elements within <all>	Generates either pointer (for unbounded data elements) or an array
Attributes having complexContent, simpleContent or complexType	Attributes can have only simpleType	Ignores such attributes without returning any error
Attributes along with complexContent orsimpleContent	Attributes along with complexContent or simpleContent are not allowed	Generates variables for attributes

The following XML schema definitions are part of the WSDL specification but are not supported by WSDL2PWY tool:

- <any> element type. For example,

```
<xsd:any minoccurs="0" maxoccurs="unbounded"/>
```

- Nested <group> or <choice> or <sequence> elements. For example,

```
<xsd:element name="ele1" type="xsd:string"/>
<xsd:element name="ele2" type="xsd:int"/>
    <xsd:group name="group1">
        <xsd:choice>
         <xsd:element name="ele3" type="xsd:string"/>
            <xsd:element name="ele4" type="xsd:string"/>
        </xsd:choice>
    </xs:group>
</xsd:sequence>
```

- Substitution group. For example,

```
<xs:element name="name" type="xs:string"/>
<xs:element name="navn" substitutionGroup="name"/>
```

- <extension> element. For example,

```
<xsd:extension base="xsd:decimal">
    <xsd:attribute name="my_attri" type="tns1:internationalPrice1" use="optional"/>
    <xsd:attribute name="currency" type="xsd:int" use="optional"/>
</xsd:extension>
```

The following parameters are not supported by contract-first development approach, but are supported by service-first development approach. In contract-first development approach, these parameters are ignored if they are present in services.xml file.

- stringTermination = NullTerminated/NonNULL
- responseType = strict/lenient

The WSDL2C Tool

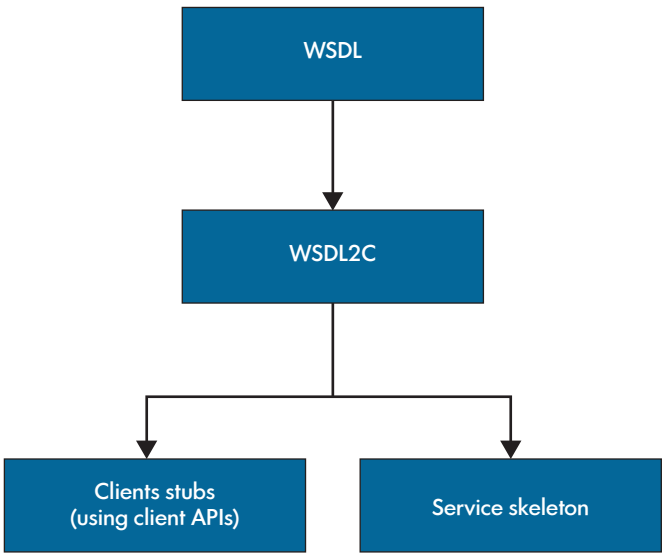
The WSDL2C tool is a standalone Java-based command-line tool that runs in the OSS environment on NonStop systems. This tool uses a WSDL file as an input to generate the client stubs and service skeleton files. The generated service skeleton files implement the basic interface code required while developing a service using NonStop SOAP 4 service APIs.

NOTE:

- The WSDL2PWY tool must be used when developing services implemented as Pathway applications.
- The WSDL2C tool must be used when developing services implemented as dynamic-link libraries using NonStop SOAP 4 service APIs.

Figure 16 shows the files generated by the WSDL2C tool.

Figure 16 Files Generated by the WSDL2C Tool



This section describes the following topics:

- "Generating NonStop SOAP 4 Client Stubs" (page 211)
- "Generating NonStop SOAP 4 Service Skeleton Files" (page 213)

Generating NonStop SOAP 4 Client Stubs

The WSDL2C tool generates the NonStop SOAP 4 client stubs in the C programming language. The generated client stubs contain the basic interface code to be implemented when developing a client using NonStop SOAP 4 client APIs.

To generate the NonStop SOAP 4 client stubs using the WSDL2C tool, complete the following steps:

NOTE: Ensure that the WSDL file for your service is located in <*NonStop SOAP 4 Deployment Directory*>. If the WSDL file is not present, use the SoapAdminCL tool to create a WSDL file for your service. For information on generating the WSDL file, see NonStop SOAP 4 Configuration Files (page 177).

1. Set the OSS environment variable NSSOAP_HOME to the OSS location where the NonStop SOAP 4 installation directory is located:

   ```
   OSS> export NSSOAP_HOME=<NonStop SOAP 4 Installation Directory>
   ```

   For example:

   ```
   OSS> export NSSOAP_HOME=/usr/tandem/nssoap/t0865h01
   ```

where,

`/usr/tandem/nssoap/t0865h01`

is the NonStop SOAP 4 installation directory location.

2. Add the directory containing the `WSDL2C` executable image to the OSS `PATH` variable.

   `OSS> export PATH=<NonStop SOAP 4 Installation Directory>/tools:$PATH`

   For example:

   `OSS> export PATH=/usr/tandem/nssoap/t0865h01/tools:$PATH`

3. Add the `<Java Installation Directory>/bin` directory to the `PATH` environment variable, using the command:

   `OSS> export PATH=<Java Installation Directory>/bin:$PATH`

   For example:

   `OSS> export PATH=/usr/tandem/java/bin:$PATH`

   where,

   `/usr/tandem/java/`

   is the Java installation directory.

4. Generate the client stubs using the `WSDL2C` command:

   `OSS>  WSDL2C [options] –u –uri [wsdl_path]`

   where,

   `-u`

   unpacks the databinding classes. This attribute must be specified if you use option `-d` and specify its value as “adb”. If you specify option `-d` as “none”, then `-u` becomes optional attribute.

   `-uri <wsdl_path>`

   specifies the location of the WSDL file.

   `[options]` include the following:

   `-d`

   specifies the data binding option to be used.

   `-o <output location>`

   specifies the output location where the client stubs and build scripts must be generated.

   `-a`

   generates the client stubs to invoke a Web service in asynchronous or non-blocking mode. In this mode, the client sends a request to the Web service and continues processing without waiting for a response.

   `-s`

   generates the client stubs to invoke a Web service in synchronous mode or blocking mode. In this mode, the client sends a request to the Web service and waits for a response.

   `-wv <version>`

   specifies the WSDL file version. The valid options are 2, 2.0, and 1.1.

   `-f`

   flattens the generated files.

   `-or`

   overwrites the existing files.

   NOTE: Do not use the `-a` and `-s` options together.

For example, use the following command to generate asynchronous client stubs using the WSDL2C tool:

```
OSS> WSDL2C -o "/home/nssoap/test/client" -a -u
     -uri "/home/nssoap/test/services/test_service/SoapPW_test.wsdl"
```

where,

`/home/nssoap/test/client`
is the output location.

`-uri /home/nssoap/test/services/test_service/SoapPW_test.wsdl`
specifies the WSDL Web address provided to the WSDL2C tool.

5. On successful execution, the following files will be generated:
   - Client source stub files to implement the SOAP client application business logic.
   - Header files that hold the declarations for the functions generated in the client source stub files.
   - Data binding header files that hold declarations for the data structures present in the WSDL schema.

Generating NonStop SOAP 4 Service Skeleton Files

The WSDL2C tool generates the NonStop SOAP 4 service skeleton files in the C programming language. The generated service skeleton files contain the basic interface code to be implemented when developing services implemented as DLLs using NonStop SOAP 4 service APIs.

To generate the NonStop SOAP 4 service skeleton files using the WSDL2C tool, complete the following steps:

1. Set the OSS environment variable NSSOAP_HOME to the OSS location where the NonStop SOAP 4 installation directory is located:

   ```
   OSS> export NSSOAP_HOME=<NonStop SOAP 4 Installation Directory>
   ```

   For example:

   ```
   OSS> export NSSOAP_HOME=/usr/tandem/nssoap/t0865h01
   ```

   where,

   `/usr/tandem/nssoap/t0865h01`
   is the NonStop SOAP 4 installation directory.

2. Add the directory containing the WSDL2C executable image to the OSS PATH variable.

   ```
   OSS> export PATH=<NonStop SOAP 4 Installation Directory>/tools:$PATH
   ```

   For example:

   ```
   OSS> export PATH=/usr/tandem/nssoap/t0865h01/tools:$PATH
   ```

3. Add the <*Java Installation Directory*>/bin directory to the PATH environment variable, using the command:

   ```
   OSS> export PATH=<Java Installation Directory>/bin:$PATH
   ```

   For example:

   ```
   OSS> export PATH=/usr/tandem/java/bin:$PATH
   ```

   where,

   `/usr/tandem/java/`
   is the Java installation directory.

4. Generate the service skeleton files using the WSDL2C command:

   ```
   OSS> WSDL2C [options] -u -uri [wsdl_path]
   ```

   where,

[options] include the following:

-o<*output location*>
specifies the output location where the service skeleton files must be generated.

-g
generates the client stubs and the service skeleton files. This option can be used only if the -ss option is specified.

-f
flattens the generated files.

-or
overwrites the existing files.

-wv <*version*>
specifies the WSDL file version.

-d <*databinding_option*>
specifies that the databinding option must be used. NonStop SOAP 4 provides the axisdatabinding (adb) databinding option to generate the service skeleton files in the C programming language. If you do not want to use the databinding option, use the -d none option.

NOTE: Databinding is an option provided by NonStop SOAP 4 that helps the XML schema parsing process. By default, the WSDL2C tool develops services with the databinding option set to axisdatabinding (adb).

-ss
generates the service skeleton files.

-uri <*wsdl_path*>
specifies the location of the WSDL file.

For example, use the following commands to generate the service skeleton files:

```
OSS> export NSSOAP_HOME=/usr/tandem/nssoap/t0865h01
OSS> WSDL2C -o "/home/nssoap/test/client" -ss -d adb
-uri "/home/nssoap/test/services/test_service/SoapPW_test.wsdl"
```

where,

/home/nssoap/test/client
is the output location.

-uri /home/nssoap/test/services/test_service/SoapPW_test.wsdl
specifies the WSDL Web address provided to the WSDL2C tool.

5. On successful execution, the following service skeleton files are generated under the src directory in the location specified in the -o option of the WSDL2PWY command:
   - Service skeleton source file to implement your application business logic.
   - Header file that holds the declarations for the functions generated in the service skeleton source file.
   - Data binding header files that hold declarations for the data structures present in the WSDL schema.

NOTE: If the -o option is not specified, the files are generated in the src directory under the current directory.

The NonStop SOAP 4 Administration Utility

The NonStop SOAP 4 Administration Utility is a graphical user interface (GUI) based utility that runs in the Microsoft Windows environment. This utility allows you to perform the following tasks:

- Create new NonStop SOAP 4 SDL files.
- Deploy new services on the NonStop SOAP 4 server using an existing SDL file.

  When deploying a Web service, the following files are deployed in *<NonStop SOAP 4 Deployment Directory>*:

  - Service directory for the new NonStop SOAP 4 Web service
  - Service configuration file (`services.xml`)
  - HTML clients
  - WSDL file (`.wsdl`)
- Migrate NonStop SOAP 3 services to NonStop SOAP 4.

  Migrating a NonStop SOAP 3 Web service to NonStop SOAP 4 involves the following tasks:

  1. Creating a NonStop SOAP 4 SDL file using an existing NonStop SOAP 3 SDR.
  2. Deploying the SDL file on the NonStop system.

NOTE: The NonStop SOAP 4 Administration Utility does not generate or deploy the NonStop SOAP 3 services.

Multiple instances of the NonStop SOAP 4 Administration Utility (T0904) supports multiple releases of NonStop SOAP 4. These instances can be started by executing the `AdminTool.bat` multiple times on the Windows system. A single instance of the SOAP Administration utility can be configured for only a single SOAP 4 release.

Installing NonStop SOAP 4 Administration Utility

The NonStop SOAP 4 Administration Utility is distributed in the `T0904.zip` file.

To install the NonStop SOAP 4 Administration Utility on a Windows system, complete the following steps:

1. Extract the `T0904.zip` file to the standard OSS directory (`/usr/tandem/nssoap/t0904`) by completing the following steps:
   a. Receive the product files from the disk (distribution subvolume (DSV) locations) or tape.
   b. In the DSM/SCM planner interface, select the `Manage OSS Files` option for the target configuration.

      NOTE: If you do not select the `Manage OSS Files` option in the DSM/SCM planner interface, DSM/SCM will place the `T0865PAX` file in the Guardian subvolume `$ISV.ZOSSUTL` (where, `ISV` is the installation volume). Use the `COPYOSS` command to extract and place the contents of the `T0904.zip` file into the OSS file system.

   c. Copy the received product files to a new software revision of the configuration you want to update.
   d. Run the `Build` request and the `Apply` request on the configuration revision.
   e. Run ZPHIRNM to rename the product files.
2. Copy (or transfer using FTP) the `T0904.zip` file to a location on your Windows system.
3. Extract the contents of the `T0904.zip` file.

4. Double-click the `AdminTool.bat` file (`\T0904\SoapAdmin`). The NonStop SOAP 4 Administration Utility screen appears.

   The NonStop SOAP 4 Administration Utility is now ready for use.

   NOTE: For information on using the NonStop SOAP 4 Administration Utility, see the NonStop SOAP 4 Administration Utility online help integrated with the NonStop SOAP 4 Administration Utility.

11 NonStop SOAP 4 Features

This chapter provides information about the following features of NonStop SOAP 4:

- "DDL Comments" (page 217)
- "Hot-Deployment of the NonStop SOAP 4 Server" (page 229)
- "Hot-Update for the Deployed Services" (page 229)
- "Internationalization and Encoding" (page 230)
- "Communicating with a Non-Pathway Process" (page 231)
- "Validation Module" (page 233)
- "SOAP_WSDL_NAME DDL Comment" (page 233)
- "Support for Multiple DDL Definitions" (page 234)
- "Check on SOAP Service Deployment" (page 235)
- "Unbounded data elements support" (page 235)

DDL Comments

NonStop SOAP 4 allows you to flag the DDL fields in the DDL file using DDL comments. The DDL comments feature enables you to modify the request and response SOAP message structures at runtime. The comment tags should be the first set of nonwhite space character following the * character (which denotes DDL line as a comment). There could be spaces or tabs following the * character. If there are more than one comment tags, then the tags should be separated by a space and adjacent to each other.

NOTE: The DDL Comments must not be applied for a DDL Definition Name. For example:

```
*  @SOAP_OPTIONAL @SOAP_SUPPRESS_IN          DEFINITION REQ.
```

This is wrong and should be avoided.

"Comment Tags" describes the effect of specifying two comment tags. The order in which the comment tags are specified is not significant.

Table 16 Comment Tags

Comment Tag 1	Comment Tag 2	Effect
@SOAP_OPTIONAL	@SOAP_SUPPRESS_IN	@SOAP_SUPPRESS_IN
@SOAP_OPTIONAL	@SOAP_SUPPRESS_OUT	Optional for a request message, suppressed for the output message
@SOAP_OPTIONAL	@SOAP_SUPPRESS_INOUT	@SOAP_SUPPRESS_INOUT
@SOAP_SUPPRESS_IN	@SOAP_SUPPRESS_OUT	@SOAP_SUPPRESS_INOUT
@SOAP_SUPPRESS_IN	@SOAP_SUPPRESS_INOUT	@SOAP_SUPPRESS_INOUT
@SOAP_SUPPRESS_OUT	@SOAP_SUPPRESS_INOUT	@SOAP_SUPPRESS_INOUT

NOTE: For more information about @SOAP_SUPPRESS_IN @SOAP_SUPPRESS_OUT and @SOAP_SUPPRESS_INOUT comment tags refer to (page 164)

Before you begin using the DDL comments feature, please ensure the following:

- For the DDL comment tags to be parsed by the `SoapAdminCL` tool, the DDL file must include the `?comments` complier directive.

  For example:

```
?comments
DEF FLD1.
02  FLD11.
03  FLD12.
*@SOAP_ATTRIBUTE
04 FLD13 PIC X(20).
END
```

- The `Operation` tag in the SDL file must have the `ProcessSoapDDLComments` attribute set to `yes`.

  For example:

```
<Operation
            Name="<operation Name>"
            TMFTransactionSupport="[yes | no]"
            AbortTransactionOnFault="[yes | no]"
            NameSpaceQualified="[yes | no]"
            SoapMessageType="[document | rpc]"
            ProcessSoapDDLComments="yes"
            ...>
```

You can proceed to the following tasks:

- "Specifying DDL Fields as Optional" (page 218)
- "Exposing DDL Fields as XML Attributes" (page 221)
- "Specifying `Base64` Encoding" (page 225)
- "Using the `SOAP_OCCURS_DEP_ON` DDL Comment Tag" (page 226)

Specifying DDL Fields as Optional

NonStop SOAP 4 allows you to flag a DDL field as optional, using the `@SOAP_OPTIONAL` DDL comment tag. If this tag is set for a group field in the DDL file, the entire hierarchy of the group field is treated as optional.

In Example 5, the `HOSPITAL` group field is tagged as optional (the associated children fields are also treated as optional).

Example 5 A Sample DDL file with the `@SOAP_OPTIONAL` Tag

```
?comments
    DEF PATIENT.
   02 PATIENT-DATA.
    03 ID PIC 9(5).
    03 AGE PIC 9(3).
    * @SOAP_OPTIONAL
    03 HOSPITAL.
       04 HOSP-NAME PIC X(30).
    04 ADDRESS.
       05 STREET PIC X(50).
       05 CITY PIC X(25).
       05 STATE PIC X(2).
       05 ZIP PIC 9(5).
    04 COUNTY.
    05 CODE PIC 9(6).
             05 Details.
                  06 COUNTY-NAME PIC X(15).
                  06 STATUS PIC X.
             05 DESCRIPTION PIC X(100).
 END
```

Example 6 shows the corresponding WSDL file generated using the `SoapAdminCL` tool. In the WSDL file and the XML schema file, the highlighted attribute entry `minOccurs ="0"` specifies that the corresponding fields are optional.

Example 6 An XSD Schema for a Sample DDL file with the @SOAP_OPTIONAL Tag

```
<xsd:complexType name="patient_data">
<xsd:sequence>
<xsd:element name="id" type="xsd:long" minOccurs="1" maxOccurs="1"/>
<xsd:element name="age" type="xsd:long" minOccurs="1" maxOccurs="1"/>
<xsd:element name="hospital" type="tns:hospital" minOccurs="0"
maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="hospital">
<xsd:sequence>
<xsd:element name="hosp_name" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
<xsd:element name="address" type="tns:address" minOccurs="0"
maxOccurs="1"/>
<xsd:element name="county" type="tns:county" minOccurs="0"
maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="address">
<xsd:sequence>
<xsd:element name="street" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
<xsd:element name="city" type="xsd:string" minOccurs="0" maxOccurs="1"/>
<xsd:element name="state" type="xsd:string" minOccurs="0" maxOccurs="1"/>
<xsd:element name="zip" type="xsd:long" minOccurs="0" maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="county">
<xsd:sequence>
<xsd:element name="code" type="xsd:long" minOccurs="0" maxOccurs="1"/>
<xsd:element name="details" type="tns:details" minOccurs="0"
maxOccurs="1"/>
<xsd:element name="description" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="details">
<xsd:sequence>
<xsd:element name="county_name" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
<xsd:element name="status" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>
```

Because the `hospital` node and all its leaf nodes are specified as optional in the DDL file, the `hospital` element and all its child elements also have the XSD schema attribute `minOccurs` set to 0 in the WSDL file and the XML schema file.

Example 7 and Example 8 show valid XML message representations based on the XML schema shown in Example 6.

Example 7 An XML Message-I for a Sample DDL file with the @SOAP_OPTIONAL Tag

```
<patient_data>
  <id></id>
  <age></age>
</patient_data>
```

In Example 7, the hospital element is not present; however, it is valid because the HOSPITAL field is optional.

Example 8 An XML Message-II for a Sample DDL file with the @SOAP_OPTIONAL Tag

```
<patient_data>
  <id></id>
  <age></age>
  <hospital>
    <hosp_name></hosp_name>
    <county>
      <code></code>
      <details>
        <county_name></county_name>
        <status></status>
      </details>
      <description></description>
    </county>
  </hospital>
</patient_data>
```

In Example 8, the ADDRESS element is not present; however, it is valid because the parent element is hospital, which is an optional element.

Exposing DDL Fields as XML Attributes

Each DDL field in the DDL file is mapped to a corresponding element in the XML schema. As a result of the mapping, the size of the SOAP messages exchanged between the client and the NonStop SOAP 4 server increases and impacts the runtime performance.

Example 9 A Sample XML file with DDL fields mapped to the Corresponding Elements

```
<AccountDetails>
    <AccountNumber>120012</AccountNumber>
    <MemberName>
        <First>John</First>
        <Middle>K</Middle>
        <Last>Doe</Last>
    </MemberName>
</AccountDetails>
```

NonStop SOAP 4 allows you to specify a DDL field as an XML attribute, which helps reduce the size of the SOAP message.

Example 10 A Sample XML file with DDL fields represented as XML attributes

```
<AccountDetails AccountNumber="120012">
     <MemberName First="John" Middle="K" Last="Doe"/>
</AccountDetails>
```

The DDL fields can be represented in XML as attributes or elements by doing one of the following:

- "Setting the SoapDDLAttribute in the SDL File" (page 222)
- "Flagging a DDL Field with SOAP DDL Comment Tags" (page 222)

Setting the SoapDDLAttribute in the SDL File

The SDL file provides SoapDDLAttribute at the Service element level that accepts a value of yes or no. This enables you to flag all the DDL fields of a definition file without changing the DDL dictionary.

- When set to yes, SoapDDLAttribute specifies that all the leaf fields of the request and response DDL definitions of the service must be represented as XML attributes.
- When set to no, SoapDDLAttribute specifies that all the leaf fields of the request and response DDL definitions of the service must be represented as XML elements.
- All DDL leaf fields, except the OCCURS and OCCURS_DEP_ON fields, are XML attributes.

The default value is no.

Example 11 shows a sample SDL file with SoapDDLAttribute.

Example 11 A Sample SDL File with SoapDDLAttribute

```
<sdl ...>
   <Pathway ...>
   ...
      <ServerClass Name="SC-1" ...>
         <Service ServiceName="Service-A"
            SoapMessageType="document"
            SoapDDLAttribute="yes">
         </Service>
      </ServerClass>
   </Pathway>
</sdl>
```

The SoapDDLAttribute attribute is not applicable for services when the SoapMessageType attribute in the SDL file is set to rpc because the message format in the rpc style resembles a function call.

In this message format, the order of the elements is important, which cannot be achieved when the message fields are represented as XML attributes. Therefore, do not set the SoapDDLAttribute to yes if a service in the SDL file has the SoapMessageType attribute set to rpc. The SoapAdminCL tool validates this condition when adding or updating a service. Any validation errors encountered are displayed as follows:

```
SOAPADMIN ERROR >> Error in definition of the service <service name>.
                  The attribute "SoapDDLAttribute" cannot have a value "yes"
                  when the attribute "SoapMessageType" is set to "rpc".
```

Flagging a DDL Field with SOAP DDL Comment Tags

DDL fields can be selectively represented as XML attributes or elements by tagging them appropriately in the DDL dictionary. You can use the following DDL comment tags to flag the DDL fields in the request/response definitions as attributes or elements:

@SOAP_ATTRIBUTE

Flags a DDL field(s) to be represented as XML attribute(s).

@SOAP_ELEMENT

Flags a DDL field(s) to be represented as XML element(s).

For example, a DDL file with the field tagged with the @SOAP_ATTRIBUTE will appear as:

```
?comments
DEF PATIENT.
02 PATIENT.
03 NAME.
*@SOAP_ATTRIBUTE
04 FIRST PIC X(20).
END
```

The presence of the @SOAP_ATTRIBUTE comment tag at the leaf level denotes that the field must be translated to an XML attribute. If the @SOAP_ATTRIBUTE tag is present at the group level, it denotes that all the child leaf fields must be translated to XML attributes.

The @SOAP_ELEMENT comment tag also works in the same manner as the @SOAP_ATTRIBUTE comment tag for the leaf and group levels, but translates the field(s) as XML elements, not as XML attributes.

NOTE: A service that uses a DDL file with the @SOAP_ELEMENT and @SOAP_ATTRIBUTE comment tags must have its ProcessSoapDDLComments attribute set to yes in an SDL file for NonStop SOAP 4 to process the comment tags.

To customize the XML representation of the DDL fields, the @SOAP_ELEMENT and @SOAP_ATTRIBUTE comment tags can be combined and matched within the same group level based on the following rules:

- A single DDL field cannot have both the @SOAP_ATTRIBUTE and @SOAP_ELEMENT comment tags present.
- A service using a DDL file with the @SOAP_ATTRIBUTE or @SOAP_ELEMENT comment tags must have its ProcessSoapDDLComments attribute set to yes in an SDL file for NonStop SOAP 4 to process the comment tags.
- If the comments appear on a particular field, it is valid for the entire hierarchy under that field (if any) unless overridden by another tag at a child level. The default value is controlled by the value of the SoapDDLAttribute attribute set in the SDL file.
- If the SoapDDLAttribute attribute is present and is set to yes, by default, all the leaf fields are treated as attributes, unless a comment pertaining to a DDL hierarchy changes it (provided that the ProcessSoapDDLComments attribute is set to yes).
- The @SOAP_ATTRIBUTE and @SOAP_ELEMENT comment tags are ignored for services that have the SoapMessageType attribute set to rpc.

Examples

Example 12 shows the @SOAP_ATTRIBUTE and @SOAP_ELEMENT comment tag at the leaf and group levels.

Example 12 A Sample DDL File with the @SOAP_ATTRIBUTE and @SOAP_ELEMENT Tag

```
?comments
DEF PATIENT.
   02 PATIENT-DATA.
      * @SOAP_ATTRIBUTE
         03 ID PIC 9(5).
         03 AGE PIC 9(3).
      * @SOAP_ATTRIBUTE
         03 NAME.
            04 FIRST PIC X(20).
            04 MIDDLE PIC X.
            04 LAST PIC X(25).
      * @SOAP_ATTRIBUTE
         03 HOSPITAL.
            04 HOSP-NAME PIC X(30).
      * @SOAP_ELEMENT
            04 ADDRESS.
               05 STREET PIC X(50).
               05 CITY PIC X(25).
               05 STATE PIC X(2).
               05 ZIP PIC 9(5).
          04 COUNTY.
      * @SOAP_ATTRIBUTE
               05 CODE PIC 9(6).
      * @SOAP_ELEMENT
               05 COUNTY-NAME PIC X(15).
      * @SOAP_ATTRIBUTE
               05 STATUS PIC X.
      * @SOAP_ELEMENT
               05 DESCRIPTION PIC X(100).
END
```

Example 13 shows the XML message and schema representation corresponding to the DDL file.

Example 13 An XML Message for a Sample DDL file with the @SOAP_ATTRIBUTE and @SOAP_ELEMENT Tag

```
<patient_data id="12345">
   <age>55</age>
   <name first="John" middle=" " last="Smith"></name>
   <hospital hosp_name="Valley Hospital">
      <address>
         <street>365 Hospital Dr.</street>
         <city>San Jose</city>
         <state>CA</state>
         <zip>95129</zip>
      </address>
   <county code="012345" status="L">
   <county_name>Santa Clara</county_name>
   <description></description>
   </county>
   </hospital>
</patient_data>
```

Because the `first`, `middle`, and `last` leaf fields are flagged with the `@SOAP_ATTRIBUTE` comment tag in the DDL file, they are represented as XML attribute values for the XML element `<name>`. The leaf nodes of the `address` node are also flagged with the `@SOAP_ELEMENT` comment tag in the DDL file and are represented as XML elements.

Table 17 lists the results of the XML representation of a DDL file based on the associated SDL values and DDL comment tags.

Table 17 SDL Values and DDL Comment Tags

SOAPDDLAttribute Value	ProcessSoapDDL Comments Value	DDL Comment tag present?	Effect
Yes	No	Not Applicable	All the DDL leaf fields, except OCCURS and OCCURS_DEP_ON leaf fields are XML attributes.
Yes	Yes	No	All the DDL leaf fields are XML attributes.
Yes	Yes	Yes	The DDL fields overridden with a comment tag are handled appropriately. All the other fields are XML attributes.
No	Yes	Yes	Only the fields containing the comment tags are processed.

Specifying Base64 Encoding

A SOAP message might need to carry binary data or other characters that are considered invalid by the W3C XML specifications. To represent the binary data or invalid XML characters as valid XML content, the data must be transformed or encoded appropriately.

The NonStop SOAP 4 server enables the client to send encoded binary data or invalid XML characters by using the Base64 encoding feature. The Base64 encoding feature, overrides any other Request, Service, or Response encoding. To apply the Base64 encoding for an XML attribute or element, in the DDL file, flag the @SOAP_BASE64 DDL comment tag.

The Base64 encoding was introduced by the Multipurpose Internet Mail Exchange (MIME) standard (IETF MIME RFC) as a method for transmitting data that was binary in nature or contained values representing control characters that would have undesired effects on the transmission. The Base64 encoding consists of 64 encoding characters (A-Z, a-z, 0-9, +, /) that are a subset of the ASCII system and are considered valid in the XML specification.

The @SOAP_BASE64 comment tag can be present only at the DDL leaf level. The SoapAdminCL tool validates this before creating the WSDL file. If this tag is present for a group field, it is flagged as an error and the following error message is displayed:

```
SOAPADMIN ERROR >> Error processing DDL definition <definition name> for service
                  <service name>. The "@SOAP_BASE64" DDL comment tag cannot be
                  present at a group level DDL field <field name>.
```

Example 14 shows the @SOAP_BASE64 comment tag in the DDL file, the corresponding sample XML message containing the Base64 data, and the XML schema representation with the Base64 schema type.

Example 14 A Sample DDL file with the @SOAP_BASE64 Tag

```
?comments
DEF PATIENT.
02 PATIENT-DATA.
03 ID PIC 9(5).
03 AGE PIC 9(3).
03 NAME.
04 FIRST PIC X(20).
04 MIDDLE PIC X.
04 LAST PIC X(25).
* @SOAP_BASE64
03 BINARY-CONTENT type binary 32.
END
```

In the DDL file, the BINARY-CONTENT field is tagged as @SOAP_BASE64. This is indicated by the element type xsd:base64Binary in the XML schema present in the WSDL file and the XML schema file (shown in Example 15), created by the SoapAdminCL tool.

Example 15 An XML Schema for a Sample DDL file with the @SOAP_BASE64 Tag

```
<xsd:complexType name="name">
<xsd:sequence>
<xsd:element name="first" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
<xsd:element name="middle" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
<xsd:element name="last" type="xsd:string" minOccurs="1"
maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>
<xsd:complexType name="patient_data">
<xsd:sequence>
<xsd:element name="id" type="xsd:long" minOccurs="1" maxOccurs="1"/>
<xsd:element name="age" type="xsd:long" minOccurs="1" maxOccurs="1"/>
<xsd:element name="name" type="tns:name" minOccurs="1" maxOccurs="1"/>
<xsd:element name="binary_content" type="xsd:base64Binary"
minOccurs="1" maxOccurs="1"/>
</xsd:simpleType>
<xsd:restriction base="xsd:base64Binary">
<xsd:maxLength value="4">
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
```

Example 16 is a valid XML representation based on the XML schema shown in Example 15.

Example 16 XML Message for a DDL file with the @SOAP_BASE64 Tag

```
<patient_data>
   <id>12345</id>
   <age>55</age>
   <name>
      <first>John</first>
      <middle></middle>
      <last>Smith</last>
   </name>
   <binary_content>m6Lp</binary_content>
</patient_data>
```

The information passed in the binary_content tag is processed by NonStop SOAP 4 as Base64 Encoded data.

Using the SOAP_OCCURS_DEP_ON DDL Comment Tag

NonStop SOAP 4 allows you to flag a DDL field as SOAP_OCCURS_DEP_ON (occurs depending on), which declares that a field or a group is repeated a variable number of times depending on the value of another DDL field.

Example 17 shows that the occurrence of the `fiction-details` element in the SOAP message depends on the value of the `fiction_count` element.

Example 17 A Sample DDL file with the `@SOAP_OCCURS_DEP_ON` Tag

```
?comments
DEF books-in-lib.
02 fiction_count PIC 9(4) COMP.
02 libraries_fic PIC 9(4) COMP.
02 magazines_count PIC 9(4) COMP.
02 libraries_mag PIC 9(4) COMP.
* @SOAP_OCCURS_DEP_ON fiction_count
02 fiction-details OCCURS 1000 TIMES.
03 call-num PIC 9(10).
END
```

Among other valid field types for the integer variable, the following are frequently used:

- TYPE BINARY 16,2
- TYPE BINARY 32
- TYPE BINARY 32, UNSIGNED
- PIC 9
- PIC S9
- PIC 9(10)
- PIC 9(4) COMP
- PIC S9(5)
- PIC S9(5) COMP

Some of the invalid field types are:

- PIC X
- PIC X(10)
- PIC 9999V99
- TYPE FLOAT
- TYPE FLOAT 64
- TYPE LOGICAL 4

Example 18 shows the DDL file that shows the use of the `SOAP_OCCURS_DEP_ON` comment.

Example 18 A Sample DDL file with the @SOAP_OCCURS_DEP_ON Tag

```
?comments

  DEFINITION REQ.
     10  CH1  type character 2.
     10  Bin32 type binary 32.
     10  Complex1.
        15  CH2   type    character 4.
        15  CH3   type    character   1.
        15  Bin2  type binary 32.
        15  Complex2.
            20 custnum PIC X(20) UPSHIFT.
* @SOAP_OCCURS_DEP_ON Bin2
            20 Bin3 type binary 32 OCCURS 5 times.
  END
```

Example 19 shows the corresponding XML schema generated by the SoapAdminCL tool.

Example 19 An XML Schema for a Sample DDL file with the @SOAP_OCCURS_DEP_ON Tag

```
<xsd:complexType name="complex2">
 <xsd:sequence>
  <xsd:element name="custnum" minOccurs="1" maxOccurs="1">
   <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:maxLength value="20"/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:element>
  <!-- The following element occurs 0 to 5 times, depending upon the
       value of the element:bin2:urn:cpq_tns_reflector-->
  <xsd:element name="bin3" type="xsd:int" minOccurs="0" maxOccurs="5"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="complex1">
 <xsd:sequence>
  <xsd:element name="ch2" minOccurs="1" maxOccurs="1">
   <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:maxLength value="4"/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:element>
  <xsd:element name="ch3" minOccurs="1" maxOccurs="1">
   <xsd:simpleType>
    <xsd:restriction base="xsd:string">
     <xsd:maxLength value="1"/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:element>
  <xsd:element name="bin2" type="xsd:int" minOccurs="1" maxOccurs="1"/>
  <xsd:element name="complex2" type="tns:complex2" minOccurs="1" maxOccurs="1"/>
 </xsd:sequence>
</xsd:complexType>
```

Example 20 shows the corresponding SOAP message body that the NonStop SOAP 4 server expects from the client.

Example 20 The SOAP Message Body for a Sample DDL file with the `@SOAP_OCCURS_DEP_ON` Tag

```
<complex1>
<ch2>abc</ch2>
<ch3>d</ch3>
<bin2>3</bin2>
<complex2>
<custnum>10</custnum>
<bin3>1</bin3>
<bin3>2</bin3>
<bin3>3</bin3>
</complex2>
</complex1>
```

In Example 20, the `<bin2>` element in the `<complex1>` structure dictates the number of `<bin3>` occurrences in the `<complex2>` structure in accordance with the `@SOAP_OCCURS_DEP_ON` DDL comment.

Hot-Deployment of the NonStop SOAP 4 Server

NonStop SOAP 4 server allows you to deploy Web services while it is operational, running, and serving requests. The process of deploying Web services while the NonStop SOAP 4 server is actively serving requests is known as Hot-Deployment. This feature enables you to deploy new Web services in your NonStop SOAP 4 installation without bringing down the NonStop SOAP 4 server.

The NonStop SOAP 4 server scans all the services under the services directory of your NonStop SOAP 4 deployment and adds the new services in the service repository.

When a client sends a request to the NonStop SOAP 4 server, the NonStop SOAP 4 server checks the service repository and loads the services listed in the service repository. Because the service is added to the service repository during Deployment, the DLL loader can locate the requested service and process the client request.

The Hot-Deployment approach avoids reconfiguration downtime for businesses because the NonStop SOAP 4 server immediately makes the service available (that is, without restarting the iTP WebServer).

The Hot-Deployment feature is enabled by default. To disable this feature, set the `hotdeployment` attribute to `false` in the `axis2.xml` file. For example, the following entry in the `axis2.xml` file turns off the Hot-Deployment feature:

```
<parameter name=”hotdeployment” locked="false”>false< /parameter>
```

NOTE: The Hot-Deployment feature is applicable only for services and not for modules.

Hot-Update for the Deployed Services

The NonStop SOAP 4 server enables you to update configuration or definition of a Web service while it is still operating, running, or serving requests. This feature enables you to make changes to Web services in your NonStop SOAP 4 installation without bringing down the NonStop SOAP 4 server or the service.

When a client sends a request to the NonStop SOAP 4 server, the NonStop SOAP 4 server checks the modification date of the `services.xml` file and the WSDL file. If either of these files are modified, the NonStop SOAP 4 server loads the information into the service repository, which is then used for current and subsequent requests, unless a change occurs. However, the modification date check adds overhead to the SOAP message processing, therefore this feature is disabled by default.

To enable this feature, set the `hotupdate` attribute to `true` in `axis2.xml` file. For example, the following entry in `axis2.xml` file enables this feature:

```
<parameter name=”hotupdate” locked="false”>true< /parameter>
```

NOTE: The Hot-Update feature is applicable only for services and not for modules or the `axis2.xml` configuration.

Internationalization and Encoding

The NonStop SOAP 4 server supports encoding for SOAP messages. You can specify the encoding for incoming request XML, outgoing response XML and for the TS/MP service data. The encoding module manages the encoding, and is distributed with the NonStop SOAP 4 server. The default encoding is UTF-8, unless otherwise specified. By default, this module is enabled by specifying the following entry in the `axis2.xml` configuration file:

```
<module ref=”encoding”/>
```

NOTE: For specifying an encoding other than UTF-8, set the `ICU_DATA` environment variable to point to installed ICU data location in the `itp_axis2.config` file:

```
Server  $Axis2c {
...
Env ICU_DATA=/usr/tandem/xml/T0563H01/lib/icu/data/
...
}
```

The NonStop SOAP 4 server supports the following encodings:

- UTF-8 (default encoding)
- ISO-2022-JP
- Other single byte encodings supported by the ICU library

NOTE: The NonStop SOAP 4 server does not support UTF-16 or any multi byte encoding as the target service encoding or response encoding. They can still be specified as a SOAP request encoding. A request or response message with an encoding other than UTF-8 has an XML declaration node (`< ?xml version="1.0" encoding="encoding-name"?>`) as the first node in the SOAP XML document.

Configuring the request encoding

To specify a request encoding other than UTF-8, set the encoding attribute of the XML declaration element to the desired encoding.

For example, the following sample specifies a request XML encoding as Shift_JIS:

```
<?xml version="1.0" encoding=”Shift_JIS” ?>
<SOAP_ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body
xmlns:admin="urn:compaq_nsk_oss_soapquery">
         <admin:ListServices serviceType="all"/>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Configuring the response encoding

To set the encoding for SOAP response messages, use one of the following options.

1. Specify the encoding of individual responses in the encoding element of the SOAP request message header. For example, the following SOAP header specifies ISO-2022-jp as the response message encoding:

   ```
   <SOAP_ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header xmlns:nskhdr="urn:compaq_nsk_oss_EncodingHeader">
   ```

```
<nskhdr:Encoding OutputEncoding="ISO-2022-JP"/>
</SOAP-ENV:Header>
<SOAP-ENV:Body xmlns:admin="urn:compaq_nsk_oss_soapquery">
        <admin:ListServices serviceType="all"/>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2. Specify the default encoding for the SOAP response messages from a target service using one of these two options.
   - Set the `RspEncoding` attribute in the SDL file at design time and run the `SOAPAdminCL` tool to generate the required files.

     NOTE: For this option, you must set the`ICU_DATA` environment variable to the location of the ICU data library before running the `SOAPAdminCL` tool. For example,

     ```
     OSS> export ICU_DATA=/usr/tandem/xml/T0563H01/lib/icu/data/
     ```

   - Set the operation level attribute `RspEncoding`, which will be used at runtime, in the `services.xml` file. For example, the following entry sets the encoding to ISO-2022–JP:

     ```
     <parameter name="RspEncoding">ISO-2022-JP</parameter>
     ```

If the `RspEncoding` attribute is set in the `services.xml` file and `OutputEncoding` is set in the encoding header, `OutputEncoding` overrides `RspEncoding`. If the `RspEncoding` attribute is not set in the `services.xml` file and `OutputEncoding` is not set in the encoding header, the response message encoding is the same as request message encoding.

Configuring the server encoding

You can configure the server encoding by using one of these two options:

- Set the `SrvrEncoding` attribute in the SDL file at design time and run `SOAPAdminCL` tool to generate the required files.

  NOTE: For this option, you must set the`ICU_DATA` environment variable to the location of the ICU data library before running the `SOAPAdminCL` tool. For example,

  ```
  OSS> export ICU_DATA=/usr/tandem/xml/T0563H01/lib/icu/data/
  ```

- Set the service level attribute `SrvrEncoding`, which will be used at runtime, in the `services.xml` file.

the `SrvrEncoding` attribute in the SDL file or by modifying the `services.xml` file. For example, the following entry sets the encoding to ISO-2022–JP:

```
<parameter name="SrvrEncoding">ISO-2022-JP</parameter>
```

Communicating with a Non-Pathway Process

NonStop SOAP 4 can expose a non-Pathway process as a Web service. This section describes the configuration of NonStop SOAP 4 to enable it to communicate with a process.

Configuring NonStop SOAP 4 to communicate with a non-Pathway process involves the following tasks:

- “Creating an SDL File to Communicate with a Non-Pathway Process” (page 232)
- “Creating a DDL File with the Request/Response Structures” (page 232)
- “Generating NonStop SOAP 4 Files using the `SoapAdminCL` Tool” (page 232)

Creating an SDL File to Communicate with a Non-Pathway Process

The SDL file has a different hierarchy for a service, which is provided by a non-Pathway process. The new element `<process>` appears under the root `<sdl>` element. The DTD schema for the SDL file appears as:

```
<!ELEMENT sdl  (Pathway | Process | Enscribe | Tuxedo | XmlSql)+>
<!ELEMENT Process (ProcessEnvironment)+>
<!ELEMENT ProcessEnvironment (ProcessDetails)>
<!ATTLIST ProcessEnvironment Name CDATA #REQUIRED>
<!ELEMENT ProcessDetails (Service)+>
<!ATTLIST ProcessDetails Name CDATA #IMPLIED language (C | COBOL) "COBOL"
stringTermination (NullTerminated | NonNull) "NonNull" DDLDictionaryLocation
CDATA #REQUIRED SrvrEncoding CDATA "UTF-8">
```

For more information about the SDL file structure and its attributes, see "NonStop SOAP 4 Service Description Language" (page 153).

The sample SDL file for the process communication appears as:

```
<sdl>
  <Process>
      <ProcessEnvironment Name="$echo">
        <ProcessDetails Name="Reflector"
           DDLDictionaryLocation="$FC3.AXTE1"
           SrvrEncoding="UTF-8"
           language="C"
           stringTermination="NullTerminated">
          <Operation  Name="REFLECTOR"
             TMFTransactionSupport="yes"
             AbortTransactionOnFault="yes"
             NameSpaceQualified="yes"
             SoapMessageType="document"
             ProcessSoapDDLComments="no"
             SoapDDLAttribute="no"
             UseDDLDefaultValue="no">
             <OperationDescription> Reflector </OperationDescription>
             <RequestInfo>
                <DDLDefinitionName>INPUT</DDLDefinitionName>
             </RequestInfo>
             <ResponseInfo>
               <DDLDefinitionName>INPUT</DDLDefinitionName>
               <ResponseSelection defaultResp="yes"  />
             </ResponseInfo>
          </Operation>
        </ProcessDetails>
      </ProcessEnvironment>
  </Process>
</sdl>
```

Creating a DDL File with the Request/Response Structures

The request and response message structures for the process must be defined in the DDL file. The location of the DDL dictionary and other details of the process must be specified within the appropriate attributes of the `ProcessDetails` element.

The name attribute of the `ProcessDetails` element has a maximum length of 64 characters. If a value greater than the maximum length is specified, the characters after the 64th character are ignored. The remaining attributes in the process SDL hierarchy are similar to their counterparts in the Pathway hierarchy.

Generating NonStop SOAP 4 Files using the `SoapAdminCL` Tool

After the DDL and SDL files are created, run the `SoapAdminCL` tool to generate the NonStop SOAP 4 files. For information on using the `SoapAdminCL` tool to generate NonStop SOAP 4 files, see "NonStop SOAP Tools" (page 194).

Validation Module

The SOAP messages sent by the clients must be validated against the WSDL schema to ensure that correct data is received. The NonStop SOAP 4 validation module validates the received data. The validation module is a global module, and it is enabled by default.

All the service requests to the NonStop SOAP 4 server are validated against the service WSDL schema by default. If the requests confirm to the schema, then the NonStop SOAP 4 server processes the requests. If the requests do not confirm to the schema, then the validation module sends back a `Module Validation Failed` message to the client, and does not process the request. For more information about the failure, see the log files in the `<NonStop SOAP 4 Deployment Directory>/logs` folder.

The `Global Modules` section of `axis2.xml` file contains the configuration for the validation module. To turn off this feature, comment the line in `axis2.xml`file.

```
<!-- ================================================= -->
<!-- Global Modules -->
<!-- ================================================= -->
!-- <module ref="validation”/> -->
```

NOTE: Even if the validation module is turned off, the NonStop SOAP 4 server validates for most of the data type values. However, the validations for XML constructs are not very strict.

The validation module feature impacts performance. The first request for a service takes a longer time than the subsequent requests as the schema has to be loaded into the memory for the first request. The time taken by the validation module to process the request is directly proportional to the size of the request (that is the number of XML elements in the request). In cases where the service response time is extremely important, to gauge the overhead caused by the module, HP recommends measuring the NonStop SOAP 4 server response times with and without this feature. Decide to enable or disable this feature, based on these measurements.

SOAP_WSDL_NAME DDL Comment

The `SoapAdminCL` generates the WSDL file and DDL mapping file to display the TS/MP application as a Web service. You can add a comment in DDL file that defines the WSDL file name to be used in the WSDL file for a field.

The following example describes the use of the SOAP_WSDL_NAME DDL a comment in the DDL definition and the generated WSDL and DDL mapping files:

DDL Definition

```
DEFINITION INPUTDEF
 02 A1 PIC X(1)
*@SOAP_WSDL_NAME wsdlnamea2
 02 A2  PIC X(4)
END
```

The following example displays a generated sample WSDL file:

```
<xsd:element name="REFLECTOR" type="tns1:REFLECTOR"/>
<xsd:complexType name="REFLECTOR">
<xsd:sequence>
<xsd:element name="a1" minOccurs="1" maxOccurs="1">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="1"/>
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
<xsd:element name="wsdlnamea2" minOccurs="1" maxOccurs="1">
<xsd:simpleType>
<xsd:restriction base="xsd:string">
<xsd:maxLength value="4"/>
```

```
</xsd:restriction>
</xsd:simpleType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
```

For this conversion, NonStop SOAP 4 server uses `DDLMapping.xml` file to convert the request data to character buffer and response character buffer to XML. The server uses data definition structures stored in `DDLMapping.xml` file and their qualified names as a key for mapping.

The considerations for using comment names in DDL Definition are:

1. More than one alternate name for the same field cannot be used in the DDL definition file.
2. You must use the DDL field name as a reference along with SOAP DDL comment.

Support for Multiple DDL Definitions

You can use multiple DDL definitions in a single SOAP request or response. You can implement this feature by specifying multiple `<DDLDefinitionName>` elements in a `<RequestInfo>` or `<ResponseInfo>` element in the SDL file.

The following example illustrates the use of separate DDL definitions A1 and A2:

DDL Definition:

```
DEFINITION A1
     02 A11 PIC X(10000)
     02 A12 PIC X(20000)
END
DEFINITION A2
     02 A21 PIC X(10000)
     02 A22 PIC X(20000)
END
```

To specify that the Inter Process Communication (IPC) contains both definitions, A1 and A2, one after another, specify each of them by using `<DDLDefinitionName>` elements in the `<RequestInfo>` element:

```
<RequestInfo>
            <DDLDefinitionName>A1</DDLDefinitionName>
            <DDLDefinitionName>A2</DDLDefinitionName>
</RequestInfo>
```

The generated SOAP message contains the following XML structure:

```
<Req>
   <A1>
       <A11>temp</A11>
       <A12>temp</A12>
  </A1>
  <A2>
      <A21>temp</A21>
      <A22>temp</A22>
     </A2>
</Req>
```

NOTE: The same `<DDLDefinitionName>` element must not be used more than once in a single `<RequestInfo>` or `<ResponseInfo>` element. For example, the following sample implementation is incorrect:

```
<RequestInfo>
      <DDLDefinitionName>A1</DDLDefinitionName>
      <DDLDefinitionName>A1</DDLDefinitionName>
</RequestInfo>
```

Check on SOAP Service Deployment

The `SoapAdminCL` tool uses DDL Dictionaries and the SDL as an input to deploy a service under the SOAP server. In SDL DTD, the SDL element contains two attributes that denote the location of the SOAP server.

```
<sdl Url="/axis2c"> ServerAddress = "http://15.146.233.43:3001">
```

In NonStop SOAP 4 server, the `SoapAdminCL` tool detects whether the server is running at the defined location and also generates a warning message if the server is not running at the defined location.

You can trigger this facility by providing *-w* option along with *-i* and *-o* options to the SoapAdminCL tool.

Unbounded data elements support

The NonStop SOAP 4 server supports unbounded data elements for Contract-First SOAP services. The C services can process data greater than 2 MB with unbounded data elements. For enabling the unbounded data elements feature, configure the message receiver (`axis2_pway_xml_receiver`) in the `services.xml` file.

Earlier versions of NonStop SOAP 4 server convert the XML message in the SOAP body to a character buffer, and dispatch the message to a TS/MP service as shown in "NonStop SOAP 4 Request Processing Flow for TS/MP or NonStop process-based Web Services" (page 34). This conversion causes buffer size variations for unbounded XML, which cannot be processed by the TS/MP service. Also, the data cannot be greater than 2 MB.

For more information, see "The `WSDL2PWY` Tool" (page 203).

12 Transaction Management

As a value-added feature, NonStop SOAP 4 provides comprehensive support for Web services that implement transactions. A transaction, in this context, is defined as a series of logical operations against a database resulting from the receipt of one or more SOAP messages. All database changes must be done in their entirety, or all must be backed-out, to ensure that the database is in a consistent state once the transaction is complete.

NonStop SOAP 4 is fully integrated into the HP Transaction Management Facility (TMF) product, which is used to ensure database integrity. To use the transaction management feature of NonStop SOAP 4, TMF must be installed and running on your NonStop system. For more information about TMF, see the *TMF Reference Manual*.

You can perform a simple logical transaction by invoking an operation requested by a single SOAP message, or a more complex logical transaction can result from many SOAP requests that can span multiple invocations of the same or different operations within any service (a multi-step transaction).

For a simple transaction, the boundary of the transaction is confined to the processing of a single SOAP message; it starts with the invocation of the operation and ends when the operation is complete and control returns back to NonStop SOAP 4. Thus, NonStop SOAP 4 can begin and end the transaction as a part of processing that SOAP message; that is, SOAP Server Transaction Management. NonStop SOAP 4 requires an indication that the operation requires transaction protection, which can be done by setting the `TMFTransactionSupport` attribute in the `services.xml` configuration file (for information on the `TMFTransactionSupport` attribute, see "Configuring the Level of Transaction Support" (page 245)). For a multi-step (or complex) transaction, NonStop SOAP 4 requires some help from the SOAP client to indicate that a particular SOAP message is starting a transaction, and to subsequently signal when that transaction should be ended; that is, SOAP Client Transaction Management.

All the semantics for the SOAP Client Transaction Management are contained in a Transaction Header Block that may be sent in a SOAP message. The client can set the values of a `Command` attribute to start or end a transaction, and indicate that an existing transaction should be continued or aborted. In addition, the client can use another attribute to control the transaction timeout values. The `TransactionID` attribute coordinates between the client and NonStop SOAP 4, and is set to a unique value by NonStop SOAP 4 when a transaction is started. The `TransactionID` attribute is subsequently used as a handle that is passed back and forth between the client and server to associate SOAP messages to particular transactions.

The following attributes can be set in the `services.xml` file to define the kind of transaction support requested of the NonStop SOAP 4 server:

- `TMFTransactionSupport`
- `AbortTransactionOnFault`
- `TMFTimeout`

NOTE:

- For information on the `TMFTransactionSupport`, `AbortTransactionOnFault`, and `TMFTimeout` attributes, see "Configuring the Level of Transaction Support" (page 245) and "Transaction Timeouts" (page 246).
- Previous versions of NonStop SOAP (SOAP 3 and earlier) defined the need to contain a multi-step transaction within a session, which was defined in a session header block in the SOAP message. With NonStop SOAP 4, sessions are no longer required (which enables transaction management to be greatly simplified). NonStop SOAP 4 will continue to process session attributes and commands for compatibility with previous releases, and the syntax and semantics of this mechanism is described in "Session Management and Transactions" (page 249).

  There is an intent to discontinue the session construct in a future release of NonStop SOAP, though no date has been set for this action.

This chapter describes the following topics:

- "Transaction Module Configuration" (page 237)
- "SOAP Server Transaction Management" (page 238)
- "SOAP Client Transaction Management" (page 238)
- "Configuring the Level of Transaction Support" (page 245)
- "Transaction Timeouts" (page 246)
- "Session Management and Transactions" (page 249)

NOTE: It must be remembered that NonStop SOAP 4 runs as a TS/MP server class, so throughout this chapter and elsewhere in this manual, when the term SOAP server is used it is not referring to a single process instance.

Transaction Module Configuration

To enable NonStop SOAP 4 to recognize and correctly process the transaction (and session) header blocks in the SOAP request header, you must configure it to include the Transaction Module that is part of the product distribution.

To configure the transaction module in NonStop SOAP 4, complete the following steps:

1. Open the *<NonStop SOAP 4 Deployment Directory>*`/axis2.xml` file and locate the following commented section:

   ```
   <!-- ============================ -->
       <!-- Global Modules  -->
   <!-- ============================ -->
   ```

2. After the `Global Modules` section, add the `module` element and set its `ref` attribute value to `transaction`.

   ```
   <module ref="transaction"/>
   ```

   NOTE: When defining the `module` element, do not add a space between < and the `module` element.

3. Save and close the `axis2.xml` file.
4. Restart the iTP WebServer.

   ```
   OSS>./<iTP WebServer Deployment Directory>/conf/restart
   ```

The transaction module is now attached with your NonStop SOAP 4 deployment and can intercept each request and perform the requested transaction related operations.

SOAP Server Transaction Management

The NonStop SOAP 4 server can manage a simple transaction without a Transaction header block in the SOAP message. A client can invoke a service on the NonStop server and be certain that all the changes made to the database by that service are completed in their entirety or, should the transaction fail, the database restored to a prior state, using a single SOAP message.

When the NonStop SOAP 4 server receives a SOAP message that does not contain a Transaction header block (and before invoking the requested operation), it examines the setting of the `TMFTransactionSupport` attribute for this operation in the appropriate `services.xml` file. If the value is set to `Required`, the SOAP server will start a transaction through TMF and then continue with the service request. When the operation is complete and the service returns control back after executing the transaction, NonStop SOAP 4 will invoke TMF to commit the transaction before replying with a SOAP message back to the client.

If the call to the service fails, or the operation itself returns a fault indication, the SOAP server will abort the transaction, and the client will be notified that the operation failed.

NOTE: The `AbortTransactionOnFault` attribute has no effect on the feature being described here – the SOAP server will always abort the transaction as a result of an error when it is managing the transaction irrespective of the setting of `AbortTransactionOnFault`, if present for this operation.

For information on the `AbortTransactionOnFault` attribute, see "Configuring Transactions to Abort on Fault" (page 248)e.

It is possible to set a timeout for this transaction using the optional `TMFTimeout` attribute in the `services.xml` file. If not set, the default `TMFAutoAbort` timeout will apply. For information on the `TMFAutoAbort` attribute, see "Transaction Timeouts" (page 246).

SOAP Client Transaction Management

NonStop SOAP 4 allows clients to control when transactions are started, continued, and ended through values set for the `Command` attribute in a Transaction header block in the SOAP message. Based on these values, the SOAP server calls TMF to begin, resume, commit, or abort transactions before and following the invocation of one or more service operations indicated by the SOAP request. The Transaction header block can be defined in the WSDL file for a service by setting the service-level `GenerateTransactionHeader` attribute to `yes` in the SDL file for the service. After the transaction header block is defined in the WSDL file, tools that generate client code from the WSDL file (such as, `wsimport`) will create routines to access the transaction header block fields.

In addition to the `Command` attribute, there are two other important attributes used in the Transaction Header:

- `TransactionID`
- `TimeOut`

Table 18 lists the attributes in the transaction header element and their values.

Table 18 Attributes in the Transaction Header Block

Attributes of the Header Element	Values	Description
`Command`	`Begin`, `Continue`, `Commit`, `Abort`	Specifies the transaction command. Note that it is case-sensitive.
`TransactionID`	`TMF TransactionID`	Contains the value set by TMF to identify a specific transaction.
`TimeOut`	`Transaction timeout`	Specifies the transaction timeout in seconds. **NOTE:** The `TimeOut` attribute can be used only when the `Command` attribute in the transaction header block of the SOAP request header element is set to `Begin`.

The `TimeOut` attribute can be used to provide more granular control of the `TimeOut` value to be used by TMF before it automatically aborts a transaction and frees any resources (for example, database locks) that might be held by that transaction. For more information on the timeout settings, see "Transaction Timeouts" (page 246).

The value passed in the `TransactionID` attribute is a TMF transaction ID set by NonStop SOAP 4 in a Transaction header block contained in the SOAP response message. This attribute value is then sent by the SOAP client in the Transaction header block of subsequent SOAP requests as a handle to be used by NonStop SOAP 4 to associate messages received to a specific active transaction.

⚠ **WARNING!** The client must not alter the value of this attribute, or else the transaction will almost certainly fail.

During the processing of a multi-step transaction, the SOAP messages submitted by a client will probably be processed by different SOAP server instances. The only thing that associates messages to any particular logical transaction is the `TransactionID` attribute carried in a Transaction header block. Before a response is sent back to a client, the SOAP server instance suspends the active transaction. This makes the server available to receive another message from another or the same client to do work on behalf of another transaction; that is, each SOAP server instance is context-free – it retains no knowledge of any particular transaction.

It also follows from this that a client can also be involved simultaneously in multiple transactions. For a GUI application that is used by a human operator at a workstation, this probably does not make sense. However, for a complex application requester in a Services Oriented Architecture (SOA) accessing services on the NonStop server through NonStop SOAP, it is perfectly possible for it to be managing multiple parallel transactions and use the `TransactionID` attribute as the handle for each of its processing threads.

The following sections will detail how each of the four transaction commands (`Begin`, `Continue`, `Commit`, `End`) are processed by NonStop SOAP 4 with examples of the Transaction header block sent by the client and responses back from SOAP for each command.

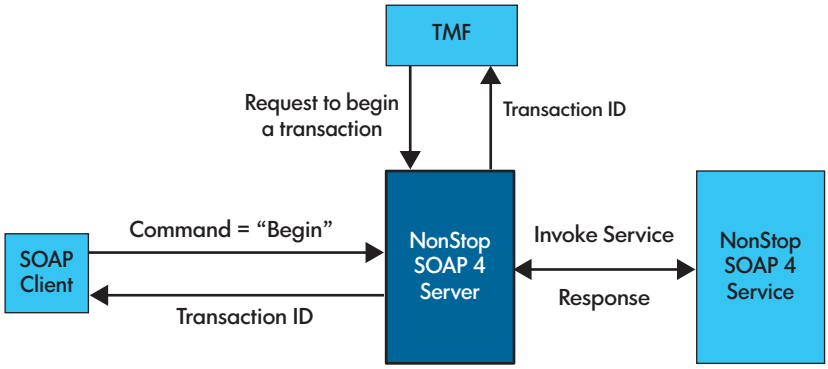
Begin a New Transaction

To begin a new TMF transaction, the client must set the value of the `Command` attribute (in the transaction header block of the NonStop SOAP 4 request header element) to `Begin` and send it as a part of the SOAP request header element.

NOTE: The body of a request that starts a transaction can invoke a target service, or it can be empty.

Figure 17 shows the schematic flow for beginning a new transaction.

Figure 17 Beginning a New Transaction



When beginning a new TMF transaction, NonStop SOAP 4 performs the following activities:

1. The NonStop SOAP 4 server begins a new TMF transaction on behalf of a client request and optionally sets a value for the transaction timeout in its call to TMF. The timeout value used depends on whether the client has set the `TimeOut` attribute in the SOAP request header block, and also if there is a value set for the `TMFTimeout` attribute in the `services.xml` configuration file.

   For more information about TMF transaction timeout, see "Transaction Timeouts" (page 246).

2. If the SOAP message body contains a request, NonStop will invoke it under the new TMF transaction. When the service returns control back to NonStop SOAP, or if the body of the request is empty, NonStop SOAP suspends the new TMF transaction and returns the TMF transaction ID in the `TransactionID` attribute in a Transaction header block in the SOAP response message back to the client. This value can then be used by the client in a subsequent request to continue the transaction. The value of the `TransactionID` attribute is in an external ASCII form. To convert the value to the internal form displayed in TMFCOM, use the `SoapAdminCL` tool with the `-t` option.

   For information on the `SoapAdminCL` tool, see "NonStop SOAP Tools" (page 194).

3. If the invoked service returns a fault and the `AbortTransactionOnFault` attribute is set to `no`, the NonStop SOAP server suspends the TMF transaction (it does not abort the transaction) and returns the TMF transaction ID in the `TransactionID` attribute in a Transaction header block along with a SOAP fault. This allows the SOAP client to decide the appropriate error handling and control whether the transaction should be aborted, or the operation retried.

   For more information about configuring the `AbortTransactionOnFault` attribute, see "Configuring Transactions to Abort on Fault" (page 248).

   NOTE: The NonStop SOAP 4 server recognizes that the application has returned a fault using the Response Selection Criteria specified in the `services.xml` file. The NonStop SOAP 4 server upon receiving the response buffer from the application, processes it based on the response selection criteria to derive the appropriate response structure to be used. Therefore, when the NonStop SOAP 4 server deduces that the matching response structure is indeed a Fault response, it prepares a SOAP fault envelope and returns it back to the client.

   For example, in case of the `EmpInfo` operation of the `employee_db` service, the service returns a fault when it is queried for a non-existent employee. In such a scenario, the response buffer's first two bytes carry the value 01. The corresponding response selection criteria, specified in the `services.xml` file, instructs the NonStop SOAP 4 server to select the Fault response structure when the first two bytes of the response buffer are 01. The NonStop SOAP 4 server then prepares a fault response.

4. If the invoked service returns a fault and the `AbortTransactionOnFault` attribute is set to `yes`, then the NonStop SOAP 4 server aborts the transaction and returns a SOAP fault to the client. The NonStop SOAP 4 server generates a temporary session identifier, because the current `SessionID` is no longer valid. The NonStop SOAP 4 server returns the session identifier in the session block header of the response message and sets this value for the `SessionID` attribute.

Example 21 shows a request/response pair of messages to begin a new transaction.

Example 21 Begin a New Transaction

```
Client Request
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_TransactionHeader">
<soapenv:Header>
  < urn:Transaction Command="Begin"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
 .
</soapenv:Body>
</soapenv:Envelope>

NonStop SOAP 4 Response

<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_TransactionHeader">
<soapenv:Header>
   <urn:Transaction TransactionID="234567"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
 .
</soapenv:Body>
</soapenv:Envelope>
```

NOTE: The response includes the TMF transaction ID that will be used in all subsequent requests associated with the transaction.

Continue a Transaction

To perform an operation under an active TMF transaction, the client must include a Transaction header block that sets the Command attribute to Continue and sets the value of the `TransactionID` attribute to the value of the `TransactionID` attribute provided by a previous response from NonStop SOAP 4 associated with this transaction.

On receiving a SOAP request to perform an operation under an active transaction:

1. The SOAP server resumes the TMF transaction specified in the `TransactionID` attribute of the transaction header block in the SOAP request header.
2. The operation specified in the body of the SOAP message is then invoked under the resumed TMF transaction ID.
3. When the service responds back to NonStop SOAP 4, the TMF transaction is suspended once again, the transaction ID value is set in the `TransactionID` attribute of the Transaction header block, and the response message is sent back to the client.
4. If the invoked service returns a fault and the `AbortTransactionOnFault` attribute is set to `yes`, then the NonStop SOAP 4 server aborts the transaction and returns a SOAP fault to

the client. The NonStop SOAP 4 server generates a temporary session identifier because the current `SessionID` is no longer valid. The NonStop SOAP 4 server returns the session identifier in the session block header of the response message and sets this value for the `SessionID` attribute.

5. If the invoked service returns a fault and the `AbortTransactionOnFault` attribute is set to `yes`, NonStop SOAP 4 aborts the transaction and returns a SOAP fault to the client. The SOAP message does not contain a Transaction header block because there is no `TransactionID` to return back to the client.
6. This series of request/response actions can continue until the client decides that the transaction needs to be ended with either a commit or an abort request.

Example 22 shows a request/response pair of messages to continue a transaction.

Example 22 Continue a Transaction

```
Client Request
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_TransactionHeader">
<soapenv:Header>
   <urn:Transaction Command="Continue" TransactionID="234567"/>
</soapenv:Header>
<soapenv:Body>
.
.
.
</soapenv:Body>
</soapenv:Envelope>

NonStop SOAP 4 Response

<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_TransactionHeader">
<soapenv:Header>
   <urn:Transaction TransactionID="234567"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
 .
</soapenv:Body>
</soapenv:Envelope>
```

where,

`234567`
: is the value assigned to the `TransactionID` attribute that was provided in a previous response by NonStop SOAP for this transaction.

Commit a Transaction

To commit an active TMF transaction, the client must include a Transaction header block that sets the `Command` attribute to `Commit` and sets the value of the `TransactionID` attribute to the value of the `TransactionID` attribute provided by a previous response from NonStop SOAP 4 associated with this transaction.

NOTE: If the SOAP request invokes a target service, the operation in the request will be executed as a part of the active TMF transaction.

When the SOAP server receives a request to commit an active TMF transaction, the following occurs:

1. The SOAP server resumes the TMF transaction specified in the `TransactionID` attribute of the transaction header block of the SOAP request.
2. The SOAP server invokes the operation (if specified in the body of the request) under the resumed TMF transaction.
3. When the service responds back to NonStop SOAP 4, a commit transaction request is sent to TMF and the SOAP response is sent back to the client. The response message will not contain a transaction header block because there is no `TransactionID` value to communicate back to the client.
4. If the service returns a fault and the `AbortTransactionOnFault` attribute is set to `no`, the transaction is not aborted. The `TransactionID` is sent back to the client in a Transaction header block in the response message along with the fault returned by the operation. This allows the client to decide whether to abort the transaction, or retry the operation.
5. If the service returns a fault and the `AbortTransactionOnFault` attribute is set to `yes`, the SOAP server aborts the transaction and returns the SOAP fault to the client. The response message will not contain a transaction header block because there is no `TransactionID` value to communicate back to the client.

Example 23 shows a request/response pair of messages to commit a transaction.

Example 23 Commit a Transaction

```
Client Request
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_TransactionHeader">
<soapenv:Header> <
   urn:Transaction TransactionID="234567" Command="Commit"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
 .
</soapenv:Body>
</soapenv:Envelope>


NonStop SOAP 4 Response
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_TransactionHeader">
<soapenv:Header>
</soapenv:Header>
<soapenv:Body>
 .
 .
 .
</soapenv:Body>
</soapenv:Envelope>
```

where,

`234567`
: is the identifier of the transaction to be committed, and was provided in a previous response from NonStop SOAP.

Abort a Transaction

To abort an active TMF transaction, the client must set the value of the `TransactionID` attribute (in the transaction header block of the NonStop SOAP request header element) to the value of the `TransactionID` attribute provided by a previous response from NonStop SOAP 4 associated with that transaction, and set the `Command` attribute to `Abort`.

NOTE: HP recommends that the body of a request that aborts a transaction be empty. The sequence of the following events indicates that the transaction is aborted before any service invocation. This means that there will not be an active transaction when the service is invoked.

When a message is received by the SOAP server to abort a message, the following occurs:

1. The SOAP server resumes the TMF transaction specified in the `TransactionID` attribute of the Transaction header block of the SOAP request header element.
2. The SOAP server issues an abort transaction request to the TMF subsystem.
3. If the SOAP message body requests a target service, it will be invoked, but there will not be any active TMF transaction, because it has been aborted in step 2.
4. NonStop SOAP 4 returns the response from the service. The response message will not contain a transaction header block because there is no `TransactionID` value to communicate back to the client.

Example 24 shows a request/response pair of messages to abort a transaction.

Example 24 Abort a Transaction

```
Client Request
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_TransactionHeader">
<soapenv:Header>
   <urn:Transaction TransactionID="234567" Command="Abort"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
 .
</soapenv:Body>
</soapenv:Envelope>


NonStop SOAP 4 Response

<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_TransactionHeader">
<soapenv:Header>
</soapenv:Header>
<soapenv:Body>
 .
 .
 .
</soapenv:Body>
</soapenv:Envelope>
```

where,

`234567`
: is the identifier of the transaction to be aborted, and was provided in a previous response from NonStop SOAP.

Configuring the Level of Transaction Support

You can configure the level of transaction management support for services deployed in NonStop SOAP 4 using the `TMFTransactionSupport` attribute. The `TMFTransactionSupport` attribute is configured at the operation level.

The `TMFTransactionSupport` attribute can be set to one of the following values:

- `Required`

  All operations in the SOAP request must be performed within a TMF transaction.

  - If the client request does not include a transaction header block in the SOAP request header, NonStop SOAP 4 begins a TMF transaction before invoking the service and subsequently commits or aborts the transaction once a response is received from the service. For more information, see "SOAP Server Transaction Management" (page 238).
  - If the client request does include a transaction header block in the SOAP request header, NonStop SOAP proceeds to examine the value of the `Command` attribute in the transaction header block and processes the TMF transaction. For information on the `Command` attribute, see "SOAP Client Transaction Management" (page 238).

- `Supports`

  Operations in the SOAP request may be performed within a TMF transaction.

  - If the client request does not include a transaction header block in the SOAP request header, no TMF transaction calls will be made and the service will be invoked outside any TMF transaction. For example, a `browse` or `get` operation does not need to be performed inside a transaction.
  - If the client request includes a transaction header block in the SOAP request header, NonStop SOAP 4 proceeds to examine the value of the `Command` attribute in the Transaction header block and proceeds to perform the requested action. For more information on the `Command` attribute, see "SOAP Client Transaction Management" (page 238).

- `Never`

  Operations in the SOAP request must not be performed within a TMF transaction. If the client request includes a transaction header block in the SOAP request header element, NonStop SOAP 4 will return a SOAP fault response.

NOTE: Previous versions of NonStop SOAP used values for the `TMFTransactionSupport` attribute of `yes` and `no`. When migrating from SOAP 3 to SOAP 4, these values will be mapped by the `SoapAdminCL` tool to `Required` and `Never` respectively.

To configure the level of transaction support for your service, complete the following steps:

1. Open the `services.xml` configuration file located in *<NonStop SOAP 4 Deployment Directory>*`/services/`*<service_name>*.

   where,

   <service_name> is the name of the service for which you want to set the level of transaction management.

2. Set the value of the `TMFTransactionSupport` attribute to either `Required`, `Supports`, or `Never`.

```
<operation>
  .
  .
TMFTransactionSupport="[Required | Supports | Never]"
  .
```

```
.
</operation>
```

3. Save and close the `services.xml` configuration file.

The `TMFTransactionSupport` attribute is an optional attribute. The default value of the `TMFTransactionSupport` attribute is `Supports`. Thus, if a client request includes a transaction header block in the SOAP request header, then NonStop SOAP 4 will process the Command attribute contained within it, even if the `TMFTransactionSupport` attribute is not set in the `services.xml` file.

NOTE:

- When developing a service the `TMFTransactionSupport` attribute can be set as an operation level attribute in the SDL file.
- When deploying a service using the `SoapAdminCL` tool, the `services.xml` configuration file will have the `TMFTransactionSupport` attribute set to a value specified in the SDL file.

  For more information about setting the `TMFTransactionSupport` attribute in the SDL file, see "NonStop SOAP 4 Service Description Language" (page 153).

Transaction Timeouts

During the processing of a transaction, it is typical for resources in the database to be locked (for example, a table row, or even a complete table) until the transaction is committed (or aborted) to ensure data integrity. For a multi-step transaction, this can pose a problem in the event that the client that started the transaction does not continue or end the transaction in a reasonably short timeframe. There could be many reasons, including loss of communication between the client and the NonStop server, or it could be as simple as a client-side operator stepping away from his/her desk in mid-transaction for a while. In the meantime, database resources remain locked and could potentially cause significant performance degradation and loss of service for other clients. It is therefore important that an application have a variety of tools available to cause a transaction to timeout and be aborted. NonStop SOAP 4 allows several configuration options to set timeout values at the operation level from the client or on the server using the `services.xml` file, or globally at a system level within TMF:

- `TimeOut`:

  The `TimeOut` attribute can be set to a timeout value by the client in the transaction header block of the SOAP request header element, which is used to begin a transaction.

- `TMFTimeout`:

  The `TMFTimeout` attribute can be specified at the operation level within the `services.xml` file. For more information on how to configure the `TMFTimeout` attribute, see "Configuring Transaction Timeout" (page 247).

- `TMFAutoAbort`:

  `TMFAutoAbort` is a system-defined timeout that specifies the time duration, after which all TMF transactions are aborted automatically by the TMF subsystem. The system default value is 2 hours. This timeout value can be changed by the system administrator using the TMFCOM utility.

  For information on customizing the `TMFAutoAbort` timeout value using the TMFCOM utility, see the *TMF Reference Manual*.

After comparing the values of `TimeOut` and `TMFTimeout`, if specified, the SOAP server takes the minimum value and uses it in the call to TMF that begins the transaction. If neither is specified, the value of `TMFAutoAbort` will determine transaction timeout.

NOTE: It is not possible to specify a timeout greater than the system-wide `TMFAutoAbort` value, which will supersede any attempt to set a higher value.

Table 19 summarizes the timeout value that will be derived if any combination of attributes is defined by the SOAP client, SOAP server, and system-wide.

Table 19 Transaction Timeout

Timeout specified in the `TimeOut` attribute	**Timeout specified in `TMFTimeout` attribute**	**`TMF AutoAbort` Timeout**	**Effective timeout**
No	No	Yes	`TMFAutoAbort` timeout.
No	Yes	Yes	Minimum of the `TMFAutoAbort` timeout and the value specified in the `TMFTimeout` attribute.
Yes	No	Yes	Minimum of the `TMFAutoAbort` timeout and the value specified in the `TimeOut` attribute.
Yes	Yes	Yes	Minimum of the `TMFAutoAbort` timeout, the `TimeOut` attribute and the `TMFTimeout` attribute.

Configuring Transaction Timeout

You can configure the `TMFTimeout` attribute at the operation level in the `services.xml` configuration file to control transaction timeout. An individual SOAP request can override this value using the `TimeOut` attribute in a Transaction header block but it can only set it to a lower value; any attempt to set a higher value for the transaction timeout will be ignored.

Similarly, the `TMFTimeout` attribute cannot be used to set a higher transaction timeout than configured for TMF with the AutoAbort timeout because TMF will ignore any value submitted to it that exceeds AutoAbort.

To configure transaction timeout in NonStop SOAP 4, complete the following steps:

1. Open the `services.xml` configuration file located in *`<NonStop SOAP 4 Deployment Directory>`*`/services/`*`<service_name>`*.

   where,

   `<service_name>` is the name of the service for which you want to configure the transaction timeout period.

2. Set the value of the `TMFTimeout` attribute in seconds.

   ```
   <operation>
     .
     .
   TMFTimeout=<transaction time out in seconds>
     .
     .
   </operation>
   ```

   where,

   `TMFTimeout` is the timeout value specified in seconds.

3. Save and close the `services.xml` configuration file.

NOTE:

- When developing a service, you can set the `TMFTimeout` attribute as an operation level attribute in the SDL file.
- When deploying a service using the `SoapAdminCL` tool, the `services.xml` configuration file will have the `TMFTimeout` attribute set to a value specified in the SDL file.

  For more information about setting the `TMFTimeout`attribute in the SDL file, see "NonStop SOAP 4 Service Description Language" (page 153).

Configuring Transactions to Abort on Fault

When SOAP client-managed transactions are used, by default, the SOAP server suspends the TMF transaction when it receives a fault indication back from a service. The SOAP server then sends a message containing the error back to the client. The client decides what action to take next; that is, NonStop SOAP 4 does not abort the transaction without instructions from the client. (The only exception is that a transaction can be aborted by TMF if it exceeds its timeout limit. In this case, NonStop SOAP 4 will receive an error back from TMF whenever it attempts its next call on behalf of this transaction, and will return the TMF error to the client in a SOAP fault message.)

This default action can be overridden through the use of the `AbortTransactionOnFault` attribute, which can be configured at the operation level in the `services.xml` configuration file.

The `AbortTransactionOnFault` attribute can be set to one of the following values:

- `yes`

  NonStop SOAP will abort a client-managed transaction if the service returns a fault or an error occurs in NonStop SOAP 4 during request processing.

- `no`

  Nonstop SOAP 4 server will not abort a client-managed transaction if the service returns a fault or an error occurs in NonStop SOAP 4 during request processing. NonStop SOAP 4 suspends the transaction and returns the error to the client, which then needs to decide the error action to take. The response message sent by the SOAP server will include a transaction header block with `TransactionID` set to the TMF transaction ID of the suspended transaction. This allows the client to send an abort request to NonStop SOAP 4, if that is the appropriate action for the application.

NOTE: For SOAP server-managed transactions (that is, the `TMFTransactionSupport` attribute is set to `Required` and there is no transaction header block in the SOAP request), the value of the `AbortTransactionOnFault` attribute, if set to `no`, is ignored and processing will proceed as if set to the value of `yes`. That is, the transaction will be aborted if the service returns a fault or an error occurs in NonStop SOAP 4 during request processing.

To configure NonStop SOAP 4 to abort a client-initiated transaction in case of an error or fault, complete the following steps:

1. Open the `services.xml` configuration file located in *<NonStop SOAP 4 Deployment Directory>*`/services/`*<service_name>*.

   where,

   <service_name> is the name of the service for which you want to enable or disable the `AbortTransactionOnFault` attribute.

2. Set the value of the `AbortTransactionOnFault` attribute to either `yes` or `no`.

```
<operation>
  .
  .
AbortTransactionOnFault="[yes | no]"
```

```
    .
    .
</operation>
```

3. Save and close the `services.xml` configuration file.

NOTE:

- When developing a service, the `AbortTransactionOnFault` attribute can be set as an operation-level attribute in the SDL file.
- When deploying a service using the `SoapAdminCL` tool, the `services.xml` configuration file will have the `AbortTransactionOnFault` attribute set to a value specified in the SDL file.

  For more information about setting the `TMFTimeout`attribute in the SDL file, see "NonStop SOAP 4 Service Description Language" (page 153).

Session Management and Transactions

Previous versions of NonStop SOAP (SOAP 3 and earlier), defined the concept of a session as a logical grouping of a series of requests and were required for transaction control; a transaction could occur only within a session. However, with the release of NonStop SOAP 4, the logical grouping of requests is now defined as a transaction, and sessions are no longer required to perform a multi-step transaction.

NOTE: If you are new to NonStop SOAP, you need not read further in this chapter because the remaining sections describe features that are simply retained to aid in compatibility and migration from earlier releases.

NonStop SOAP 4 defines the concept of a session header block that can be contained within a SOAP request, and may include the use of the attributes listed in Table 20.

Table 20 Session Header Attributes

Attribute	Values	Description
`SessionCommand`	`Begin, End`	Used to begin or end a session.
`SessionID`	Set by SOAP server	Contains a unique value set by the SOAP server in a response message to a begin session request to act as a 'handle' for the session in subsequent client requests.
`BeginNewTransaction`	`yes`	Starts a new transaction within an active session.
`CurrentTransactionCommand`	`End, Abort`	Commits or aborts a transaction.

The session header block can be defined in the WSDL file for a service by setting the service-level attribute `GenerateSessionHeader` to `yes` in the SDL file for the service.

NOTE: For any given service, the WSDL file generated by the `SoapAdminCL` tool will not define both session header blocks and transaction header blocks. The inclusion of the session and transaction header blocks in the WSDL file is controlled by the `GenerateSessionHeader` and `GenerateTransactionHeader` attributes of the `Service` element in the SDL file. Only one of these attributes may be set to `yes` for a given service. An error is issued by `SoapAdminCL` if both the attributes are set to `yes` for a particular service.

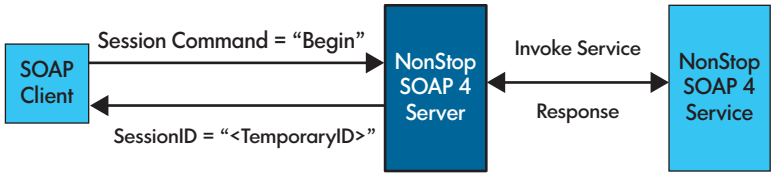
The following sections describe how each of these session attributes may be used to start and end sessions, and to begin and end transactions within these sessions.

Begin a New Session

To start a new session, the client must set the value of the `SessionCommand` attribute (in the session header block of the NonStop SOAP 4 request header element) to `Begin`.

Figure 18 shows a schematic flow for beginning a new session.

Figure 18 Beginning a New Session



When starting a new session, NonStop SOAP 4 performs the following activities:

1. The SOAP server generates a temporary SessionID.
2. If the body of the SOAP message requests a target service, NonStop SOAP 4 invokes the operation.
3. When the invoked service responds back to NonStop SOAP 4, or if there is no service request in the SOAP message body, the SOAP server constructs a session header block containing the `SessionID` attribute set to the temporary value created earlier.
4. The response is sent back to the client.

Example 25 shows a request/response pair of messages to begin a new session.

Example 25 Begin a New Session

```
Client Request
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:urn="urn:compaq_nsk_oss_SoapHeader">
<soapenv:Header>
  <urn:Session SessionCommand="Begin"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
 .
</soapenv:Body>
</soapenv:Envelope>


NonStop SOAP 4 Response

<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:urn="urn:compaq_nsk_oss_SoapHeader">
<soapenv:Header>
 <urn:Session SessionID="Tempmj12rty67"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
</soapenv:Body>
</soapenv:Envelope>
```

NOTE: The `SessionID` value created in this step is called *temporary* because it has no practical use. The `SessionID` assumes the value once a transaction has started, which can be requested in the same SOAP message as that used to start the session (and is recommended for efficiency).

Begin Transaction within a Session

To start a new transaction under an active session, the client must set the value of the `SessionID` attribute (in the session header block of the NonStop SOAP 4 request header element) to the session identifier. The client must also set the value of the `BeginNewTransaction` attribute in the session header block of the SOAP request header element to `yes`.

To start a new TMF transaction within a session, NonStop SOAP 4 performs the following activities:

1. The SOAP server begins a TMF transaction in response to a client request.
2. If the body of the request invokes a target service, the SOAP server invokes the operation specified in the body under the transaction that has started.
3. The NonStop SOAP 4 server suspends the TMF transaction and returns the response to the client if the service does not return a fault or if the `AbortTransactionOnFault` attribute is set to `no`.
4. The NonStop SOAP 4 server sets the TMF transaction ID in the `SessionID` attribute of the session header block.
5. If the service returns a fault and the `AbortTransactionOnFault` attribute is set to `yes`, the SOAP 4 server aborts the TMF transaction and returns a fault response to the client. The SOAP server generates a temporary session identifier to set in the `SessionID` attribute (there is no valid TMF Transaction ID to use), and returns it in the session header block in the response message.

Example 26 shows shows a request/response pair of messages to begin a new transaction within a session.

Example 26 Begin a New Transaction within a Session

Client Request

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_SoapHeader">
<soapenv:Header>
   <urn: Session SessionID="Tempmj12rty67" BeginNewTransaction="yes"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
</soapenv:Body>
</soapenv:Envelope>
```

NonStop SOAP 4 Response

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_SoapHeader">
<soapenv:Header>
   <urn: Session SessionID="234567"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
</soapenv:Body>
</soapenv:Envelope>
```

where,

`Tempmj12rty67`
: is the temporary session identifier provided by NonStop SOAP 4 in its response to starting the session.

`234567`
: is the TMF transaction ID.

For efficiency, it is recommended that you combine the two requests described into a single request as shown in Example 27.

Example 27 Begin a New Session and Transaction Combined

```
Client Request
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_SoapHeader">
<soapenv:Header>
   <urn:Session SessionCommand="Begin" BeginNewTransaction="yes"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
</soapenv:Body>
</soapenv:Envelope>

NonStop SOAP 4 Response
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_SoapHeader">
<soapenv:Header>
   <urn: Session SessionID="234567"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
</soapenv:Body>
</soapenv:Envelope>
```

NOTE: The value returned in the `SessionID` attribute (234567) is the TMF transaction ID.

Continue a Transaction within a Session

To continue an active transaction under a session, the client must set the value of the `SessionID` attribute (in the session header block of the NonStop SOAP 4 request header element) to the `SessionID` value supplied in the previous response from NonStop SOAP 4 for this session.

On receiving a SOAP request to continue an active transaction under a session:

1. The NonStop SOAP 4 server resumes the TMF transaction specified in the `SessionID` attribute of the transaction header block in the SOAP request header element.
2. The NonStop SOAP 4 server performs the operation specified in the body under the resumed transaction.
3. If the service does not return a fault or the `AbortTransactionOnFault` attribute is set to `no`, the NonStop SOAP 4 server suspends the TMF transaction and returns the TMF transaction ID in the `SessionID` attribute of the session header block in the SOAP response header element.
4. If the service returns a fault and the `AbortTransactionOnFault` attribute is set to `yes`, the SOAP 4 server will abort the TMF transaction and return a fault response to the client. Because the previous value for `SessionID` is no longer valid (the transaction ID it represents has been committed), the SOAP server generates a temporary session identifier to set in the `SessionID` attribute, and returns it in the session header block in the response message.

Example 28 shows a request/response pair of messages to continue a transaction within a session.

Example 28 Continue a Transaction

Client Request

```
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:urn="urn:compaq_nsk_oss_SoapHeader">
<soapenv:Header>
 <urn:Session SessionID="234567"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
 .
</soapenv:Body>
</soapenv:Envelope>
```

NonStop SOAP 4 Response

```
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:urn="urn:compaq_nsk_oss_SoapHeader">
<soapenv:Header>
 <urn: Session SessionID="234567"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
 .
</soapenv:Body>
</soapenv:Envelope>
```

Commit a Transaction within a Session

To commit an active transaction, the client must set the value of the `CurrentTransactionCommand` attribute to `Commit` in the session header block and also provide the appropriate `SessionID` – that is, the `SessionID` value supplied in the previous response from NonStop SOAP 4 for this session.

On receiving a SOAP request to commit an active TMF transaction:

1. The SOAP server resumes the TMF transaction specified in the `SessionID` attribute of the session header block in the SOAP request header element.
2. The NonStop SOAP 4 server invokes the operation (if specified in the body of the request) under the resumed TMF transaction.
3. If the service does not return a fault, the SOAP server will commit the transaction.
4. If the service returns a fault and the `AbortTransactionOnFault` attribute is set to `yes`, the SOAP 4 server will abort the TMF transaction and return a fault response to the client. Because the previous value for `SessionID` is no longer valid (the transaction ID it represents has been committed), the SOAP server generates a temporary session identifier to set in the `SessionID` attribute, and returns it in the session header block in the response message
5. If the invoked service returns a fault and the `AbortTransactionOnFault` attribute is set to `no`, the SOAP server suspends the TMF transaction and returns the TMF transaction ID in the `SessionID` attribute in a session header block along with a SOAP fault.

Example 29 shows a request/response pair of messages to commit a transaction within a session.

Example 29 Commit a Transaction within a Session

Client Request

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_SoapHeader">
<soapenv:Header>
 <urn:Session SessionID="234567"
 CurrentTransactionCommand="Commit"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
 .
</soapenv:Body>
</soapenv:Envelope>
```

NonStop SOAP 4 Response

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_SoapHeader">
<soapenv:Header>
   <urn:Session SessionID="Tempmj12rty67"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
 .
</soapenv:Body>
</soapenv:Envelope>
```

where,

`234567`
is the TMF transaction identifier of the transaction to be committed.

NOTE: The SOAP request to commit a transaction can also be combined with a request to end the session.

Abort a Transaction within a Session

To abort an active transaction, the client must set the value of the `CurrentTransactionCommand` attribute to Abort in the session header block and provide the appropriate `SessionID` – that is, the `SessionID` value supplied in the previous response from NonStop SOAP 4 for this session.

On receiving a SOAP request to abort an active transaction under a session:

1. The SOAP server resumes the TMF transaction specified in the `SessionID` attribute of the session header block of the SOAP request header element.
2. The SOAP server aborts the TMF transaction.
3. If the SOAP message body requests a target service, it will be invoked but there will not be any active TMF transaction because it has been aborted in step 2.
4. Because the previous value for `SessionID` is no longer valid (the transaction ID it represents has been aborted), the SOAP server generates a temporary session identifier to set in the `SessionID` attribute, and returns it in the session header block in the response message.

Example 30 shows a request/response pair of messages to abort a transaction within a session.

Example 30 Abort a Transaction within a Session

Client Request

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_SoapHeader">
<soapenv:Header>
   <urn:Session SessionID="234567" Command="Abort"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
 .
</soapenv:Body>
</soapenv:Envelope>
```

NonStop SOAP 4 Response

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_SessionHeader">
<soapenv:Header>
   <urn:Session SessionID="Tempmj12rty67"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
 .
</soapenv:Body>
</soapenv:Envelope>
```

where,

`234567`
: is the TMF transaction identifier of the transaction to be aborted.

NOTE: HP recommends that the body of a request that aborts a transaction be empty. The transaction is aborted before any service invocation. This means that there will not be an active transaction when the service is invoked.

End the Session

To end an active session, the client must set the value of the `SessionCommand` attribute in the session header block of the SOAP request header element to `End` and also provide the appropriate `SessionID` – that is, the `SessionID` value supplied in the previous response from NonStop SOAP 4 for this session.

On receiving a SOAP request to end a session:

1. The NonStop SOAP 4 server ends the temporary session in response to a client request.
2. If the body of the request invokes a target service, the NonStop SOAP 4 server performs the operation specified in the body without a session.
3. The NonStop SOAP 4 server returns the response to the client.
4. The NonStop SOAP 4 server does not include a session header block in the response message.

Example 31 shows a request/response pair of messages to end a session.

Example 31 End a Session

Client Request

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_SoapHeader">
<soapenv:Header>
   <urn:Session SessionCommand="End" SessionID="Tempmj12rty67"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
 .
</soapenv:Body>
</soapenv:Envelope>
```

NonStop SOAP 4 Response

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_SoapHeader">
<soapenv:Header>
</soapenv:Header>
<soapenv:Body>
 .
 .
 .
</soapenv:Body>
</soapenv:Envelope>
```

In this example, the response does not contain a session header block because there is no relevant session information to be communicated back to the client.

As noted earlier, a client can also choose to commit a transaction and end a session in the same request as shown in "End the Session" (page ?).

Example 32 Commit a Transaction and End a Session

Client Request

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_SoapHeader">
<soapenv:Header>
   <urn:Session SessionCommand="End" SessionID="234567"
 CurrentTransactionCommand="Commit"/>
</soapenv:Header>
<soapenv:Body>
 .
 .
 .
</soapenv:Body>
</soapenv:Envelope>
```

NonStop SOAP 4 Response

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:urn="urn:compaq_nsk_oss_SoapHeader">
<soapenv:Header>
</soapenv:Header>
<soapenv:Body>
 .
 .
 .
</soapenv:Body>
</soapenv:Envelope>
```

Again, the response does not contain a session header block because there is no relevant session information to be communicated back to the client now that the session closed.

The Cookie File

The implementation of sessions in previous releases (NonStop SOAP 3 and earlier) used an Enscribe key-sequenced file (the Cookie file) to maintain a relationship between a unique `SessionID` generated by NonStop SOAP and the TMF transaction ID. This was required because the NonStop SOAP server runs as a TS/MP server class under iTP WebServer and each process instance is context-free.

Transaction management within NonStop SOAP 4 no longer requires the presence of a session, and therefore, you do not have to maintain the relationship between session and transaction. While the session concept has been retained for compatibility with earlier releases, the examples in this chapter show that the value placed in the `SessionID` attribute is the actual TMF transaction ID. New users of NonStop SOAP 4 do not have to use the session construct.

As a result of this new implementation, the `Cookie` file is no longer required and will not be seen in this and future NonStop SOAP releases. This improves performance and simplifies management of the product.

Session Timeout

In NonStop SOAP 3 and earlier releases, the `SOAP_SESSION_TIMEOUT` attribute was used to terminate a session. This was an ineffective mechanism, because it did not result in the abort of any active transaction contained within that session. As a result, SOAP 3 was updated to improve the meaning of this attribute and use it as the timeout parameter to TMF when a transaction was started in addition to its previous role of triggering the end of a session.

In NonStop SOAP 4, the `SOAP_SESSION_TIMEOUT` attribute is no longer used. Instead, the `TMFTimeout` attribute is used. The `SOAP_SESSION_TIMEOUT` attribute is no longer required

because once a transaction is committed or aborted, a session does not have any meaning, and does not hold any resources in NonStop SOAP, whereas in previous releases, its context was still maintained in the Cookie file and needed to be deleted once a session was ended.

Subsessions

Previous releases of NonStop SOAP (NonStop SOAP 3 and earlier) supported a feature called Subsessions, which was designed to allow the SOAP server to maintain a dialog with a context-sensitive server class. This feature is no longer supported with NonStop SOAP 4.

The `SOAP_COOKIE_DELETION_INTERVAL` Parameter

In the previous releases, NonStop SOAP supported the configuration parameter `SOAP_COOKIE_DELETION_INTERVAL`, which denotes the minimum time interval between two automated cookie file cleanup operations. NonStop SOAP performs the automated cookie file cleanup activity when serving the next SOAP request after the cookie file cleanup interval.

This parameter has been removed in NonStop SOAP 4 because the cookie file is no longer present.

13 Using the Contract-First Approach in NonStop SOAP 4

You can use the following approaches to develop client applications and services in NonStop SOAP 4:

- **Contract-first Approach** - In this approach, the contract (WSDL file) is defined first, which states the type of service interface description that the service expects. The service code is generated based on this information.
- **Service-first Approach** -In this approach, the service code is written first and the contract (WSDL file) is generated from the written code.

This chapter describes the procedure to develop a Web service using the contract-first development approach and includes the following topics:

- "NonStop SOAP 4 Tools for Developing Web Services Using the Contract-First Approach" (page 260)
- "WSDL Considerations" (page 260)
- "Developing a NonStop SOAP 4 Pathway Web Service Using the WSDL2PWY Tool" (page 262)
- "Developing a NonStop SOAP 4 Non-Pathway Web Service Using the WSDL2C Tool" (page 270)

NonStop SOAP 4 Tools for Developing Web Services Using the Contract-First Approach

NonStop SOAP 4 provides the following tools to develop Web services using the contract-first approach:

- WSDL2PWY – The WSDL2PWY tool uses a WSDL file as input to generate client stubs and Pathway service skeleton files. The Pathway service skeleton files are used to generate a Pathway application that can be deployed as a Web service in NonStop SOAP 4.

  The WSDL2PWY tool must be used when developing services implemented as Pathway applications.

- WSDL2C – The WSDL2C tool uses a WSDL file as input to generate the client stubs and service skeleton files. The generated service skeleton files implement the basic interface code required while developing a service using NonStop SOAP 4 service APIs.

  The WSDL2C tool must be used when developing services implemented as DLLs using NonStop SOAP 4 service APIs.

  For more information about NonStop SOAP 4 tools, see Chapter 10: "NonStop SOAP Tools" (page 194).

WSDL Considerations

To develop client applications and services using the contract-first approach, you can use a pre-defined contract (WSDL file).

Using a Pre-defined WSDL File

To generate the service skeleton files, you can provide a pre-defined WSDL file as an input to WSDL2PWY tool. The WSDL2PWY tool accepts any WSDL file that conforms to the World Wide Web Consortium (W3C) standards. WSDL versions 1.1 and 2.0 are supported.

NOTE: The NonStop SOAP 4 server does not support or provides limited support for some WSDL file elements for the services deployed in Pathway applications.

Table 21 lists the WSDL file elements that are not supported by the NonStop SOAP 4 server.

NOTE: Ensure that the elements listed in Table 21 are not present in the WSDL file that is provided as an input to the `WSDL2PWY` tool. If present, your service may not function as expected.

Table 21 Unsupported WSDL File Elements

WSDL Components	Elements Not Supported	Description
Element	`appInfo`	This element is application-specific and not supported on Pathway services.
	`Notation`	This element is not supported because Pathway can work only with a fixed buffer size whereas this element can have variable buffer sizes.
	`Selector`	This element is not supported because it selects the elements based on constraints, such as `Key` or `Keyref` elements that are also not supported.
XSD Restrictions	`Enumeration`	This element is not supported because the Pathway service requires a fixed buffer size whereas this element can have variable buffer sizes.
	`Whitespace`	This option is not supported in the DDL.
Datatypes	`hexBinary`	This element is not supported, because the Pathway service requires a fixed buffer size whereas this element can have variable buffer sizes.

Table 22 lists the WSDL file elements that receive limited support from the NonStop SOAP 4 server.

Table 22 WSDL File Elements with Limited Support

WSDL Components	Elements with Limited Support	Description
Element	`Key`	NonStop SOAP 4 does not validate or verify the data sent in the `Key` element of the SOAP message.
	`Keyref`	NonStop SOAP 4 does not validate or verify the data sent in the `Keyref` element of the SOAP message.
	`Unique`	NonStop SOAP 4 does not validate the uniqueness of the data sent in the `Unique` element of the SOAP message.
XSD Restrictions	`Pattern`	NonStop SOAP 4 does not validate the data against the specified regular expression pattern.
	`MaxExclusive`	NonStop SOAP 4 does not verify the range of data sent in a NonStop message.
	`MinExclusive`	NonStop SOAP 4 does not verify the range of data sent in a NonStop message.
Datatypes	`anyURI`	NonStop SOAP 4 does not validate if the Web address sent is valid.
	`Qname`	NonStop SOAP 4 does not validate if the qualified name sent is valid.

Table 22 WSDL File Elements with Limited Support *(continued)*

WSDL Components	Elements with Limited Support	Description
	normalizedString	NonStop SOAP 4 does not normalize the string, sent in the Datatype for white spaces.
	Duration	NonStop SOAP 4 handles Duration as xsd:string.

NOTE: If you are not using a pre-defined WSDL file, you can create a new WSDL file from the DDL and SDL files by using the SoapAdminCL tool. For more information on generating a WSDL file using the SoapAdminCL tool, see "Deploying a Pathway Application and a NonStop Process as a Web Service " (page 77).

Developing a NonStop SOAP 4 Pathway Web Service Using the WSDL2PWY Tool

With the WSDL file for the service, you can create the client stubs and the Pathway service skeleton files using the WSDL2PWY tool.

To generate the stubs, complete the following steps:

1. Set the OSS environment variable NSSOAP_HOME to the OSS location where the NonStop SOAP 4 installation directory is located:

   ```
   OSS> export NSSOAP_HOME=<NonStop SOAP 4 Installation Directory>
   ```

   For example:

   ```
   OSS> export NSSOAP_HOME=/usr/tandem/nssoap/t0865h01
   ```

   where,

   /usr/tandem/nssoap/t0865h01 is the NonStop SOAP 4 installation directory location.

2. Add the directory containing the SoapAdminCL executable image to the OSS PATH variable.

   ```
   OSS> export PATH=<NonStop SOAP 4 Installation Directory>/tools:$PATH
   ```

   For example:

   ```
   OSS> export PATH=/usr/tandem/nssoap/t0865h01/tools:$PATH
   ```

3. Add the <Java Installation Directory>/bin directory to the PATH environment variable, using the command:

   ```
   OSS> export PATH=<Java Installation Directory>/bin:$PATH
   ```

   For example:

   ```
   OSS> export PATH=/usr/tandem/java/bin:$PATH
   ```

   where,

   /usr/tandem/java/
   is the Java installation directory.

4. Generate the service skeleton files using the WSDL2PWY tool, using the following command:

   ```
   OSS> WSDL2PWY -o <output directory> -ss -uri <wsdl path>
   ```

   where,

   -o
   specifies the directory path for the generated files.

   -ss
   generates the service skeleton files. If this option is not specified, the service skeleton files are not generated by the WSDL2PWY tool.

`-uri`

specifies the location where the WSDL file is available.

For example:

```
OSS> WSDL2PWY -o "/home/usr/my_nssoap/services/reflector" -ss
              -uri "/home/usr/my_nssoap/services/reflector/SoapPW_reflector.wsdl"
```

where,

`/home/usr/my_nssoap/services/reflector` is the location where the service skeleton files will be generated.

NOTE: Verify that the location of the Java executable object is included in the `PATH` environment variable. If not included, the `WSDL2PWY` tool will not be able to detect the Java object during runtime and will return an error.

On successful execution, the following files are generated:

- Service skeleton files

  The following service skeleton files are generated under the `src` directory in the location specified in the -o option of the `WSDL2PWY` command:

  - Service skeleton source file to implement your application business logic.
  - Header file that holds the declarations for the functions generated in the service skeleton source file.

  NOTE: If the `-o` option is not specified, the files are generated in the `src` directory under the current directory.

  For example:

  In case of the `reflector` service, the following files are generated in the *<NonStop SOAP 4 Deployment Directory>*`/services/reflector/src` directory:

  - `pwayreflectorService.c`
  - `pwayreflectorService.h`

- Makefile

  The `Makefile_service` Makefile is generated in the location specified in the `–o` option of the `WSDL2PWY` command. If the `-o` option is not specified, the file is generated in the current directory.

5. Update the service skeleton files with the service business logic.

   The `WSDL2PWY` tool generates a source code file and a header file for each service specified in the WSDL file. The source code file for the service contains the following functions:

   a. `main()`

   The `main` function expects two arguments, a character pointer array (`char *argv[]`) and an integer value (`int argc`). `argv` includes arguments passed to the function while starting the server executable, whereas `argc` includes a value equivalent to the number of arguments passed to the function.

   i. The `main()` function opens the `$RECEIVE` file and waits till a request is received by the service using the `READUPDATEX` procedure call.

   ii. After a request is received, the `main()` function checks if any error occurred while reading the request.

   iii. If no error is found, the request buffer is passed to the `process_application_message()` function.

iv. If an error occurs, the request is passed to the `process_system_message()` or `file_error()` function based on the error value.

v. After a response is returned to the calling NonStop SOAP 4 server, the `$RECEIVE` file is closed.

NOTE: The response buffer size is 32k. Use the `-lps` option while generating the skeleton files to pass large data. The data size limit is 2M.

b. `process_application_message()`

The `process_application_message()` function expects a character buffer as an argument (`char *recv_buff`). The character buffer includes the request received by the service. You must update this function to include calls to any operation-specific functions based on your requirements.

The call to `REPLYX` is generated by the `WSDL2PWY` tool in the `process_application_message` function. This call returns the reply generated by the service to the calling NonStop SOAP 4 server.

c. `operation_pway_<operation_name>`

The `WSDL2PWY` tool generates a function skeleton file for each operation specified in the WSDL file. The `operation_pway_<operation_name>` function has the following information passed as arguments in it:

- A character buffer that includes the request received by the service
- A pointer to a character buffer including the response that the function will generate
- A pointer to a short datatype that contains the size of the response being returned

The `operation_pway_<operation name>` function contains the structure definitions of the request, response, and fault structures defined in the WSDL file. These structures are defined in the generated header file.

For the `reflector` service, the following functions defined by the `WSDL2PWY` tool must be customized as follows:

- `main`

  In the case of the `reflector` service, the `main` function appears as follows:

```
1  int main (int argc, char *argv[])
2  {
3          /**
4           * Variable Definitions
5           */
6
7          /**
8           * recv_num contains the file identifier for $RECEIVE
9           */
10         short recv_num = 0;
11
12         /**
13          * receiveinfo variable is defined to be passed as argument
14          * to FILE_GETRECEIVEINFO_
15          */
16         short receive_info[17];
17
18         /**
19          * recv_name_length contains the length of variable recv_name
20          */
21         short recv_name_length = 0;
22
23         /**
24          * count_read is passed as argument to READUPDATEX and will contain
25          * the number of bytes read from $RECEIVE
26          */
27         unsigned int count_read = 0;
28         short error = 0;
29
30         /**
31          * recv_buff is a character buffer to contain
32          * the client request buffer received
33          */
34         char recv_buff[ RECV_BUFFER_LENGTH ];
35         const char recv_name[] = "$RECEIVE";
```

```
 36
 37          recv_name_length = (short)strlen( recv_name );
 38
 39          /**
 40           * FILE_OPEN_ is called to obtain a unique file identifier (recv_num)
 41           * for $RECEIVE
 42           */
 43          error = FILE_OPEN_( recv_name,
 44                              recv_name_length,
 45                              &recv_num,
 46                              ,           /*  Access  */
 47                              ,           /*  Exclusion  */
 48                              ,           /*  Nowait depth  */
 49                              1  );       /*  Receive Depth */
 50
 51          if ( recv_num == -1 )
 52          {
 53              printf( " Error while Opening $RECEIVE \n " );
 54              printf( " Error is = %d  \n " , error );
 55              exit(1);
 56          }
 57
 58          while ( 1 )
 59          {
 60                  /**
 61                   * Initialising the request buffer
 62                   */
 63                  memset(recv_buff, '\0', RECV_BUFFER_LENGTH);
 64
 65                  /**
 66                   * Read $RECEIVE file and return the number of bytes read in
 67                   * variable count_read and the data in recv_buff
 68                   */
 69                  READUPDATEX(recv_num,
 70                              (char *)recv_buff,
 71                              RECV_BUFFER_LENGTH - 1,
 72                              &count_read);
 73
 74                  /**
 75                   * FILE_GETINFO_ will return the file system error resulting
 76                   * from the last operation on file id recv_num in variable error
 77                   */
 78                  FILE_GETINFO_(recv_num, &error);
 79
 80                  /**
 81                   * FILE_GETRECEIVEINFO_ will return information about the last
 82                   * message read on $RECEIVE file in variable receiveinfo
 83                   */
 84                  FILE_GETRECEIVEINFOL_((short *)&receive_info);
 85
 86                  switch (error) {
 87                          /**
 88                           * Process Application message.
 89                           */
 90                          case NO_ERROR:
 91                                  process_application_message(recv_buff, count_read);
 92                                  break;
 93
 94                          /**
 95                           * Process System message.
 96                           */
 97                          case SYSTEM_MSG:
 98                                  process_system_message((short *)receive_info );
 99                                  break;
100
101                          default:
102                                  file_errors(error);
103
104                  } /*  End of switch statement        */
105
106                  /**
107                   * Setting the value of count_read to 0 at the end of request processing
108                   */
109                  count_read = 0;
110
111                  /**
112                   * if no opens are pending, then break the while loop and exit
113                   */
114                  if (!totalOpenCnt)
115                          break;
116          } /* while end */
117
118         error = FILE_CLOSE_(recv_num);
119          if ( error != 0 )
120          {
121                  printf( " Error while Closing $RECEIVE \n " );
122                  printf( " Error number is = %d  \n " , error );
123          }
124          return 0;
```

```
125  }  /*  End of main()  */
126
```

Line 10	The `recv_num` variable contains the file identifier for the `$RECEIVE` file and must be initialized to 0.
Line 16	Defines the `receiveinfo` variable. The `receiveinfo` variable is passed as an argument to the `FILE_GETRECEIVEINFO_` call.
Line 21	The `recv_name_length` variable contains the length of the `recv_name` variable and must be initialized to 0.
Line 27	The `count_read` variable contains the number of bytes read from `$RECEIVE` and is passed as an argument to `READUPDATEX`. The `count_read` variable must be initialized to 0.
Line 28	Checks the returned error.
Line 34	`recv_buff` is a character buffer that contains the received client request. Set the length of `recv_buff` to the maximum size of the response buffer (32KB).
Line 35 - Line 37	Initializes a variable with the `$RECEIVE` file name and another variable with the length of the `$RECEIVE` file name.
Line 43 - Line 49	The `FILE_OPEN_` call is called to obtain a unique file identifier (`recv_num`) for `$RECEIVE`. `FILE_OPEN_` opens the `$RECEIVE` file.
Line 51 - Line 56	Error handling if the `FILE_OPEN` is not able to open the `$RECEIVE` file.
Line 58- Line 116	Breaks the continuous loop if the server stops because of some error.
Line 63	Initializes the request buffer with `NULL` values.
Line 69- Line 72	Reads the `$RECEIVE` file and returns the number of bytes read in `count_read` variable and the data in `recv_buff`. At this point, the server waits until it receives a request from the client.
Line 78	The `FILE_GETINFO_` call returns a file system error if an error occurred in the last operation on `recv_num` file id. The error is returned in the `error` variable.
Line 84	`FILE_GETRECEIVEINFO_` call returns information about the last message read in the `$RECEIVE` file.
Line 86 - Line 104	The switch statement handles the error returned by `FILE_GETINFO`.
Line 90 - Line 92	In case of no error (`NO_ERROR`), call the `process_application_message()` function.
Line 97 - Line 99	In case of system message (`SYSTEM_MSG`), call the `process_system message()` function.
Line 101 - Line 102	In case of other errors, call the `file_errors()` function.
Line 109	Sets the value of `count_read` to 0 at the end of request processing
Line 114- Line 115	If no files are open, break the while loop and exit.
Line 118	Close the `$RECEIVE` file if the while loop breaks.
Line 119 - Line 123	Error handling if an error occurs while closing the `$RECEIVE` file.
Line 124	Return statement

- `process_application_message`

  In the case of the `reflector` service, the `process_application_message()` function appears as:

```
1  void process_application_message(char * recv_buff, unsigned long count_read) {
2
3          short error = 0;
4
5           /**
```

```
 6           * count_transferred defined to be passed as argument
 7           * to operation_pway_<operation name> function.
 8           * This variable will contain size of the
 9           * response buffer.
10           */
11         unsigned long count_transferred = 0;
12
13          /**
14           * response_buffer defined to be passed as argument
15           * to operation_pway_<operation name> function.
16           * This variable will contain the response buffer
17           */
18         char *response_buffer;
19
20          /**
21           * TO DO :
22           * Write the logic to invoke the functions defined for each operation.
23           * Sample call:
24           * operation_pway_<operation name>(recv_buff,
25           *                                 &response_buffer,
26           *                                 &count_transferred,
27           *                                 count_read);
28           */
29
30         error = REPLYX(response_buffer, count_transferred) ;
31          if (error != 0) {
32                 printf(" Error while using REPLYXL to transfer application message \n ");
33                 printf(" Error number is = %d  \n ", error);
34                 file_errors(error);
35          }
36
37  } /* End of process_application_message()   */
38
```

Line 3	The `error` variable is used to check the returned error and is initialized to 0.
Line 11	The `count_transferred` variable is defined to be passed as an argument to the `operation_pway_<operation name>` function. The `count_transferred` variable contains the size of the response buffer.
Line 20- Line28	Function call for each operation.
Line 30	Write a response to the `$RECEIVE` file using the `REPLYX` call.
Line 31- Line 35	Error handling if an error occurs while writing the response to the `$RECEIVE` file.

- `operation_pway_REFLECTOR`

  The WSDL file for the `reflector` service defines a single operation named `REFLECTOR`. Therefore, the `WSDL2PWY` tool generates a function for the `REFLECTOR` operation named `operation_pway_REFLECTOR`. The code snippet for the `operation_pway_REFLECTOR()` function is:

```
 1  /**
 2   * This function contains the business logic for operation: REFLECTOR
 3   *
 4   * Arguments:
 5   * request_buffer  - This is a character pointer that contains the request
 6   *                   buffer received from NonStop SOAP 4 server.
 7   * response_buffer - This is an output variable and will contain the response
 8   *                   buffer
 9   * reply_count     - This is an output variable and will contain the size of
10   *                   the response buffer
11   * count_read      - This is a variable of unsigned long datatype which contains the
12   *                   size of the request buffer
13   * Return Type: void
14   */
15  void operation_pway_REFLECTOR(char *request_buffer, char **response_buffer, unsigned long
*reply_count, unsigned long count_read)
16  {
17
18         short fileError;    19
20      /**
21       * Request structure variable definition
22       */
23       pwayREFLECTOR_t* req_1;
24
25      /**
26       * Response structure variable definition
27       */
28       pwayREFLECTORResponse0_t* resp_1;
29
30
31      /**
32       *
```

```
33       * Write the code implement business logic. The response buffer
34       * needs to be written into variable response_buffer and response
35       * size needs to be written into variable reply_count.
36       * The memory needed for the response structure also needs to be
37       * allocated.
38       */
39  }
40
```

Following is a list of functions based on the code sample:

Line Number(s)	Function
Line 18	The `fileError` variable is used to check the returned error and must be initialized to 0.
Line 23	Request structure variable definition.
Line 28	Response structure variable definition.
Line 31- Line 38	Write the code to implement your business logic. The response buffer must be written in the `response_buffer` variable and the response size must be written in the `reply_count` variable.

After customizing the service skeleton files generated by the `WSDL2PWY` tool, build the service object by running the `Makefile_service` file generated by the `WSDL2PWY` tool. To run the `Makefile_service` file, enter the following command:

```
OSS> make -f Makefile_service
```

On successful execution of the previous command, an object named *<service_name>*`.pway` is created in the same location as the Makefile (`Makefile_service`). For example, the `reflector` service will have an object name `reflectorService.pway`.

After generating the service object, you must configure it as a Pathway server class.

To configure the service object as a Pathway server class, complete the following steps:

1. Create the *<service_name>*`/src` directory in *<NonStop SOAP 4 Deployment Directory>*`/services/`*<service_name>*.
2. Copy the `conf` directory from `/usr/tandem/nssoap/t0865h01/sample_services/reflector/` to *<NonStop SOAP 4 Deployment Directory>*`/services/`*<service_name>*.

   For example, in case of the `reflector` service, copy the `conf` directory to the *<NonStop SOAP 4 Deployment Directory>*`/services/reflector` directory.
3. Copy the *<service_name>*`.pway` object from its existing location to *<NonStop SOAP 4 Deployment Directory>*`/services/`*<service_name>*`/src`.

   For example, in case of the `reflector` service, copy `reflectorService.pway` to the *<NonStop SOAP 4 Deployment Directory>*`/services/reflector/src` directory.
4. Rename the *<service_name>*`.pway` object to *service name*.

   For example, in case of the `reflector` service, rename `reflectorService.pway` to `reflector`.
5. Run the `pathConf` script:

   ```
   OSS> ./pathConf <Pathmon name>
   ```

   For example, to configure the `reflector` service under the PATHMON named `$ECHO`, use the following command:

   ```
   OSS> ./pathConf \$ECHO
   ```

   The service object is now configured as a Pathway server class.

Deploying a NonStop SOAP 4 Pathway Web Service

A NonStop SOAP 4 Pathway Web service can be deployed using an external WSDL file (a WSDL file not generated by the `SoapAdminCL` tool).

To deploy the Pathway Web service using an external WSDL file, complete the following steps:

1. Place the WSDL file in the *<NonStop SOAP 4 Deployment Directory>*`/services/`*<service_name>* `directory`.

   For example, in case of the `reflector` service, place the WSDL file for the `reflector` service in the *<NonStop SOAP 4 Deployment Directory>*`/services/reflector` directory.
2. Next, you must create a `services.xml` file for your service. You can start by copying an existing `services.xml` file to your service directory and then modify it to reflect information about your particular service. A sample `services.xml` file is found in the *<NonStop SOAP 4 Installation Directory>*`/sample_services/echo` directory.
3. Edit the `services.xml` file to reflect the configuration settings for your business environment. NonStop SOAP 4 reads the `services.xml` file at run-time to deploy the service based on the instructions set in the `services.xml` file.

   The `services.xml` file is used to set the following configuration parameters:

   1. Define service description parameters, such as name of the service, name of the service class, path of the service WSDL file, name of the PATHMON or process, and the server language.
   2. Define the response selection criteria in case the service is designed to support multiple valid responses.
   3. Configure transaction management specific parameters such as, granularity of the TMF transaction support and the operation-specific timeout.
   4. Specify references of any modules that need to be engaged at the service level.
   5. Specify the message exchange pattern and the message receiver that each operation needs to use.

   NOTE: For more information on `services.xml` file, see "NonStop SOAP 4 Configuration Files" (page 177).
4. Save and close the `services.xml` file.

Accessing a NonStop SOAP 4 Pathway Web Service

To access the deployed `reflector` service, complete the following steps:

1. Go to the *<iTP WebServer Deployment Directory>*`/conf` directory and run the `restart` script:

```
OSS> cd <iTP_WebServer_Deployment Directory>/conf
OSS> ./restart
```

2. Access the `reflector` service, using the following Web address pattern:

   `http://<ip address>:<port>/<url_pattern>/client`

   where,

   `<ip address>:<port>`
   is the IP address and port of iTP WebServer integrated with NonStop SOAP 4.

   `url_pattern`
   is the string entered in Step 6 in "Setting up the Deployment Environment" (page 38). The default value is `axis2c`.

   `client`
   is the name of the directory in *`<NonStop SOAP 4 Deployment Directory>`* where NonStop SOAP 4 HTML clients for the `reflector` service are located.

Developing a NonStop SOAP 4 Non-Pathway Web Service Using the `WSDL2C` Tool

The `WSDL2C` tool helps in generating a service that can be deployed as a non-Pathway service in NonStop SOAP 4. This tool is located in the *`<NonStop SOAP 4 Installation Directory>`*`/tools` directory. For more information on the `WSDL2C` tool, see Chapter 10: "NonStop SOAP Tools" (page 194).

The `WSDL2C` tool accepts any WSDL file that complies with the W3C standards. However, the WSDL file restrictions for the `WSDL2PWY` tool (see Table 21 (page 261)) are not applicable for the `WSDL2C` tool.

To create a service using the `WSDL2C` tool, complete the following steps:

1. Use an existing WSDL file or create a new WSDL file.
2. Develop a non-Pathway service using the `WSDL2C` tool.
3. Compile the service and deploy it in the NonStop SOAP 4 server.

For more information on creating services and clients using the `WSDL2C` tool, see "Generating the Service Skeleton Files" (page 100) and "Generating NonStop SOAP 4 Client Stubs using the `WSDL2C` tool" (page 119).

14 WS–Security in NonStop SOAP 4

WS-Security provides a platform to secure your services beyond transport level protocols, such as HTTPS. HTTPS performs a secure message transfer from one end point to another. However, in the real world, the message is transferred over multiple domains and you must preserve the identity, integrity, and security of the message across multiple trusted domains or points. WS-Security provides an end-to-end solution for Web service security.

WS-Security allows you to perform the following:

- Pass authentication tokens between services
- Encrypt messages or part of messages
- Sign messages
- Timestamp messages

NonStop SOAP 4 uses Axis2c Rampart module to implement WS-Security. WS-Security can be activated by using WS-SecurityPolicy 1.1. WS-SecurityPolicy 1.1 provides a set of standards for validating the security properties of a received message.

You can configure the client with the help of algorithms that are supported in WS-SecurityPolicy. You cannot secure the NonStop SOAP service with a non WS-SecurityPolicy approach.

The Axis2c Rampart module provides an implementation of the primary security standards for Web Services, such as the OASIS Web Services Security specification from the OASIS Web Services Security TC.

The Rampart module provides an implementation of the following WS-Security standards:

- SOAP Message Security V1.0 (http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf)
- Username Token Profile V1.0 (http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf)
- X.509 Certificate Token Profile V1.0 (http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf)

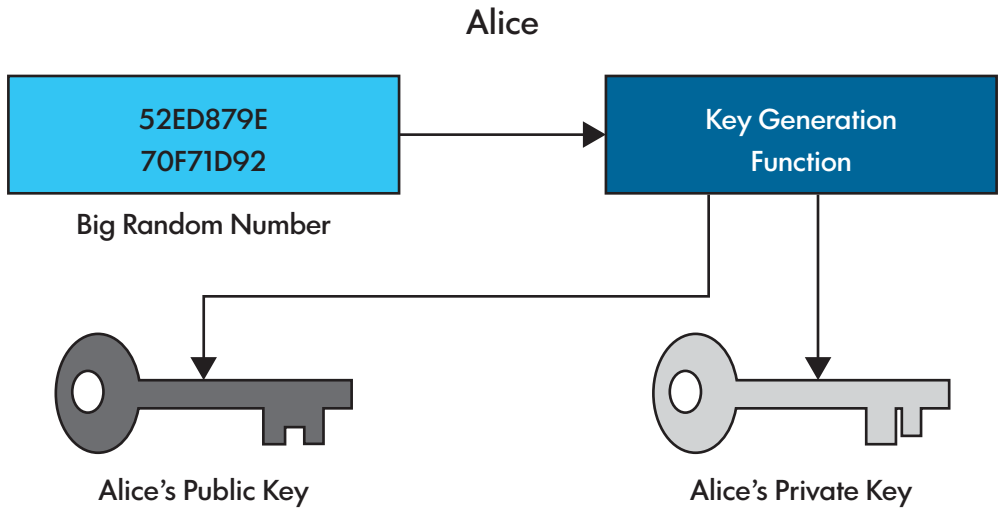
This chapter includes the following topics:

- "Overview of Encryption and Signing" (page 271)
- "Supported WS–Security Features " (page 273)
- "Securing a NonStop SOAP 4 Service" (page 274)
- "Rampart Specific Assertions " (page 275)
- "Publishing the Security Requirements" (page 277)
- "Configuring the Client to Invoke a Secured Web Service" (page 277)
- "Extensible Modules " (page 278)
- "Sample Programs" (page 279)
- "Recommendations" (page 284)

Overview of Encryption and Signing

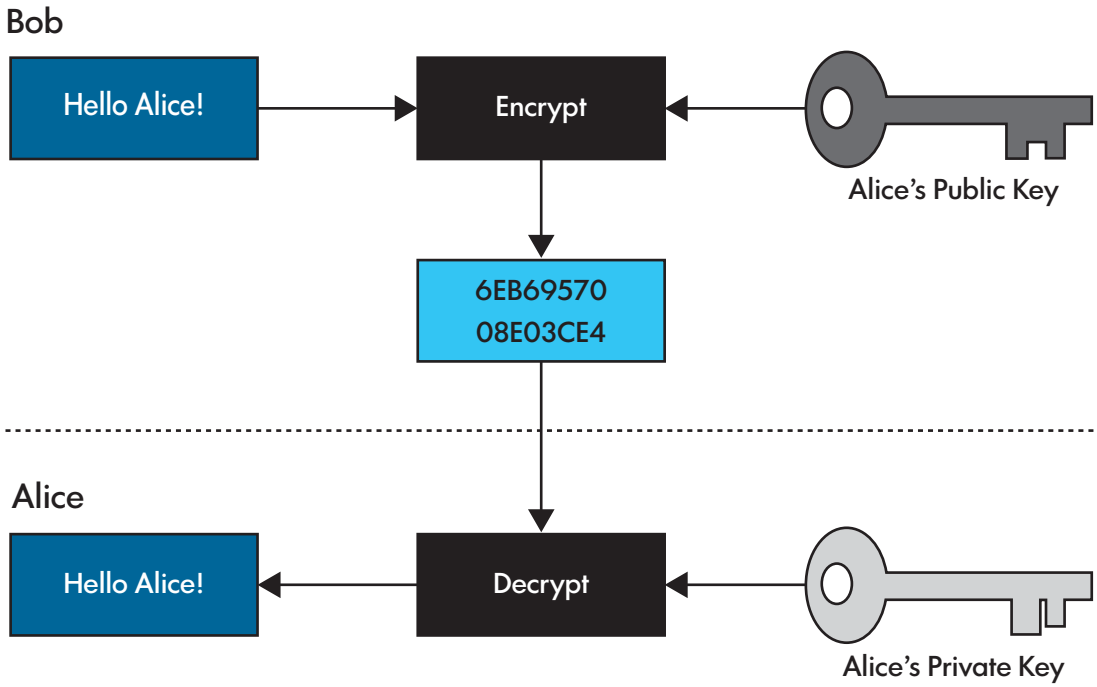
WS-Security uses public or private key cryptography. Public key cryptography contains a pair of public and private keys. Although different, the two parts of the key pair are mathematically linked. These are generated by using a large prime number and a key function.

Figure 19



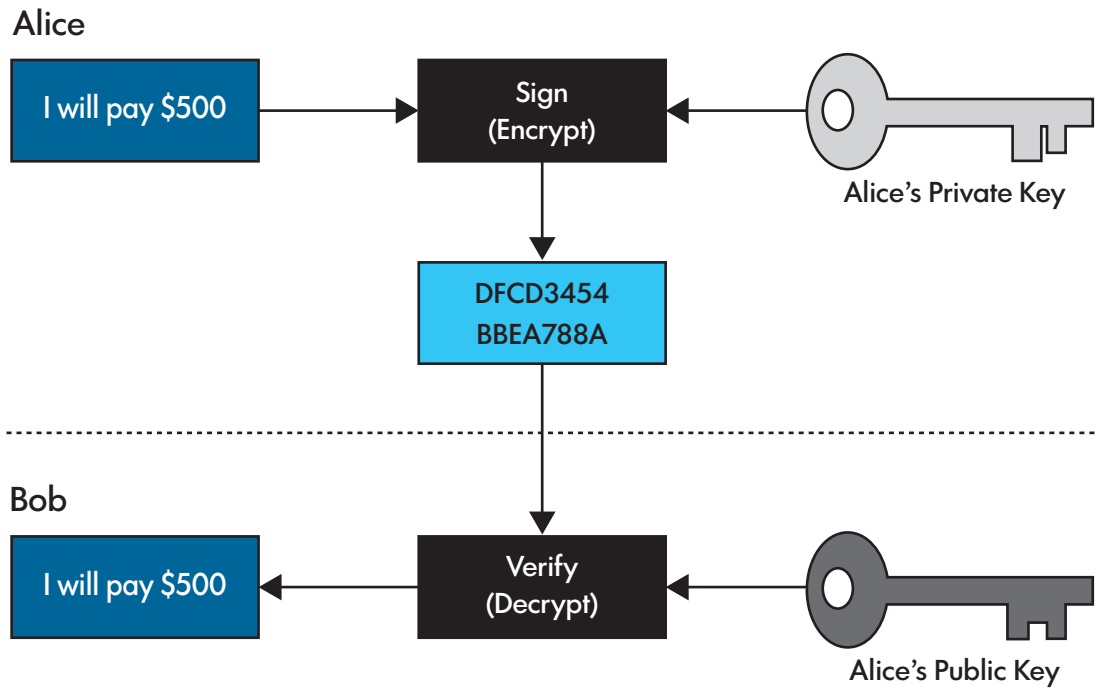
These keys are used to encrypt the messages. For example, if Bob wants to send a message to Alice, he can encrypt a message using her public key. Alice can then decrypt this message using her private key. Only Alice can decrypt this message as she is the only one with the private key.

Figure 20



Messages can also be signed. You can ensure the authenticity of the message. If Alice wants to send a message to Bob, and Bob wants to be sure that it is from Alice, then Alice can sign the message by using her private key. Bob can then verify that the message is from Alice by using her public key.

Figure 21



Supported WS–Security Features

The `Rampart` module, when engaged with NonStop SOAP 4, provides the following security features:

1. SOAP message encryption

   Any part of the SOAP message can be encrypted. NonStop SOAP 4 supports the following levels of message encryption:

   - Derived key support for additional security
   - Symmetric and Asymmetric modes of operation
   - Support for Advanced Encryption Standard (AES) and Triple Data Encryption Standard (DES)
   - Signature encryption
   - Keys encryption

2. SOAP message signature

   Any part of the SOAP message can be signed using a private key to maintain the integrity of the SOAP message.

   NonStop SOAP 4 supports the following levels of message signature:

   - XML signature with `RSA-SHA1` algorithm
   - Message authentication with `HMAC-SHA1` algorithm
   - Signature confirmation support
   - SOAP Header signing

3. Support for Key Store

   X.509 certificates and private keys are supported. The keys are stored in the Privacy Enhanced Mail (PEM) files.

4. Timestamps

   Allows timestamps to be added to a message to enable the server to verify the message validity in terms of each SOAP message.

5. Username Tokens

   NonStop SOAP 4 can send and verify username tokens with Username and plaintext password or Username and digested password.

6. Protection Orders

   NonStop SOAP 4 supports encrypt before signing and sign before encrypting.

7. Extensible Modules

   NonStop SOAP 4 supports password provider module and authentication module.

8. Keys Management

   NonStop SOAP 4 supports X.509 token profile and Key identifiers, Thumb prints, Issuer or Serial pairs, embedded, and direct references key management techniques.

NOTE: Fault messages cannot be secured.

Securing a NonStop SOAP 4 Service

You can use the following steps to secure a NonStop SOAP 4 service:

1. Setting up the key store

   To sign or encrypt messages back and forth, both the client and the service must posses public-private key pairs. If you are going to secure your service with XML signature or encryption techniques, you must have the X.509 certificates. For testing your service on a development environment, you can either create the X.509 certificate yourself by using the required tools, such as OpenSSL or you can use the certificates that are shipped with the sample programs. You must get the certificates from a Certification Authority when the services are secured on the production environment.

2. Writing the password provider

   The Rampart module uses a password provider library to authenticate the username tokens and to retrieve the private key to sign SOAP messages. Each private key has a password associated with it. To retrieve the private key, you must provide the password of the relevant key. The sample password provider included in the sample program reads password for the username/private key from a flat file. You can change the sample password provider to retrieve the passwords from a database, a LDAP server or any other storage by writing the relevant password retrieval logic.

3. Constructing the security policy

   In NonStop SOAP 4, a policy based configuration approach is followed to configure the security. You must construct a suitable security policy using WS-SecurityPolicy1.1 to define the security requirements of the Web service. WS-SecurityPolicy1.1 is built on top of WS-Policy framework and defines a set of policy assertions that can be used in defining individual security requirements or constraints. The individual policy assertions can be combined by using policy operators defined in the WS-Policy framework to create security policies that can be used to secure messages exchanged between a Web Service and a client. You can use the security policies defined in the sample `services.xml` or `policy.xml` as template to build your own policies. For more information on the sample programs, see "Sample Programs" (page 279). The complete specification of WS-SecurityPolicy can be found at http://specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf .

4. Providing Rampart specific configuration details

   Rampart module uses `RampartConfig` assertion to provide Rampart specific configuration details to Rampart Engine. You can use this configuration to specify the username, password type, path to the password provider library, and the path to certificates used for signing and encryption. Based on your security requirements, you can add the required `RampartConfig` assertions to the security policies. For more information about Rampart configuration assertions, see "Rampart Specific Assertions " (page 275).

5. Applying the security policy

   You can use the service descriptor `services.xml` file to engage the Rampart module and to apply the security policy to the Web service. The `services.xml` file is located at `<Nonstop SOAP 4 Deployment Directory>/services/<service_name>`.

   To engage the Rampart module, you must add the element *<module ref="rampart"/>* in `services.xml` file.

   To apply the security policy, add the policy that you have created for the service in the `services.xml` file.

   The following sample shows the rampart enabled and security policy enabled `services.xml` file:

```
<service name="sec_echo1">
<parameter name="ServiceClass" locked="xsd:false">sec_echoparameter name="ServiceClass"
locked="xsd:false">sec_echo</parameter>
<description>This is a testing service.description>This is a testing service.</description>
<module ref="rampart"/>
<operation name="echoString">
<parameter name="wsamapping">http://example.com/ws/2004/09/policy/Test/EchoRequest</parameter>
</operation>
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
<!--Your policies are here-->
</wsp:Policy>
</service>
```

   You can also engage the Rampart module for all the Web services by adding`<module ref="rampart"/>` in `axis2.xml`. The `axis2.xml` is located at *<NonStop SOAP 4 deployment directory>*.

   The following sample displays the `axis2.xml` file with rampart module enabled:

```
<axisconfig name="Axis2/C">
   .......
<!-- ============= -->
<!-- Global Modules  -->
<!-- ============= ->
.....
<module ref="rampart"/>
.....
</axisconfig>
```

   If you add *<module ref="rampart"/>* element in `axis2.xml`, then you need not add this element in `services.xml`.

6. Restart the NonStop SOAP 4 server.

```
OSS>./<iTP WebServer Deployment Directory>/conf/restart
```

NOTE: If the service is regenerated through SoapAdminCL, the security policies in the `services.xml` are removed.

Rampart Specific Assertions

Rampart specific policy assertions can be used along with WS-SecurityPolicy assertions while securing the services.

Table 23 displays the RampartConfig assertions:

Table 23 Rampart Specific Assertions

Parameter	Description	Example
User	This denotes the username that must be used in the UserNameToken.	`<rampc:User>Bob</rampc:User>`
EncryptionUser	This denotes the username that must be used to retrieve the password of the private key. You can retrieve the password by using a password callback provider.	`<rampc:EncryptionUser>b</rampc:EncryptionUser>`
PasswordType	This denotes the password type to be used in UserNameToken. The valid values are `plainText` or `Digest`.	`<rampc:PasswordType>Digest</rampc:PasswordType>`
PasswordCallbackClass	This denotes the path to the password provider library that provides the password required to create the UsernameToken or to sign the message.	`<rampc:PasswordCallbackClass>/usr/tandem/nssoap/t0865h01_AAL/sample_services/sec_echo/service/libpwcb.so</rampc:PasswordCallbackClass>`
AuthnModuleName	This denotes the path to the Authentication Module library. This can be used on the server side if you want to have your own authentication logic for validating the UserNameToken. The Rampart module invokes this library by passing the username and password.	`<rampc:AuthnModuleName>/usr/tandem/nssoap/t0865h01_AAL/sample_services/sec_echo/service/libmod_authn.so</rampc:AuthnModuleName>`
ReceiverCertificate	This denotes the Path to the receiver's public key.	`<rampc:ReceiverCertificate>/usr/tandem/nssoap/t0865h01_AAL/sample_services/sec_echo/service/alice_cert.cert</rampc:ReceiverCertificate>`
Certificate	This denotes the Path to the public key.	`<rampc:Certificate/usr/tandem/nssoap/t0865h01_AAL/sample_services/sec_echo/service/bob_cert.cert</rampc:Certificate>`
PrivateKey	This option denotes the path to the private key.	`<rampc:PrivateKey>/usr/tandem/nssoap/t0865h01_AAL/sample_services/sec_echo/service/bob_key.pem</rampc:PrivateKey>`
TimeToLive	This is used to create the `Expires` element in the TimeStampToken. This parameter can be used to specify the validity time for the message.	`<rampc:TimeToLive>10</rampc:TimeToLive>`
ClockSkewBuffer	This option is used to adjust the server time while validating the TimeStampToken, if the client and server clocks are not in sync. The TimeStampToken is considered valid only if `Message Created time in the Request < Current time of the Server < Message Expires time in the Request`. If ClockSkewBuffer is mentioned, it is added to the server time, while validating the TimeStampToken. The value is in seconds.	`<rampc:ClockSkewBuffer>10rampc:ClockSkewBuffer>10</rampc: ClockSkewBuffer>`
PrecisionInMilliseconds	If this flag is set to true, the Rampart module creates and validates the TimeStampToken with milliseconds precision.	`<rampc:PrecisionInMilliseconds>truerampc:PrecisionInMilliseconds>true</rampc:PrecisionInMilliseconds>`

Publishing the Security Requirements

The security requirements of the NonStop SOAP 4 service must be shared with the client program developer to build a secured SOAP message. The service developer provides the client program developer with a `policy.xml` file that contains the WS-SecurityPolicy assertions.

You can extract the `<wsp:Policy>.....</wsp:Policy>` tag from `services.xml` to create a `policy.xml` file that can be sent to the client.

Apart from the `policy.xml`, the service developer must share the username and password if the security requirement has username token policies. Service developer must communicate the username details to the client developer in a secured way.

If your security requirement has encryption, then you must also share the public key. The public key is used to encrypt SOAP message.

Configuring the Client to Invoke a Secured Web Service

Based on the security requirements of the service, you can configure the client to build a secure SOAP message.

Configuring the Axis2c Client

Before you start the client configuration, you must have the Axis2c libraries that include the Rampart module installed on your system. You can use the Rampart module library to build a secure SOAP message.

NonStop SOAP service is based on Axis2c and hence configuring the Axis2c client is straightforward with the artifacts shared by the service developer.

The steps used to configure the Axis2c client are:

1. Create a client repository

   Client repository is a local directory which is used by the Axis2c client to read the configuration files.

   a. Create a local directory and copy the `policy.xml` file shared by the service developer.

   b. Copy the `axis2.xml`, Axis2c libraries, and Rampart module from your Axis2c installation directory to the client repository.

   If you are planning to run the client program from the system where NonStop SOAP 4 server is installed, then you must not install Axis2c separately. The required objects can be copied from *<NonStop SOAP 4 Deployment Directory>*

   You can access the files from the following paths:

   <NonStop SOAP 4 Deployment Directory> – This folder contains the `axis2.xml` file.

   <NonStop SOAP 4 Deployment Directory>/lib – This folder contains the Axis2c libraries.

   <NonStop SOAP 4 Deployment Directory>/modules/rampart – This folder contains the Rampart module.

   c. You must ensure that the Rampart module is enabled and the security phase is activated in `axis2.xml`. The following shows the rampart module enabled in `axis2.xml`:

```
<axisconfig name="Axis2/C">
.............
<!-- ============= -->
<!-- Global Modules  -->
<!-- ============== -->
.........
<module ref="rampart"/>
<!-- ============= -->
<!-- Phases  -->
<!-- ============= -->
<phaseOrder type="inflow">
<!-- System pre defined phases-->
<phase name="Transport"/>
<phase name="PreDispatch"/>
<phase name="Dispatch"/>
<phase name="PostDispatch"/>
```

```
<!-- End system pre defined phases-->
<!-- After PostDispatch phase, module or service author can add any phase as required-->
<!-- User defined phases could be added here -->
<!--phase name="userphase1"/-->
<phase name="Security"/>
</phaseOrder>
<phaseOrder type="outflow">
<!-- User defined phases could be added here -->
<!--phase name="userphase1"/-->
<!--system predefined phase-->
<phase name="MessageOut"/>
<phase name="Security"/>
</phaseOrder>
......
</axisconfig>
```

2. Setting up the key store

   To encrypt the message that is sent to the server, the client must have the public key of the service. You can take the public key from the service developer.

   If the client wants to sign a message, then you must have a private key (X.509 certificate). You can use OpenSSL to create a certificate yourself or get it from Certification Authority.

3. Writing the password provider

   This is similar to the password provider library provided by the service provider. This library provides the password for the client's private key and for the username token.

4. Configuring the Rampart module

   The `policy.xml` file shared by the service developer, contains the Rampart specific configuration details, such as password provider library location, public key, and private key locations. You must change the path to your key store and the password provider library.

5. Invoking the service

   For invoking a Web service from Axis2c client, you must provide the client repository path in your program as follows:

   ```
   svc_client = axis2_svc_client_create(env, "/my/path/to/client/repository");
   ```

Configuring Non–Axis2c Clients

You can invoke the NonStop SOAP 4 service from other client programs, such as .NET or Java. The client program must have the required framework to build the secure SOAP message.

The two methods to configure the client are:

- WS-SecurityPolicy approach:

  The `policy.xml` can be used by the tools to build the stubs. The `policy.xml` contains Rampart specific assertions such as path to certificates. Based on your framework requirements, these assertions must be replaced or removed.

- Non WS-SecurityPolicy approach:

  The `policy.xml` shared by the service developer can be used for security requirements. Accordingly, you can configure the client program to meet the security requirements.

Extensible Modules

The service developer or a client program developer can customize the password lookup functionality of a Rampart module to suit the customer requirement. The service developer can also customize the authentication functionality of the Rampart module.

The following are the two Rampart modules that can be extended:

1. Password provider
   - The service verifies if the incoming request message has the proper password set. For this, the Rampart module must retrieve the password associated with the username from the service and compare it with the one that is set in the request message.
   - The password can be stored in a database, LDAP or some other storage based on the service requirements.

     To make the password retrieval logic more flexible, Rampart provides the password callback provider. You can create a password callback provider with an user defined password retrieval logic.

     The provider can be linked with the rampart by specifying the rampart module assertion "PasswordCallbackClass" in the security policies. While processing the request at runtime, rampart module invokes the password provider library by passing the username, to get the password for the user.
   - You can notice the sample password provider placed at "Sample Programs" (page 279). You can also use this provider to retrieve the password to read the private key for signing the message.

     The developer must add the rampart assertion 'EncryptionUser' in the policy file to retrieve the password for the private key.
   - You can use a flat file or database to associate the password to the user.

     Rampart module invokes the provider with the help of the username to retrieve the password. The password can be used to validate the UserNameToken or to read the private key.

   The client program developer who uses Axis2c service can also use the same steps to retrieve the password.
2. Authentication provider

   The Rampart module uses the username mentioned in the request message to invoke the password provider. The actual authentication happens inside the Rampart module. To customize the authentication functionality, the authentication provider can be added in the security policy by using the rampart assertion *AuthnModuleName*.

   For more information on the sample authentication provider, see "Sample Programs" (page 279).

Sample Programs

The developer can locate the WS-Security sample programs at *<NonStop SOAP 4 Installation Directory>/sample_services/sec_echo*

The descriptions about the different directories that are placed in the *sec_echo* folder are:

Table 24 WS-Security Sample Programs Directory

Directory	Description
`Service : ./service`	This directory contains the source of the sample service. The developer can configure different security scenarios that are described under the `secpolicy` folder. The service returns the same message that it receives from the client.
`Client: ./client`	This directory contains the source of the client program that can send secured SOAP messages.
`Security policies: ./secpolicy/scenarioX`	This directory provides several scenarios that displays how the WS-Security features can be configured through WS-SecurityPolicy language. For more information on WS-Security scenarios, see "WS-Security Scenarios " (page 281).

Table 24 WS-Security Sample Programs Directory *(continued)*

Directory	Description
`Password Provider : ./extensible_modules/password_provider`	This directory contains the source of sample password provider library that can be used to retrieve the password for UserNameToken and for a private key.
`Authentication Provider : ./extensible_modules/authn_provider`	This directory contains the source of a sample authentication provider library. The developer can customize the validation of UserNameToken by using this library.
`Keys : ./keys`	This directory contains all the certificates and private keys that are used in the samples. The files with the extension `.pem` contain the private key. The files with the extension `.cert` contain the certificate that has the public key.
`Data : ./data`	This directory contains the data files, such as passwords that are used in the samples.
`setup.sh`	This script sets up the client repository that is needed to run the client program.
`run_scenario.sh`	This script secures the client and service based on the given scenario number and then runs the client program.
`client_axis2.xml`	This file is used by the client program to engage the rampart module.
`README`	This file has the instructions to run the sample programs.

Functionality of the Sample Client and Server Program

The client builds a `"Hello"` request message and secures the message by using policies defined in the `policy.xml` file. Then, the client invokes the secured service and prints the response to the console. The client uses Axis2c Rampart module to secure the messages.

The server is an echo server. It responds with the same text it receives from the client. The response message is secured by using the policies defined in the `services.xml`. The server also uses Axis2c Rampart module to secure the messages.

Running the Web Services Security Sample Programs

Perform the following steps to run the sample WS-Security sample programs:

1. Set the *NSSOAP_HOME* environment variable to `<NonStop SOAP 4 Installation Directory>`.
2. `cd <NonStop SOAP 4 Installation Directory>/sample_services/sec_echo`
3. `./setup.sh`

   This script sets up the client repository that is needed to run the Web service security client program. Then, this script copies all the WS-Security scenarios. This script also copies source files of client and server program, password provider library, and authentication provider library.

   The script will ask for a path to set up the repository.
4. `cd <the directory where the client repository is created>/sample_services/sec_echo`.
5. Set the *AXIS2C_HOME* environment variable to `<NonStop SOAP 4 Deployment Directory>`.

   This environment variable is used to create and secure the sample service program.
6. `./run_scenario.sh <NSSOAP 4 server URL> <scenario no>`

   Here, `<NSSOAP 4 server URL>` is the URL where the Non Stop SOAP 4 server is deployed.

   For example, *http://15.146.233.43:1434/axis2c*.

<scenario no> is the scenario number that you want to run.

For additional information on the sample scenarios, see "WS-Security Scenarios " (page 281).

Based on the scenario number this script copies the policy.xml from ./ secpolicy/scenario(X) folder to the client repository. Then, this script deploys the sample server program and secures it on the NonStop SOAP 4 server. Further, this script executes the client program.

The <client repository>/logs directory contains the client program's log files.

The <NonStop SOAP 4 Deployment Directory>/logs contains the server program's log files.

NOTE: For each scenario, a separate service folder scenario(X) is created in <NonStop SOAP 4 Deployment Directory>/services.

WS-Security Scenarios

You can configure the sample client and service by using different security policies. For each scenario, the secpolicy directory contains services.xml and policy.xml files.

You can secure the service by using the services.xml and secure the client by using the policy.xml. The assertions that are explained in the following scenarios are available in the sample policy.xml and services.xml.

Scenario 1: Timestamp

This scenario demonstrates the steps that are required to add a timestamp to the request message. This also describes the steps required for using AsymmetricBinding.

To add a timestamp to the SOAP message, you can specify the timestamp details in the policy by adding the following assertion:

```
<sp:IncludeTimestamp/>
```

Next, you must specify the duration of the validity of the message. You can add the following rampart specific assertion:

```
<rampc:TimeToLive>360</rampc:TimeToLive>
```

The time duration is specified in seconds. The time difference is set to 360 seconds and if the message does not arrive within these limits, NonStop SOAP 4 server rejects the message. The following is a sample timestamp tag that is added to the security header:

```
<wsu:Timestamp xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd”>
<wsu:Created>2012-06-18T05:10:01.448Z</wsu:Created>
<wsu:Expires>2012-06-18T05:16:01.448Z</wsu:Expires>
</wsu:Timestamp>
```

You can use the following assertion in the services.xml file for configuring the NonStop SOAP 4 server to use ClockSkewBuffer for validating the timestamp:

```
<rampc:ClockSkewBuffer>60</rampc:ClockSkewBuffer>
```

Scenario 2: UsernameToken

To add a UsernameToken to the SOAP message, you have to specify the following:

1. The user

   ```
   <rampc:User>
   ```

2. The password type

   ```
   <rampc:PasswordType>
   ```

3. The password callback module

   ```
   <rampc:PasswordCallbackClass>
   ```

You can add the following assertions to the policy file:

```
<rampc:RampartConfig xmlns:rampc="http://ws.apache.org/rampart/c/policy">
<rampc:User>Alice</rampc:User>
<rampc:PasswordType>Digest</rampc:PasswordType>
<rampc:PasswordCallbackClass><Client Repository>/sample_services/sec_echo/extensible_modules/password_provider/
libpwcb.so</rampc:PasswordCallbackClass>
</rampc:RampartConfig>
```

Also, the following shows a sample inclusion of UsernameToken in the policy:

```
<sp:UsernameToken sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/Always"/>
```

On the server side, you can enable the Authentication Module, if you do not want to use the Rampart's inbuilt password authentication logic.

Scenario 3: Encryption

You can encrypt the SOAP message by using this scenario. To encrypt the message, you can refer to the AlgorithmSuite assertion that defines the different algorithms. This scenario uses *Basic256Rsa15* algorithm suite.

```
<sp:AlgorithmSuite>
<wsp:Policy>
<sp:Basic256Rsa15/>
</wsp:Policy>
</sp:AlgorithmSuite>
```

For additional information about algorithm suite, see http://specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf .

The scenario also includes the assertion that can be used to encrypt the whole body.

```
<sp:EncryptedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
<sp:Body/>
</sp:EncryptedParts>
```

The public key of the NonStop SOAP 4 service is used to encrypt the content and it is specified in `policy.xml`.

```
<rampc:ReceiverCertificate><Client
Repository>/sample_services/sec_echo/keys/bob_cert.cert</rampc:ReceiverCertificate>
```

To decrypt an incoming message, you must specify your own private key.

```
<rampc:PrivateKey><Client Repository>/sample_services/sec_echo/keys/alice_key.pem</rampc:PrivateKey>
```

Scenario 4: Signature

This scenario explains the steps required to sign the SOAP message. Similar to the encryption, to apply the signature you have to specify the signing parts, certificates, and keys. To specify which parts of the message must be signed, use the following assertion:

```
<sp:SignedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
<sp:Body/>
</sp:SignedParts>
```

The assertion in the sample signs the whole body.

Optionally, the following sample can be used if you want to sign a header:

```
<sp:SignedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
<sp:Header Namespace="http://www.w3.org/2005/08/addressing"/>
</sp:SignedParts>
```

For signature, you can use the following sample algorithm suite:

```
<sp:AlgorithmSuite>
<wsp:Policy>
<sp:Basic192Rsa15/>
</wsp:Policy>
</sp:AlgorithmSuite>
```

For additional information about algorithm suite, see http://specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf .

Scenario 5: Combining TimeStamp, UsernameToken, Encryption, and Signature with Protection order Sign->Encrypt

This scenario describes how TimeStamp, UsernameToken, Encryption, and Signature scenarios can be combined together.

The following assertion can be used to encrypt the signature:

```
<sp:EncryptSignature/>
```

The default protection order is SignBeforeEncrypting. The protection order property indicates the order in which integrity and confidentiality are applied to the message, in cases where both integrity and confidentiality are required. The SignBeforeEncrypting property indicates that the content must be signed first. The encryption is performed on the signed content.

Scenario 6: Combining TimeStamp, UsernameToken, Encryption, and Signature with Protection Order Encrypt->Sign

This scenario is similar to Scenario 5, except the protection order. This scenario demonstrates how the protection order “Encryption and then Sign” can be used. You can use the following assertion:

```
<sp:EncryptBeforeSigning/>
```

If this property is specified in the policies, the content is encrypted first and then the encrypted data is signed.

Scenario 7: Symmetric Binding. Encryption using Derived Keys

A derived key is a cryptographic key created from a password or other user data. Derived keys allow applications to create session keys as needed, eliminating the need to store a particular key. The use of the same session key (for example, when using Secure Conversation) for repeated message exchanges is sometimes considered a risk. To reduce this risk, Require Derived Keys is used.

The first six scenarios demonstrate the AsymmetricBinding. The following scenarios demonstrate the SymmetricBinding configuration. This scenario demonstrates how the encryption can be performed using derived keys. You can use the following assertion:

```
<sp:RequireDerivedKeys/>
```

Scenario 8: Symmetric Binding, Signature

This scenario demonstrates how Signature can be used with SymmetricBinding. For additional information about signature, see “Scenario 4: Signature” (page 282)

Scenario 9: Symmetric Binding. Both Encryption and Signature with Protection Order Encrypt->Sign

This scenario demonstrates how encryption and signature can be used with SymmetricBinding. The protection order is encrypt and then sign. This scenario is similar to Scenario 6 except the binding is different.

Scenario 10: Symmetric Binding. Both Encryption and Signature with Protection Order Sign->Encrypt

This scenario is similar to Scenario 9, except the protection order is different. The protection order is sign and encrypt.

Scenario 11: Symmetric Binding. Both Encryption and Signature with Protection Order Encrypt->Signature Encryption

This scenario is similar to Scenario 9, except that the signature is encrypted.

Scenario 12: Symmetric Binding. Both Encryption and Signature with Protection Order Sign->Encrypt. Signature Encryption

This scenario is similar to Scenario 10, except that the signature is encrypted.

Recommendations

While you design your security model for your services, you must study the potential threats to your model and what technologies or protocol can be used to mitigate each threat. The developers can use either the transport level security, such as HTTPS or WS-Security features or both to secure the services. Based on your company's security requirements, it is recommended to review the security considerations section in the WS-Security specification.

http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html#seccon

You must also consider WS-I Security for designing your security model. The document explains the different security challenges, threats, and security solutions.

http://www.ws-i.org/Profiles/BasicSecurity/2004-02/SecurityScenarios-0.15-WGD.pdf

A NonStop SOAP 4 Error and Warning Messages

If NonStop SOAP 4 encounters a situation that does not allow it to successfully serve a request, it logs an error message or a warning message in the `soaperror.log` file found in the `/logs` directory under the NonStop SOAP 4 deployment directory. The error and warning messages provides a brief description of the condition that prohibited NonStop SOAP 4 from serving the request.

You can set the level of log messages that you intend to capture by setting the `LOG_MODE` environment variable in the `itp_axis2.config` file. For more information on setting the log levels, see "Defining the Log Levels of the NonStop SOAP 4 Server" (page 178).

This appendix includes the following topics:

- "NonStop SOAP 4 Error Messages" (page 285)
- "NonStop SOAP 4 Warning Messages" (page 304)

NonStop SOAP 4 Error Messages

NonStop SOAP 4 error messages are classified in the following groups:

- "Client API Errors" (page 285)
- "NonStop SOAP 4 Engine Errors" (page 286)
- "NonStop SOAP 4 Utility Errors" (page 290)
- "Pathway Message Receiver Errors" (page 290)
- "CGI Errors" (page 294)
- "Transaction Management Errors" (page 294)

Client API Errors

NonStop SOAP 4 logs the following error messages when the client API functions are incorrectly used.

Blocking invocation expects response.

Cause

The client has used an IN-OUT service invocation. This makes the client code wait for the response.

Effect

The client waits for a response from the NonStop SOAP 4 server.

Recovery

Change the service invocation call in the client code and rebuild the client executable.

Cannot infer transport from URL.

Cause

The NonStop SOAP 4 server cannot understand the transport used from the endpoint reference.

Effect

The request is not served.

Recovery

Correct the endpoint reference and try sending the request again.

MEP cannot be NULL in the MEP client.

Cause

The client is using a message exchange pattern (MEP) that is not supported or identified by the NonStop SOAP 4 server.

Effect

The request is not served.

Recovery

Correct the MEP used by the client for service invocation and try sending the request again.

MEP mismatch.

Cause

There is a mismatch between the MEP used by the client and the one allowed by the service.

Effect

The request is not served.

Recovery

Correct the MEP used by the client for service invocation and try sending the request again.

NonStop SOAP 4 Engine Errors

NonStop SOAP 4 logs the following error messages because of runtime inconsistencies or error conditions in the NonStop SOAP 4 engine.

Invalid message context state.

Cause

The message context is either NULL or corrupt.

Effect

The request is not served.

Recovery

Check if the request message is well formed and try sending the request again.

Service accessed has invalid state.

Cause

The accessed service has no name for the server class mentioned in the `services.xml` file.

Effect

The request is not served.

Recovery

The service developer must mention a valid name for the server class.

Service not yet found.

Cause

The service accessed is not located by the NonStop SOAP 4 server.

Effect

The request is not served.

Recovery

Ensure that the service requested exists in the NonStop SOAP 4 deployment.

Invalid phase found in the phase validation.

Cause

NonStop SOAP 4 encountered an invalid phase during startup.

Effect

The request is not served.

Recovery

Ensure that the invalid phase is removed from the NonStop SOAP 4 message flow. Edit the `axi2.xml` file to verify the correct phase order of the NonStop SOAP 4 deployment.

Configuration file cannot be found.

Cause

NonStop SOAP 4 did not locate the `axis2.xml` configuration file.

Effect

The request is not served.

Recovery

Ensure that the valid `axis2.xml` file is present in the *<NonStop SOAP 4 Deployment Directory>* directory and restart the NonStop SOAP 4 server.

Data element of the OM node is NULL.

Cause

The AXIOM node does not contain any data element.

Effect

The request is not served.

Recovery

Ensure that all the modules and services deployed in NonStop SOAP 4 have their shared libraries and configuration files with the right permissions.

Invalid handler state.

Cause

The handler description and deployment details are not correct.

Effect

The request is not served.

Recovery

Ensure that the reported handler includes the deployment information mentioned in the corresponding `module.xml` configuration file. Also, ensure that the handler is not set to be deployed in a non-existent phase.

Invalid module reference encountered.

Cause

A non-existent module is referenced by the NonStop SOAP 4 server.

Effect

The request is not served.

Recovery

Ensure that the reported module exists in the NonStop SOAP 4 deployment. In case the module does not exist, either deploy the module or remove the reference to the module from the `services.xml` and `axis2.xml` file.

Module not found.

Cause

The referenced module does not exist or has not been engaged.

Effect

The request is not served.

Recovery

Check the module reference in the `axis2.xml` or `services.xml` file to ensure that the reported module is deployed and engaged to the NonStop SOAP 4 server.

Module validation failed.

Cause

The NonStop SOAP 4 server cannot engage the module.

Effect

The request is not served.

Recovery

Ensure that the shared libraries of all the modules are not corrupt and that they have proper permissions.

Module XML file is not found in the given path.

Cause

The NonStop SOAP 4 server cannot engage the module.

Effect

The request is not served.

Recovery

Ensure that the `module.xml` file is placed in the `<NonStop SOAP 4 Deployment Directory>/modules/<module_name>` location with proper permissions.

No dispatcher found.

Cause

The NonStop SOAP 4 server did not find any dispatchers.

Effect

The request is not served.

Recovery

Restart the NonStop SOAP 4 server.

Operation name is missing.

Cause

The NonStop SOAP 4 server found an invalid operation to load.

Effect

The request is not served.

Recovery

Check if the operation name is mentioned in the `services.xml` file.

Service XML file is not found in the given path.

Cause

The NonStop SOAP 4 server cannot locate the `services.xml` file.

Effect

The request is not served.

Recovery

Ensure that the `services.xml` file for the reported service exists in the `<NonStop SOAP 4 Deployment Directory>/services/<service_name>` location with proper permissions.

Invalid Service.

Cause

The NonStop SOAP 4 server cannot load the service.

Effect

The request is not served.

Recovery

In case of non-Pathway services, ensure that the shared library of the service is proper and that the corresponding `services.xml` file is present with appropriate permissions. In case of Pathway services, ensure that the `services.xml` file and the WSDL file are present with appropriate permissions.

Could not map the MEP URI to an Axis2/C MEP constant value.

Cause

An MEP other than the WSDL 2.0 specification is specified in the request.

Effect

The request is not served.

Recovery

Ensure that the client uses only the WSDL 2.0 specification MEPs.

Parameter container not set.

Cause

The NonStop SOAP 4 server cannot load the parameters specified in the configuration files.

Effect

The request is not served.

Recovery

Restart the NonStop SOAP 4 server and send the request again.

Parameter locked, cannot override.

Cause

The client or the service developer tried to programmatically override an immutable configuration parameter.

Effect

The request is not served.

Recovery

Set the `locked` attribute of the corresponding parameter to `false`.

Message context processing a fault already.

Cause

The NonStop SOAP 4 server tried to create a new fault message for a response that has already been assigned a fault context.

Effect

The request is not served.

Recovery

None.

Two services cannot have the same name, a service with same name already.

Cause

The NonStop SOAP 4 server encounters two services with the same name.

Effect

The request is not served.

Recovery

Ensure that all the services deployed in the NonStop SOAP 4 deployment have unique names assigned to them.

NonStop SOAP 4 Utility Errors

NonStop SOAP 4 logs the following error messages when utility functions are incorrectly used.

Could not open the file.

Cause

The NonStop SOAP 4 server cannot open the file.

Effect

The request is not served.

Recovery

Ensure that the reported file exists at the specified location with appropriate permissions.

Failed in creating DLL.

Cause

The NonStop SOAP 4 server cannot engage a shared library.

Effect

The request is not served.

Recovery

Check if the reported library is corrupt or has inappropriate permissions.

DLL description has invalid state of not having valid DLL create function, \ of valid delete function or valid dll_handler

Cause

The NonStop SOAP 4 server cannot engage a shared library.

Effect

The request is not served.

Recovery

Check if the reported library is corrupt.

Trying to do operation on invalid file descriptor.

Cause

The NonStop SOAP 4 server tried to read or write to an invalid file.

Effect

The request is not served.

Recovery

Check if the reported file exists with proper permissions.

Parameter not set.

Cause

The NonStop SOAP 4 server tried to read a configuration parameter that has not been set.

Effect

The request is not served.

Recovery

Check to see if the reported parameter is listed in the relevant configuration file.

Pathway Message Receiver Errors

NonStop SOAP 4 logs the following error messages because of incorrect processing in the Pathway message receiver.

WSDL file not found.

Cause

The NonStop SOAP 4 server cannot locate the WSDL file for the service.

Effect

The request is not served.

Recovery

Check if the WSDL file is placed in the service directory and the correct reference is specified in the corresponding `services.xml` file.

Error parsing WSDL file.

Cause

The NonStop SOAP 4 server cannot parse the WSDL file.

Effect

The request is not served.

Recovery

Check if the WSDL file is a well-formed XML file.

Communication with service failed.

Cause

The NonStop SOAP 4 server cannot communicate with the Pathway server class.

Effect

The request is not served.

Recovery

Ensure that the requested server class is running with no Pathway associated errors. In case the requested server class is not running, perform the recovery procedure recommended by the *NonStop TS/MP Pathsend and Server Programming Manual*.

Invalid response selection criteria in the `services.xml` file.

Cause

The NonStop SOAP 4 server cannot find one of the mandatory parameter tags related to response selection criterion in the `services.xml` file.

Effect

The request is not served.

Recovery

Edit the `services.xml` file for the requested service and correct the response selection criterion.

Request passed does not match the WSDL request reference.

Cause

The NonStop SOAP 4 server cannot match the request with the request XSD structure described in the WSDL file.

Effect

The request is not served.

Recovery

None.

Value of soapOccurs is greater than value of maxOccurs.

Cause

The client provided a value for the `soapOccurs` element, which exceeds the `maxOccurs` value specified in the WSDL file.

Effect

The request is not served.

Recovery

Pass a value for the `soapOccurs` element that is less than or equal to the value of `maxOccurs`.

Value of soapOccurs is less than value of minOccurs.

Cause

The client provided a value for the `soapOccurs` element, which is less than the `minOccurs` value specified in the WSDL file.

Effect

The request is not served.

Recovery

Pass a value for the `soapOccurs` element that is greater than or equal to the value of `minOccurs`.

Occurrence of an element in request was less than the minOccurs attribute specified in the WSDL file.

Cause

The client request contains occurrences of an element less than the `minOccurs` attribute specified for the element in the WSDL file.

Effect

The request is not served.

Recovery

Modify the request to include the occurrences of the element greater than or equal to the value of the `minOccurs` attribute.

Input element length does not match WSDL specifications. For details refer error log.

Cause

The length of an element in the client request is greater than the value specified in the WSDL file.

Effect

The request is not served.

Recovery

Modify the length of the element to be less than the value specified in the WSDL file.

No valid match found for response selection criteria in services.xml.

Cause

The NonStop SOAP 4 server cannot locate a valid response structure based on the response selection criterion specified in the `services.xml` file.

Effect

The request is not served.

Recovery

Check the response selection criterion mentioned in the `services.xml` file to match the response buffers.

The request size must be less than or equal to 2MB.

Cause

The NonStop SOAP 4 server cannot handle messages that are greater than 2MB in size.

Effect

The request is not served.

Recovery

None.

The request XML is invalid.

Cause

The request message is not a well formed XML message.

Effect

The request is not served.

Recovery

Ensure that the request is a well formed XML message.

Error occurred while processing the xsd:choice element.

Cause

An internal error occurred while processing the `xsd:choice` element.

Effect

The request is not served.

Recovery

None.

Occurrence of an element in request was more than the soapOccurs comment specified in WSDL.

Cause

The client request has more occurrences of an element greater than those permitted by the `soapOccurs` comment in the WSDL file.

Effect

The request is not served.

Recovery

Modify the client request to conform to the exact WSDL specification.

Invalid value for Boolean data type. Valid values are [true, yes, 1, false, no, 0].

Cause

The client request has specified a value for a Boolean data type that is different than the permitted values, namely `true`, `yes`, `false`, `no`, `1`,and `0`.

Effect

The request is not served.

Recovery

Modify the client request to use one of the permitted values for the Boolean data type.

No valid match found for the response selection criteria in services.xml. The associated transaction has been aborted.

Cause

The NonStop SOAP 4 server cannot locate a valid response structure based on the response selection criterion specified in the `services.xml` file. The transactions, if any, associated with the request are also rolled back.

Effect

The request is not served.

Recovery

Check the response selection criterion mentioned in the `services.xml` file to match the response buffers.

CGI Errors

NonStop SOAP 4 logs the following error messages when there is a communication failure between the iTP WebServer and the CGI interface.

CGI failed while writing response.

Cause

An internal error occurred between iTP WebServer and CGI communication.

Effect

The request is not served.

Recovery

Resend the request.

CGI failed while writing response.

Cause

An internal error occurred between iTP WebServer and CGI communication.

Effect

The request is not served.

Recovery

Resend the request.

Transaction Management Errors

NonStop SOAP 4 logs the following error messages when faulty session or transaction headers are found.

Invalid value for TMFTransactionSupport parameter in services.xml file.

Cause

The *TMFTransactionSupport* parameter carries a value other than `Required`, `Supports` ,or `Never`.

Effect

The transaction is not started.

Recovery

Send the request that conforms to the transaction settings for the requested service.

Session ID not found.

Cause

The client request does not contain a session ID that is required to perform the requested TMF operation.

Effect

The request is not served.

Recovery

Send the request with the session ID.

Mandatory 'Command' tag absent in transaction header.

Cause

The client request does not contain the mandatory `Command` attribute as part of the transaction header block.

Effect

The request is not served.

Recovery

Send the request again with the `Command` attribute present in the transaction header block.

Invalid transaction ID in transaction header.

Cause

The client request contains an invalid transaction identifier.

Effect

The request is not served.

Recovery

Send the request again with a valid transaction identifier.

Invalid element in transaction header. Refer logs for detail.

Cause

The client request contains an invalid element or a combination of mutually exclusive elements in the transaction header block.

Effect

The request is not served.

Recovery

Send the request again with a valid combination of transaction header block elements as specified in the *TMF Reference Manual*.

Invalid transaction timeout in the transaction header.

Cause

The client request contains an invalid value for the `TimeOut` attribute in the transaction header block.

Effect

The request is not served.

Recovery

Send the request again with a valid value for the `TimeOut` attribute in the transaction header block.

Error occurred while resuming transaction.

Cause

TMF failed to resume the transaction specified by the transaction identifier supplied by the client.

Effect

The request is not served.

Recovery

The error message will report the associated TMF error number. To identify the recommended recovery procedure, see the *TMF Management Manual*.

Error occurred while aborting transaction.

Cause

TMF failed to abort the transaction specified by the transaction identifier supplied by the client.

Effect

The request is not served.

Recovery

The error message will report the associated TMF error number. To identify the recommended recovery procedure, see the *TMF Management Manual*.

Service does not support transactions.

Cause

The client request tried to use transactions against a service that does not use transactions.

Effect

The request is not served.

Recovery

Access the service without using any transaction or session header block.

Error occurred while ending transactions.

Cause

TMF failed to commit the transaction specified by the transaction identifier supplied by the client.

Effect

The request is not served.

Recovery

The error message will report the associated TMF error number. To identify the recommended recovery procedure, see the *TMF Management Manual*.

SoapAdminCL Errors

The errors generated by the SoapAdminCL tool can be categorized as:

- "Pathway Service Definition Errors" (page 296)
- "Memory Allocation Errors" (page 298)
- "SoapAdminCL Error Messages" (page 298)

Pathway Service Definition Errors

The Pathway service definition error messages report invalid values in the SDL file.

Error>> Invalid Pathmon Name: <Pathmon Name Specified in the SDL>.

Cause

The PATHMON name specified in the SDL file has an invalid format.

Effect

The command fails.

Recovery

Enter the PATHMON name in the valid format (that is, `$ <alphabet> <alphanumeric characters>`) and run the SoapAdminCL command. Ensure that the PATHMON name does not exceed seven characters.

Error>> Invalid Serverclass Name: <Server Class Name Specified in the SDL>.

Cause

The ServerClassName attribute in the SDL file has an invalid format.

Effect

The command fails.

Recovery

Enter the ServerClassName attribute in the valid format (that is, alphanumeric characters and hyphens) and run the SoapAdminCL command.

Error>> Length of Server Class:<ServerClass> is greater than 15.

Cause

The serverclass name has more than 15 characters.

Effect

The command fails.

Recovery

Update the `serverclass` name to contain less than or equal to 15 characters, and run the `SoapAdminCL` command.

Error>> Invalid DDL Dictionary Name: <DDL Dictionary Specified>

Cause

The Guardian location specified in the `DDLDictionaryLocation` is invalid.

Effect

The command fails.

Recovery

Specify a valid Guardian location in the `DDLDictionaryLocation` and run the `SoapAdminCL` command.

Parser - Error>> Invalid DDL Definition Name: <DDL Definition Specified>

Cause

A DDL definition name associated with either the `RequestInfo` element or the `ResponseInfo` element in the SDL file is invalid.

Effect

The command fails.

Recovery

Specify a valid DDL definition name in the `RequestInfo` element and the `ResponseInfo` element in the SDL file. Ensure that the DDL definition name contains only alphanumeric characters and an underscore.

Parser - Error>> Response Selection Node <path> Attribute: <Attribute name>, Value:<Given value> is not a number.

Cause

The `Start` and `End` attributes do not have numeric values or have other invalid characters or special symbols in them.

Effect

The command fails.

Recovery

Update the `Start` and `End` attributes with numeric values and run the `SoapAdminCL` command.

Parser - Error>> Response Selection Node <path> Attribute:BufVal needs to be a Number when the Attribute: ConversionOp is a number.

Cause

In the `ResponseSelection` element of the SDL file, the `ConversionOp` attribute specifies decimal conversion; however the value of `BufVal` is not numeric.

Effect

The command fails.

Recovery

If the `ConversionOp` attribute specifies decimal conversion, ensure that you enter numeric value for `BufVal` and then run the `SoapAdminCL` command.

Error>> For the Response Selection Element in the path <XML Path to the Response Selection element>, either all the "Start', "End", "BufVal",attributes must be specified or none.

Cause

The syntax of the `ResponseSelection` element is incorrect.

Effect

The command fails.

Recovery

Correct the syntax of the `ResponseSelection` element to specify all or none of the attributes listed in the message, and run the `SoapAdminCL` command.

Memory Allocation Errors

The following log messages signify memory allocation errors.

Parser - Error>>Internal error(SdlPway:parse) in the construction

Cause

Insufficient memory to run the `SoapAdminCL` command.

Effect

The command fails.

Recovery

Run the `SoapAdminCL` command after some time. If the problem persists, contact the system administrator.

Parser - Error>>SdlPway::parse - Unable to allocate memory

Cause

Insufficient memory to run the `SoapAdminCL` command.

Effect

The command fails.

Recovery

Run the `SoapAdminCL` command after some time. If the problem persists, contact the system administrator.

Parser - Error>>SdlPway::getSrvrClasses - Unable to allocate memory

Cause

Insufficient memory to run the `SoapAdminCL` command.

Effect

The command fails.

Recovery

Run the `SoapAdminCL` command after some time. If the problem persists, contact the system administrator.

`SoapAdminCL` Error Messages

SOAPADMIN ERROR >>SOAP server is not running at the defined location

Cause

The NonStop SOAP 4 server is not running at the defined server SDL location.

Effect

None

Recovery

You can restart the server at the defined SDL location. Or correct the location in the SDL file and re-run SOAPAdminCL.

SOAPADMIN ERROR >>Duplicate entry for field name <field name> in DDL Definition

Cause

This is due to the duplicate entry of the field name.

Effect

NonStop SOAP service deployment files will not be generated. Then, `SoapAdminCL` will exit.

Recovery

Remove the duplicate entry and use the same command again.

SOAPADMIN ERROR >>Duplicate entry of DDL Definition <DDLDefinition> in SDL

Cause

This is due to the duplicate entry of the DDL Definition name in SDL Requestinfo/ResponseInfo.

Effect

NonStop SOAP service deployment files will not be generated. Then, SoapAdminCL exits.

Recovery

Remove the duplicate entry and use the same command again.

SOAPADMIN ERROR >> Optional Element <element name> is used as soap occurs depending on for <element name> You cannot use optional element as SOAP OCCURS DEPENDING ON field.

Cause

The `@SOAP_OCCURS_DEP_ON` field is defined as optional using `@SOAP_OPTIONAL DDL` comment.

Effect

The command fails.

Recovery

The DDL field defined as optional using `@SOAP_OPTIONAL` cannot be used in `@SOAP_OCCURS_DEP_ON`. Resolve the conflict in the DDL file, regenerate the DDL dictionaries, and run the `SoapAdminCL` command.

SOAPADMIN ERROR >> Optional Element <element name> is used as occurs depending on for <element name>. You cannot use optional element as OCCURS DEPENDING ON field.

Cause

The `OCCURS DEPENDING ON` clause cannot be applied to a field which is defined as optional using `@SOAP_OPTIONAL DDL` comment.

Effect

The command fails.

Recovery

The DDL field defined as optional using `@SOAP_OPTIONAL` cannot be used in `OCCURS DEPENDING ON` clause. Resolve the conflict in the DDL file, regenerate the DDL dictionaries, and run the `SoapAdminCL` command.

SOAPADMIN ERROR >> Error in definition of the service <service name>. The attribute "SoapDDLAttribute" cannot have a value "yes" when the attribute "SoapMessageType" is set to "rpc".

Cause

The `SoapDDLAttribute` attribute has a value `yes` when the `SoapMessageType` attribute is set to `rpc`.

Effect

The command fails.

Recovery

Resolve the conflict between the `SoapDDLAttribute` attribute and the `SoapMessageType` attribute and run the `SoapAdminCL` command.

SOAPADMIN ERROR >> Error processing DDL definition <definition name> for Service <service name>. The SOAP DDL comment tags "@SOAP_ATTRIBUTE" and "@SOAP_ELEMENT" cannot be present for a single DDL field.

Cause

The `@SOAP_ATTRIBUTE` and `@SOAP_ELEMENT` SOAP DDL comment tags are present for a single DDL field.

Effect

The command fails.

Recovery

The same DDL field cannot be exposed either as an element or attribute. Resolve the conflict in the DDL file, regenerate the DDL dictionaries, and run the `SoapAdminCL` command.

SOAPADMIN ERROR >> Error processing DDL definition <definition name> for service <service name>. The "@SOAP_BASE64" DDL comment tag cannot be present at a group level DDL field <field name>.

Cause

The `@SOAP_BASE64` DDL comment tag is present at a group level DDL field `<field name>`.

Effect

The command fails.

Recovery

The `@SOAP_BASE64` DDL comment is applicable at the leaf node level. This is not applicable for the group field. Therefore, you must set the DDL comment to the appropriate DDL field, regenerate the DDL dictionaries, and then run the `SoapAdminCL` command.

SOAPADMIN ERROR >> The depends on field <field name> applied to the @SOAP_OCCURS_DEP_ON comment is not a numeric type. Exiting.

Cause

The `depends on` field `<field name>` applied to the `@SOAP_OCCURS_DEP_ON` DDL comment does not have a numeric value.

Effect

The command fails.

Recovery

The `@SOAP_OCCURS_DEP_ON DDL` comment guides the NonStop SOAP 4 server to limit the number of occurrences of the field in the SOAP request/response. It must have a numeric value. Set the proper field for this comment, regenerate the DDL dictionaries, and run the `SoapAdminCL` command.

SOAPADMIN ERROR >> The depends on field <field name>, applied to the @SOAP_OCCURS_DEP_ON comment is not defined. Exiting.

Cause

The `depends on` field `<field name>` applied to the @SOAP_OCCURS_DEP_ON comment is not defined.

Effect

The command fails.

Recovery

Define the `depends on` field `<field name>` applied to the `@SOAP_OCCURS_DEP_ON` DDL comment, regenerate the DDL dictionaries, and then run the `SoapAdminCL` command.

SOAPADMIN ERROR >> The @SOAP_OCCURS_DEP_ON field cannot be applied to <field name> as it is not an OCCURS field. Exiting.

Cause

The `@SOAP_OCCURS_DEP_ON` comment is applied to the field, which is not an `OCCURS` field.

Effect

The command fails.

Recovery

Check the validity of the DDL field and resolve the conflict. Regenerate the DDL dictionaries and run the `SoapAdminCL` command.

SOAPADMIN ERROR >> The depends on field for the SOAP_OCCURS_DEP_ON comment tag was not found.

Cause

The `depends on` field is not defined in the DDL file mentioned in the `@SOAP_OCCURS_DEP_ON` comment.

Effect

The command fails.

Recovery

Check the validity of the DDL field and resolve the conflict. Regenerate the DDL dictionaries and then run the `SoapAdminCL` command.

SOAPADMIN ERROR >> The @SOAP_OCCURS_DEP_ON comment tag cannot be applied more than once to the same field.

Cause

The `@SOAP_OCCURS_DEP_ON` comment appears more than once for the same element.

Effect

The command fails.

Recovery

Ensure that the `@SOAP_OCCURS_DEP_ON` comment appears only once for every element, regenerate the DDL dictionaries, and run the `SoapAdminCL` command.

SOAPADMIN ERROR >> The depends on field <field-name> for the SOAP_OCCURS_DEP_ON comment tag cannot be uniquely identified.

Cause

The `depends on` field is defined more than once in the DDL file mentioned in the `@SOAP_OCCURS_DEP_ON` comment.

Effect

The command fails.

Recovery

Select a unique name for the `depends on` field, regenerate the DDL dictionaries, and run the `SoapAdminCL` command.

Parser - Error while parsing the input at Line: x, Column: y Detail Error: <Error details>

Cause

The SDL file passed is not a valid XML document.

Effect

The command fails.

Recovery

Ensure that the SDL file is well formed and run the `SoapAdminCL` command.

SOAPADMIN Parser - Error >> The service <service name> already exists in the SDR within a different hierarchy.

Cause

The same service name appears twice in the SDL file.

Effect

The command fails.

Recovery

Select a unique name for each service in the SDL file and run the `SoapAdminCL` command.

SOAPADMIN ERROR >> HTML client file already exists...

Cause

The HTML client file exists at the mentioned location.

Effect

The command fails.

Recovery

Check whether the service is already deployed at the mentioned location. If you want to regenerate the files or redeploy the services at the mentioned location, use the `SoapAdminCL` command with the `-f` or `-force` option.

SOAPADMIN ERROR >> WSDL file already exists...

Cause

The WSDL file exists at the mentioned location.

Effect

The command fails.

Recovery

Check whether the service is already deployed at the mentioned location. If you want to regenerate the files or redeploy the services at the mentioned location, use the `SoapAdminCL` command with the `-f` or `-force` option.

SOAPADMIN ERROR >> Service configuration file already exists...

Cause

The service configuration file exists at the mentioned location.

Effect

The command fails.

Recovery

Check whether the service is already deployed at the mentioned location. If you want to regenerate the files or redeploy the services at the mentioned location, use the `SoapAdminCL` command with the `-f` or `-force` option.

SOAPADMIN ERROR >> Error in definition of service "<service_name>". Both attributes "GenerateSessionHeader" and "GenerateTransactionHeader" cannot have a value "yes"

Cause

The `GenerateSessionHeader` and the `GenerateTransactionHeader` attributes are set to `yes`.

Effect

The command fails.

Recovery

Ensure that the `GenerateSessionHeader` and the `GenerateTransactionHeader` attributes are not set to `yes`.

WSDL2PWY and WSDL2C Error Messages

The WSDL2PWY and WSDL2C utilities return the following errors when invalid options are specified.

Error: The -o option was specified without a specifying the folder or with an improper folder name. Please note the tool usage:

Cause

The output folder is not specified or the folder name is incorrect.

Effect

The WSDL processing stops and the output files are not generated.

Recovery

Specify a valid folder name and run the tool.

Error: The –uri option was not specified. Please note the tool usage:

Cause

The –uri option is not specified for the WSDL file.

Effect

The WSDL processing stops and the output files are not generated.

Recovery

Specify the –uri option before the WSDL file name and run the tool.

Error: The -wv option was specified without a specifying version number. Please note the tool usage:

Cause

The version number is not specified with the -wv option.

Effect

The WSDL processing stops and the output files are not generated.

Recovery

Specify the valid WSDL version (1.1 or 2.0) and run the tool.

Error: The parameter specified for -wv option is invalid. Please note the tool usage:

Cause

The version number is not specified with the -wv option.

Effect

The WSDL processing stops and the output files are not generated.

Recovery

Specify the valid WSDL version (1.1 or 2.0) and run the tool.

Error: The -uri option was specified without a WSDL location. Please note the tool usage:

Cause

The WSDL file name is not specified.

Effect

The WSDL processing stops and the output files are not generated.

Recovery

Specify a WSDL file name with –uri option before the file name and run the tool.

Error: Error Parsing Options

Error Details: The option -g was specified without specifying the option –ss

Cause

The –g generate all option is specified without specifying the –ss option.

Effect

The WSDL processing stops and the output files are not generated.

Recovery

Specify the `-ss` option with the `-g` option and run the tool.

Error: Error Parsing Options

Error Details: The option -sd was specified without specifying the option –ss

Cause

The `-sd` generate service descriptor option is specified without specifying the `-ss` option.

Effect

The WSDL processing stops and the output files are not generated.

Recovery

Specify the `-ss` option with the `-sd` option and run the tool.

Error: Error Parsing Options

Error Details: An invalid option `<option>` been specified at the beginning. Unable to process the request

Cause

An invalid first option is specified without - prefix.

Effect

The WSDL processing stops and the output files are not generated.

Recovery

Specify valid options, prefixed with - and run the tool.

NonStop SOAP 4 Warning Messages

NonStop SOAP 4 logs the following warning messages:

PathmonName not set in the message context.

Cause

The PATHMON name is not set using Message Receiver User Functions.

Effect

The NonStop SOAP 4 server continues to serve the request.

Recovery

None.

is_txn_started not set in the message context.

Cause

The transaction module is not attached to the NonStop SOAP 4 deployment.

Effect

The NonStop SOAP 4 server continues to serve the request.

Recovery

None.

InputEncoding not set in message context.

Cause

The input encoding is not specified in the SOAP request.

Effect

The NonStop SOAP 4 server continues to serve the request.

Recovery

None.

OutputEncoding not set in message context.

Cause

The output encoding is not specified in the `services.xml` file.

Effect

The NonStop SOAP 4 server continues to serve the request.

Recovery

None.

Body Stream not available for writing.

Cause

A client sends an empty HTTP request.

Effect

Internal server error is returned.

Recovery

The client needs to send a well-formed SOAP message.

Byte buffer not available for writing.

Cause

A service sends an empty response.

Effect

The client receives an empty response.

Recovery

None.

Creation of Content-Type string failed.

Cause

The `content-type` of HTTP request is not understood by NonStop SOAP 4.

Effect

The request is not served.

Recovery

Set the `content-type` of the HTTP request as `application/soap+xml` (for SOAP 1.2) or `text/xml` (SOAP 1.1) and resend the message.

Provided client repository path %s does not exist or no permission to read, therefore set axis2c home to DEFAULT_REPO_PATH.

Cause

The `AXIS2C_HOME` variable is not set to the correct location or the location does not have read permission.

Effect

Log messages will be returned to the `DEFAULT_REPO_PATH` directory instead of the `AXIS2C_HOME/logs` directory.

Recovery

Set the correct location in the `AXIS2C_HOME` variable and ensure that it has read permission.

Retrieving Axis2 configuration from Axis2 configuration context failed. Initializing modules failed.

Cause

The `axis2.xml` file is corrupt.

Effect

None.

Recovery

Ensure that the `axis2.xml` file is well formed.

Retrieving Axis2 configuration from Axis2 configuration context failed, Loading services failed.

Cause

The `axis2.xml` file is corrupt.

Effect

None.

Recovery

Ensure that the `axis2.xml` file is well formed.

Retrieving Axis2 configuration from Axis2 configuration context failed. Initializing transports failed.

Cause

The `axis2.xml` file is corrupt.

Effect

None.

Recovery

Ensure that the `axis2.xml` file is well formed.

Worker is not ready. Cannot serve the request.

Cause

The server is busy and cannot handle more requests.

Effect

None.

Recovery

Send the request after some time or increase the instances of the `Axis2CGI` serverclass specified in the `itp_axis2.config` file.

Error occurred in processing request.

Cause

A request processing error has occurred.

Effect

NonStop SOAP 4 will try to process the request using a new instance of worker.

Recovery

None.

An invalid option was specified. However, the WSDL was parsed. Please note the tool usage

Cause

An invalid option is specified.

Effect

This message is a warning, and the invalid option is ignored. The WSDL processing continues and output files are generated.

Recovery

Ensure the valid options are specified.

Skipping the validation for rpc type soap messages.

Cause

The `SoapMessageType` parameter is set to `rpc` in the `services.xml` file.

Effect

Validation of the request is skipped even when validation module is attached.

Recovery

Recommended to set the `SoapMessageType` parameter to `document` in the `services.xml` file.

B Install Files and Folders

Verifying the Extracted Product Files

The following files and directories are extracted in the *<NonStop SOAP 4 Installation Directory>*:

- LICENSE - the Apache license file
- `axis2.xml` - the default NonStop SOAP 4 configuration file
- `/bin` - includes the executable object for the NonStop SOAP 4 server and the installation script
  - `axis2cgi.pway` - the server object to access SOAP services through iTP WebServer
  - `deploy.sh` - the script file that sets up the deployment directory environment and deploys the sample services
- `/include` - includes all the header files to build the API-based clients and services
- `/lib` - includes the libraries required by NonStop SOAP 4
  - `libaxis2_axiom.so` - the NonStop SOAP 4 object model library
  - `libaxis2_engine.so` - the NonStop SOAP 4 engine
  - `libaxis2_http_receiver.so` - the NonStop SOAP 4 Hypertext Transfer Protocol (HTTP) transport for request handling
  - `libaxis2_http_sender.so` - the NonStop SOAP 4 HTTP transport for response handling
  - `libaxis2_parser.so` - the NonStop SOAP 4 parser
  - `libaxis2_pway_receiver.so` - the NonStop SOAP 4 Pathway Message Receiver
  - `libaxis2_pway_xml_receiver.so` - the NonStop SOAP 4 Pathway XML Message Receiver
  - `libaxutil.so` - the NonStop SOAP 4 utility
  - `libneethi.so` - the NonStop SOAP 4 WS-Policy implementation
  - `libtsmp20.so` - TS/MP 2.0 library
  - `libtsmp24.so` - TS/MP 2.4 library
  - `libxml2.so` - the XML parser library
  - `/tools` - includes all the JAR files required to run the `WSDL2C` and `WSDL2PWY` tool
- `/modules` - includes a sub-directory for each of the following NonStop SOAP 4 modules
  - `/encoding` - includes the `encoding` module
    - `libaxis2_mod_encoding.so` – the `encoding` module
    - `module.xml` - the module configuration file for the `encoding` module
  - `/rampart` - includes the `rampart` module
    - `libmod_rampart.so` – the `rampart` module
    - `module.xml` - the module configuration file for the `rampart` module

- /transaction - includes the transaction module
  - libaxis2_mod_transaction.so – the transaction module
  - module.xml – the module configuration file for the transaction module
- /validation - includes the validation module
  - libValidationModule.so – the validation module
  - module.xml – the module configuration file for the validation module

- /sample_services
  - /echo
    - /client
      - Makefile - Makefile for the echo client application
      - echo.c - source file for the echo client
    - echo.exe - an echo client built using the NonStop SOAP 4 client APIs
    - libecho.so - an echo service built using the NonStop SOAP 4 service APIs
    - /service
      - Makefile - Makefile for the echo sample application
      - echo.c - source file for the echo sample application
      - echo.h - header file for the echo sample application
      - echo_skeleton.c - source file for the echo service
      - echoddl - DDL file for the echo sample application
      - echosdl.xml - SDL file for the echo sample application
    - services.xml - NonStop SOAP 4 service configuration file for the echo service
  - /empdb
    - /src
      - Makefile - Makefile for the empdb service
      - README - readme file describing the steps to use the empdb application
      - emp.c - source file for the empdb service
      - emp.h - header file for the empdb service
      - empsdl.xml - SDL file for the empdb sample application
  - /math
    - /client - includes the source files for the math client
      - Makefile - Makefile for the math client
      - axis2_stub_mathService.c - source file for the math client
      - axis2_stub_mathService.h - header file for the math client
    - /service - includes the source file and configuration file for the math service
      - Makefile - Makefile for the math service
      - axis2_skel_mathService.c - source file for the math service
      - axis2_skel_mathService.h - header file for the math service
      - axis2_svc_skel_mathService.c - source file for the math service
      - mathddl - DDL file for the math service
      - mathsdl.xml - SDL file for the math service

- /modules
  - /empdb_MRUF
    - /src
      - Makefile - Makefile for the empdb Message Receiver User Functions module
      - empdb_MRUF.c - source file for the empdb Message Receiver User Functions module
      - empdb_MRUF.h - header file for the empdb Message Receiver User Functions module
  - /logging_MRUF
    - /src
      - Makefile - Makefile for the logging Message Receiver User Functions module
      - logging_MRUF.c - source file for the logging Message Receiver User Functions module
      - logging_MRUF.h - header file for the logging Message Receiver User Functions module
  - /mod_empdb
    - empdb.xsl - input stylesheet file for the mod_empdb module
    - module.xml - the module configuration file for the mod_empdb module
    - /src
      - Makefile - Makefile for the sample mod_empdb module
      - empdb_post_process_handler.c - source file for the post_process handler for the mod_empdb module
      - empdb_pre_process_handler.c - source file for the pre_process handler for the mod_empdb module
      - mod_empdb.c - source file for the mod_empdb module
      - mod_empdb.h - header file the mod_empdb module
  - /mod_logging
    - module.xml - the module configuration file for the mod_empdb module
    - /src
      - Makefile - Makefile for the sample mod_logging module
      - logging_in_handler.c - source file for the in handler, for the mod_logging module, for processing the request
      - logging_out_handler.c - source file for the out handler, for the mod_logging module, for processing the response
      - mod_logging.c - source file for the mod_logging module
      - mod_logging.h - header file the mod_logging module

- /reflector
  - /conf
    - pathConf - Pathway configuration file for the reflector service
  - /src
    - reflector.c - source file for the reflector service
    - reflectorddl - DDL file for the reflector service
    - reflectorsdl.xml - SDL file for the reflector service
- sec_echo
  - README - explains how to set up sample client repository and how to run the scenario samples provided.
  - /client
    - sec_echo.txe - sample client application used to access the sec_echo service.
    - /src
      - Makefile - Makefile to build the sample sec_echo client.
      - echo.c - source file for the sample sec_echo client.
  - client_axis2.xml - axis2.xml for the client repository.
  - data - passwords file containing username and password for authentication.
  - extensible_modules
    - authn_provider - A module used for authentication.
      - Makefile - Makefile to build the authentication module.
      - authn_module.c - source file for authentication module.
    - password_provider - A module used to get password for a specific user from the "passwords" file.
      - /src
        - Makefile - Makefile to build the password provider module.
        - password_module.c - source file for password provider module.
  - keys - contains sample certificates and private keys.
  - run_scenario.sh - A script used to run the sample scenarios.
  - secpolicy - contains policies and services.xml files for all the sample scenarios.
  - /service
    - libsec_echo.so - sec_echo service implementation library.
    - src
      - Makefile - Makefile to build the sec_echo service.
      - echo.c - source file for the sec_echo service.

        - echo.h - header file for the sec_echo service.
        - echo_skeleton.c - source file for the sec_echo service.
    - setup.sh - A script to set up sample WS-Security client repository used to run the scenarios.
- /tools - includes the tools distributed with NonStop SOAP 4
    - SoapAdminCL.txe - tool to generate the HTML clients, the WSDL file, SOAP request and response XML files, XML schema files, and the services.xml file using the NonStop SOAP 4 Service Definition Language (SDL) file.
    - SoapAdminCL - A tool which is interface to SoapAdminCL.txe tool. It validates if the NonStop SOAP Server mentioned in the Service Definition Language (SDL) file is running or not (if -w option is specified) and then calls the SoapAdminCL.txe.
    - WSDL2C - tool to generate services and clients in the C programming language using the service-specific WSDL file.
    - WSDL2PWY - tool to generate services and clients in the C programming language using the service-specific WSDL file.
    - /adminserver - tool to use the NonStop SOAP 4 Administration Utility.
        - libadminserver.so -DLL file for adminserver.

NOTE: If all the files listed in this section are not present, delete all the files, check for write permissions, and repeat the installation procedure.

Verifying the Deployed Files

The following files and directories are deployed in the *<NonStop SOAP 4 Deployment Directory>*:

- axis2.xml - is the NonStop SOAP 4 configuration file
- /bin - includes the executable object for the NonStop SOAP 4 server and the installation script
    - axis2cgi.pway - is a soft link to axis2cgi.pway in *<NonStop SOAP 4 Installation Directory>*
- /client - is the NonStop SOAP 4 client directory
- deploy_instructions - includes instructions required to deploy the NonStop SOAP 4 server under iTP WebServer
- /include - is a soft link to the include directory in the *<NonStop SOAP 4 Installation Directory>*
- itp_axis2.config - is the iTP WebServer configuration file
- /lib - is a soft link to the /lib directory in the *<NonStop SOAP 4 Installation Directory>*
- /logs - is the NonStop SOAP 4 log file directory
- /modules
    - /encoding - is a soft link to the encoding module directory in *<NonStop SOAP 4 Installation Directory>*
    - /rampart - is a soft link to the rampart module directory in *<NonStop SOAP 4 Installation Directory>*

    - /transaction - is a soft link to the transaction module directory in <NonStop SOAP 4 Installation Directory>
    - /validation - is a soft link to the validation module directory in <NonStop SOAP 4 Installation Directory>
- /services - is the NonStop SOAP 4 service deployment directory

NOTE: If all the listed files are not present, run the deploy.sh script again, as mentioned in Step 2 of "Running the Deployment Script" (page 39).

The following files and directories must be present in the <NonStop SOAP 4 Deployment Directory> directory:

- /services
    - /echo
        - /client
            - Makefile - Makefile for the echo client
            - echo.c - source file for the echo client
        - libecho.so - echo service built using NonStop SOAP 4 server APIs
        - /service
            - Makefile - Makefile for the echo service
            - echo.c - source file for the echo service
            - echo.h - header file for the echo service
            - echo_skeleton.c - source file for the echo service
            - echoddl - DDL file for the echo service
            - echosdl.xml - SDL file for the echo service
        - services.xml - NonStop SOAP 4 service configuration file for the echo service
- /client
    - /echo
        - echo.exe - client for the echo service

C SoapAdminCL DDL datatype to XML datatype conversion

SoapAdminCL converts the DDL datatypes to the corresponding XML schema type to generate the WSDL. This maps the data structure defined in DDL to the XML schema in the WSDL.

Table 25 SoapAdminCL conversion table for DDL datatype to XML datatype Conversion:

DDL Datatype	Schema datatype
PIC A(10)	xsd:string maxLength = 10
PIC X(10)	xsd:string maxLength = 10
PIC 9(10)	xsd:unsignedLong totalDigits=10
PIC A(2)X(10)9(2)A(5)	xsd:string maxLength = 19
PIC SV9(3)	xsd:decimal totalDigits = 3, fractionDigits = 3
PIC 9V9(2)	xsd:decimal totalDigits = 3, fractionDigits = 2
PIC T9V9	xsd:decimal totalDigits = 2, fractionDigits = 1
PIC 9(2)T	xsd:short totalDigits = 2
PIC N(10)	xsd:string maxLength = 20
PIC 9(4) COMP	xsd:unsignedShort
PIC S9(4) COMP	xsd:short
PIC 9(5) COMP	xsd:unsignedInt
PIC S9(5) COMP	xsd:int
PIC 9(10) COMP	xsd:unsignedLong. Not supported for COBOL.
PIC S9(10) COMP	xsd:long
TYPE CHARACTER len	xsd:string maxLength=len
TYPE BINARY 8	xsd:byte
TYPE BINARY 8 UNSIGNED	xsd:unsignedByte
TYPE BINARY 16	xsd:short
TYPE BINARY 16 UNSIGNED	xsd:unsignedShort
TYPE BINARY 16,2	xsd:short for C, and for COBOL, xsd:decimal totalDigits = 4, fractionDigits = 2
TYPE BINARY 32	xsd:int
TYPE BINARY 32 UNSIGNED	xsd:unsignedInt
TYPE BINARY 64	xsd:long
TYPE BINARY 64,16	xsd:long for C, and for COBOL xsd:decimal totalDigits = 18, fractionDigits = 16
TYPE BINARY 64 UNSIGNED	xsd:unsignedLong. Not supported for COBOL.
TYPE FLOAT 32	xsd:float
TYPE FLOAT 64	xsd:double
TYPE LOGICAL 1	xsd:string maxLength = 1
TYPE LOGICAL 2	xsd:short

Table 25 `SoapAdminCL` conversion table for DDL datatype to XML datatype Conversion: *(continued)*

DDL Datatype	Schema datatype
`TYPE LOGICAL 4`	`xsd:int`
`ENUM`	`xsd:short totalDigits = 2`

NOTE: The following DDL elements are not supported by the `SoapAdminCL` tool:

- `BINARY 64 UNSIGNED` for COBOL.
- `Bit Length` for C and COBOL

D NonStop SOAP 4 APIs

This appendix lists the following NonStop SOAP 4 APIs:

- "AXIOM APIs" (page 316)
- "Client API Module" (page 343)
- "Context Hierarchy" (page 365)
- "Service Skeleton API" (page 373)
- "Utilities" (page 374)

AXIOM APIs

Attributes

The `axiom_attribute_create()` Function

The `axiom_attribute_create()` function creates an AXIOM attribute structure.

Synopsis:

```
AXIS2_EXTERN axiom_attribute_t* axiom_attribute_create
                                ( const axutil_env_t *  env,
                                  const axis2_char_t *  localname,
                                  const axis2_char_t *  value,
                                  axiom_namespace_t *  ns )
```

Parameters:

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`localname`
is an input parameter and is a pointer to the local name of the attribute. It cannot have a `NULL` value.

`value`
is an input parameter and is a pointer to the normalized attribute value. It cannot have a `NULL` value.

`ns`
is an input parameter and is a pointer to the namespace, if any, of the attribute. It is an optional parameter and can have a `NULL` value.

Return Values:

Pointer to the newly created attribute structure. If an error occurs, it returns `NULL`.

The `axiom_attribute_free()` Function

Synopsis:

```
AXIS2_EXTERN void axiom_attribute_free
                  ( struct axiom_attribute *  om_attribute,
                    const axutil_env_t *  env )
```

Description:

This function frees an `axiom_attribute` structure.

Parameters:

`om_attribute`
is a pointer to the attribute structure to be freed.

env
is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

Return Values:

The axiom_attribute_get_qname() Function

Synopsis:

```
AXIS2_EXTERN axutil_qname_t* axiom_attribute_get_qname
                              ( struct axiom_attribute *  om_attribute,
                                const axutil_env_t *  env )
```

Description:

This function creates and returns a qname structure for this attribute.

Parameters:

om_attribute
is a pointer to attribute structure for which the qname is to be returned

env
is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

Return Values:

This function returns qname for a given attribute and returns NULL on error.

The axiom_attribute_serialize() Function

Synopsis:

```
AXIS2_EXTERN int axiom_attribute_serialize
                ( struct axiom_attribute *  om_attribute,
                  const axutil_env_t *  env,
                  axiom_output_t *  om_output )
```

Description:

This function provides the serialize op.

Parameters:

om_attribute
is a pointer to the attribute structure to be serialized.

env
is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

om_output
is the AXIOM output handler to be used in serializing

Return Values:

This function returns the status of the op. It returns:

- AXIS2_SUCCESS on success
- AXIS2_FAILURE on failure

The axiom_attribute_get_localname() Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axiom_attribute_get_localname
                            ( struct axiom_attribute *  om_attribute,
                              const axutil_env_t *  env )
```

Description:

This function returns the localname of this attribute

Parameters:

`om_attribute`
is a pointer to attribute structure.

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return Values:

This function returns the `localname` and returns `NULL` on error.

The `axiom_attribute_get_value()` Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axiom_attribute_get_value
                           ( struct axiom_attribute *  om_attribute,
                             const axutil_env_t *  env )
```

Description:

This function returns value of this attribute.

Parameters:

`om_attribute`
pointer to `om_attribute` structure

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return Values:

This function returns the attribute value on success or returns `NULL` on error.

The `axiom_attribute_get_namespace()` Function

Synopsis:

```
AXIS2_EXTERN axiom_namespace_t* axiom_attribute_get_namespace
                                ( struct axiom_attribute *  om_attribute,
                                  const axutil_env_t *  env )
```

Description:

This function returns `namespace` of this attribute.

Parameters:

`om_attribute`
pointer to `om_attribute` structure

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

Return Values:

This function returns a pointer to `om_namespace` structure and returns `NULL` on error.

The `axiom_attribute_set_localname()` Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_attribute_set_localname
                            ( struct axiom_attribute *  om_attribute,
                              const axutil_env_t *  env,
                              const axis2_char_t *  localname )
```

Description:

This function sets the `localname` of the attribute.

Parameters:

`om_attribute`
pointer to `om_attribute` structure

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`localname`
is the `localname` that should be set for this attribute.

Return Values:

This function returns:

- `AXIS2_SUCCESS` on success
- `AXIS2_FAILURE` on failure

The `axiom_attribute_set_value()` Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_attribute_set_value
                            ( struct axiom_attribute *  om_attribute,
                              const axutil_env_t *  env,
                              const axis2_char_t *  value )
```

Description:

Thie function sets the attribute value.

Parameters:

`om_attribute`
pointer to `om_attribute` structure

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`value`
is the value that should be set for this attribute.

Return Values:

This function returns:

- `AXIS2_SUCCESS` on success
- `AXIS2_FAILURE` on failure

The `axiom_attribute_set_namespace()` Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_attribute_set_namespace
                            ( struct axiom_attribute *  om_attribute,
                              const axutil_env_t *  env,
                              axiom_namespace_t *  om_namespace )
```

Description:

This function sets the namespace of the attribute.

Parameters:

`om_attribute`
pointer to `om_attribute` structure

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

om_namespace
is a pointer to om_namespace structure that should be set for this attribute.

Return Values:

This function returns:

- AXIS2_SUCCESS on success
- AXIS2_FAILURE on failure

Comment

The axiom_comment_create() Function

Synopsis:

```
AXIS2_EXTERN axiom_comment_t* axiom_comment_create
                              ( const axutil_env_t *  env,
                                axiom_node_t *  parent,
                                const axis2_char_t *  value,
                                axiom_node_t **  node )
```

Description:

This function creates a comment structure.

Parameters:

env
is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

parent
is the parent node of the comment.

value
is the comment text

node
is an out parameter and returns the node corresponding to the comment created. Node type will be set to AXIOM_COMMENT. It cannot be NULL.

Return Values:

This function returns a pointer to the newly created comment structure

The axiom_comment_free() Function

Synopsis:

```
AXIS2_EXTERN void axiom_comment_free
                  ( struct axiom_comment *  om_comment,
                    const axutil_env_t *  env  )
```

Description:

This function free the axis2_comment_t structure

Parameters:

om_comment
is a pointer to axis2_commnet_t structure to be freed.

env
is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

Return Values:

This function returns the status of the op. It returns:

- AXIS2_SUCCESS on success
- AXIS2_FAILURE on failure

The axiom_comment_get_value() Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axiom_comment_get_value
                           ( struct axiom_comment *  om_comment,
                             const axutil_env_t *  env )
```

Description:

This function gets the comments data

Parameters:

om_comment
: pointer to axis2_commnet_t structure to be freed

env
: is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

Return Values:

This function returns the comment text.

The axiom_comment_set_value() Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_comment_set_value
                           ( struct axiom_comment *  om_comment,
                             const axutil_env_t *  env,
                             const axis2_char_t *  value )
```

Description:

This function sets the comment data.

Parameters:

om_comment
: pointer to axis2_commnet_t structure to be freed.

env
: is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

value
: is the comment text

Return Values:

This function returns:

- AXIS2_SUCCESS on success
- AXIS2_FAILURE on failure

Document

axiom_document_create

Synopsis:

```
AXIS2_EXTERN axiom_document_t* axiom_document_create
                               ( const axutil_env_t *  env,
                                 axiom_node_t *  root,
                                 struct axiom_stax_builder *  builder )
```

Description:

this function creates an axiom_document_t structure.

Parameters:

env
is a pointer to the environment structure. The env parameter must not be NULL.

root
pointer to document's root node. Optional, can be NULL.

builder
pointer to axiom_stax_builder.

Return Values:

This function returns a pointer to the newly created document.

axiom_document_free

Synopsis:

```
AXIS2_EXTERN void axiom_document_free
                  ( struct axiom_document *  document,
                    const axutil_env_t *  env )
```

Description:

This function frees the document structure.

Parameters:

document
pointer to axiom_document_t structure to be freed

env
is a pointer to the environment structure. The env parameter must not be NULL.

Return Values:

This function returns the status of the op. It returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axiom_document_get_root_element

Synopsis:

```
AXIS2_EXTERN axiom_node_t* axiom_document_get_root_element
                           ( struct axiom_document *  document,
                             const axutil_env_t *  env )
```

Description:

This function gets the root element of the document.

Parameters:

document
document to return the root of.

env
is a pointer to the environment structure. The env parameter must not be NULL.

Return Values:

This function returns a pointer to the root node. If no root present, this method tries to build the root. It returns NULL on error.

axiom_document_set_root_element

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_document_set_root_element
                           ( struct axiom_document *  document,
                             const axutil_env_t *  env,
                             axiom_node_t *  om_node )
```

Description:

This function sets the root element of the document. If a root node already exists, it is freed before setting to root element.

Parameters:

document
: document structure to return the root of.

env
: is a pointer to the environment structure. The `env` parameter must not be NULL.

Return Values:

This function returns the status code `AXIS2_SUCCESS` on success , else returns `AXIS2_FAILURE` on error.

axiom_document_build_all

Synopsis:

```
AXIS2_EXTERN axiom_node_t* axiom_document_build_all
                           ( struct axiom_document *  document,
                             const axutil_env_t *  env )
```

Description:

This method builds the rest of the xml input stream from current position till the root element is completed .

Parameters:

document
: pointer to `axiom_document_t` structure to be built.

env
: is a pointer to the environment structure. The `env` parameter must not be NULL.

Return Values:

None

Element

axiom_element_create

Synopsis:

```
AXIS2_EXTERN axiom_element_t* axiom_element_create
                              ( const axutil_env_t *  env,
                                axiom_node_t *  parent,
                                const axis2_char_t *  localname,
                                axiom_namespace_t *  ns,
                                axiom_node_t **  node )
```

Description:

This function creates an AXIOM element with given localname.

Parameters:

env
: is a pointer to the environment struct. The `env` parameter must not be NULL.

parent
: parent of the element node to be created. can be NULL.

localname
: localname of the elment. cannot be NULL.

ns
: namespace of the element. can be NULL. If the value of the namespace has not already been declared then the namespace structure ns will be declared and will be freed when the tree is

freed. If the value of the namespace has already been declared using another namespace structure then the namespace structure ns will be freed.

node
This is an out parameter. cannot be NULL. Returns the node corresponding to the comment created. Node type will be set to AXIOM_ELEMENT

Return Values:

This function returns a pointer to the newly created element structure

axiom_element_create_with_qname

Synopsis:

```
AXIS2_EXTERN axiom_element_t* axiom_element_create_with_qname
                              ( const axutil_env_t *  env,
                                axiom_node_t *  parent,
                                const axutil_qname_t *  qname,
                                axiom_node_t **  node )
```

Description:

This function creates an AXIOM element with given `qname`.

Parameters:

env
is a pointer to the environment struct. The `env` parameter must not be NULL.

parent
parent of the element node to be created. It can be NULL.

qname
`qname` of the elment.cannot be NULL.

node
This is an out parameter. cannot be NULL. Returns the node corresponding to the comment created. Node type will be set to `AXIOM_ELEMENT`.

Return Values:

This function returns a pointer to the newly created element structure.

axiom_element_add_attribute

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_element_add_attribute
                            ( axiom_element_t *  om_element,
                              const axutil_env_t *  env,
                              axiom_attribute_t *  attribute,
                              axiom_node_t *  node )
```

Description:

This function adds an attribute to current element. The current element takes responsibility of the assigned attribute.

Parameters:

om_element
element to which the attribute is to be added. It cannot be NULL.

env
is a pointer to the environment struct. The `env` parameter must not be NULL.

attribute
attribute to be added.

node
`axiom_node_t` node that `om_element` is contained in

Return Values:

This function returns the status of the op. AXIS2_SUCCESS on success else AXIS2_FAILURE.

axiom_element_get_attribute

Synopsis:

```
AXIS2_EXTERN axiom_attribute_t* axiom_element_get_attribute
                                ( axiom_element_t *  om_element,
                                  const axutil_env_t *  env,
                                  axutil_qname_t *  qname )
```

Description:

This function gets (finds) the attribute with the given `qname`.

Parameters:

element

element whose attribute is to be found.

env

is a pointer to the environment struct. The `env` parameter must not be NULL.

Return Values:

This function returns a pointer to the attribute with given qname if found, else returns NULL. On error, returns NULL and sets the error code in environment's error structure.

axiom_element_get_attribute_value

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axiom_element_get_attribute_value
                           ( axiom_element_t *  om_element,
                             const axutil_env_t *  env,
                             axutil_qname_t *  qname )
```

Description:

This function gets (finds) the attribute value with the given `qname`.

Parameters:

element

element whose attribute value is to be found.

env

is a pointer to the environment struct. The `env` parameter must not be NULL.

qname

`qname` of the attribute to be found. should not be NULL.

Return Values:

This function returns the attribute value with given qname if found, else returns NULL. On error, returns NULL and sets the error code in environment's error structure.

axiom_element_free

Synopsis:

```
AXIS2_EXTERN void axiom_element_free
                 ( axiom_element_t *  element,
                   const axutil_env_t *  env )
```

Description:

This function frees the given element.

Parameters:

element

AXIOM element to be freed.

env
is a pointer to the environment struct. The env parameter must not be NULL.

qname
qname of the attribute to be found. should not be NULL.

Return Values:

This function returns the status of the operation. It returns AXIS2_SUCCESS on success , else returns AXIS2_FAILURE on error.

axiom_element_get_localname

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axiom_element_get_localname
                           ( axiom_element_t *  om_element,
                             const axutil_env_t *  env )
```

Description:

This function returns the local name of this element.

Parameters:

om_element
pointer to om_element.

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns the local name of element, returns NULL on error.

axiom_element_set_localname

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_element_set_localname
                            ( axiom_element_t *  om_element,
                              const axutil_env_t *  env,
                              const axis2_char_t *  localname )
```

Description:

This function sets the local name of this element.

Parameters:

om_element
pointer to om_element.

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returs the status code of operation. It returns AXIS2_SUCCESS on success, AXIS2_FAILURE on error.

axiom_element_get_namespace

Synopsis:

```
AXIS2_EXTERN axiom_namespace_t* axiom_element_get_namespace
                                ( axiom_element_t *  om_element,
                                  const axutil_env_t *  env,
                                  axiom_node_t *  ele_node )
```

Description:

This function gets the namespace of om_element.

Parameters:

om_element
pointer to om_element.

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This pointer returns a pointer to axiom_namespace_t structure NULL if there is no namespace associated with the element, NULL on error with error code set to environment's error.

axiom_element_set_namespace

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_element_set_namespace
                            ( axiom_element_t *  om_element,
                              const axutil_env_t *  env,
                              axiom_namespace_t *  ns,
                              axiom_node_t *  node  )
```

Description:

This function sets the namespace of the element.

Parameters:

om_element
pointer to om_element.

env
is a pointer to the environment struct. The env parameter must not be NULL.

ns
pointer to namespace. If the value of the namespace has not already been declared then the namespace structure ns will be declared and will be freed when the tree is freed.

Return Values:

This function returns the status code of the op, with error code set to environment's error.

axiom_element_get_all_attributes

Synopsis:

```
AXIS2_EXTERN axutil_hash_t* axiom_element_get_all_attributes
                             ( axiom_element_t *  om_element,
                               const axutil_env_t *  env )
```

Description:

This function gets the attribute list of the element.

Parameters:

om_element
pointer to om_element.

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns axutil_hash pointer to attributes hash. This hash table is read only.

axiom_element_get_children

Synopsis:

```
AXIS2_EXTERN axiom_children_iterator_t* axiom_element_get_children
                                        ( axiom_element_t *  om_element,
                                          const axutil_env_t *  env,
                                          axiom_node_t *  element_node )
```

Description:

This function returns a list of children iterator returned iterator is freed when `om_element` structure is freed iterators reset function must be called by user

Parameters:

om_element
pointer to `om_element`.

env
is a pointer to the environment struct. The `env` parameter must not be NULL.

element_node
pointer to this element node.

Return Values:

None

axiom_element_get_children_with_qname

Synopsis:

```
AXIS2_EXTERN axiom_children_qname_iterator_t* axiom_element_get_children_with_qname
                                              ( axiom_element_t *  om_element,
                                                const axutil_env_t *  env,
                                                axutil_qname_t *  element_qname,
                                                axiom_node_t *  element_node )
```

Description:

This function returns a list of children iterator with qname returned iterator is freed when AXIOM element structure is freed.

Parameters:

om_element
pointer to om_element.

env
is a pointer to the environment struct. The `env` parameter must not be NULL.

element_node
pointer to this element node.

Return Values:

This function returns the children qname iterator struct.

axiom_element_remove_attribute

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_element_remove_attribute
                            ( axiom_element_t *  om_element,
                              const axutil_env_t *  env,
                              axiom_attribute_t *  om_attribute )
```

Description:

This function removes an attribute from the element attribute list. You must free this attribute, element free function does not free attributes that are not in its attribute list.

Parameters:

om_element
pointer to `om_element`.

env
is a pointer to the environment struct. The `env` parameter must not be NULL.

om_attribute
attribute to be removed.

Return Values:

This function returns `AXIS2_SUCCESS` if the attribute was found and removed, else returns `AXIS2_FAILURE`.

axiom_element_set_text

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_element_set_text
                            ( axiom_element_t *  om_element,
                              const axutil_env_t *  env,
                              const axis2_char_t *  text,
                              axiom_node_t *  element_node )
```

Description:

This function sets the text of the given element.

△ **CAUTION:** This method will wipe out all the text elements (and hence any mixed content) before setting the text.

Parameters:

om_element
pointer to `om_element`.

env
is a pointer to the environment struct. The `env` parameter must not be NULL.

text
text to set.

element_node
node of element.

Return Values:

This function returns `AXIS2_SUCCESS` if attribute was found and removed, else returns `AXIS2_FAILURE`.

axiom_element_get_text

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axiom_element_get_text
                          ( axiom_element_t *  om_element,
                            const axutil_env_t *  env,
                            axiom_node_t *  element_node )
```

Description:

This function selects all the text children and concatenates them to a single string. The string returned by this method call will be released by AXIOM when this method is called again. So it is recomended to have a copy of the return value if this method is going to be called more than once and the return values of the earlier calls are important.

Parameters:

om_element
pointer to `om_element`.

env
is a pointer to the environment struct. The `env` parameter must not be NULL.

element
node , the container node of this AXIOM element

Return Values:

This function returns the concatenated text of all text children's text values return null if no text children is available or on error.

axiom_element_to_string

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axiom_element_to_string
                           ( axiom_element_t *  om_element,
                             const axutil_env_t *  env,
                             axiom_node_t *  element_node )
```

Description:

This function returns the serilized text of this element and its children

Parameters:

om_element
pointer to om_element.

env
is a pointer to the environment struct. The `env` parameter must not be NULL.

element_node
the container node this element is contained on.

Return Values:

This function returns a character array of xml , returns NULL on error.

axiom_element_get_child_elements

Synopsis:

```
AXIS2_EXTERN axiom_child_element_iterator_t* axiom_element_get_child_elements
                                             ( axiom_element_t *  om_element,
                                               const axutil_env_t *  env,
                                               axiom_node_t *  element_node )
```

Description:

This function returns an iterator with child elements of type AXIOM_ELEMENT iterator is freed when om_element node is freed

Parameters:

om_element
pointer to om_element

env
is a pointer to the environment struct. The `env` parameter must not be NULL.

element_node

Return Values:

This function returns `axiom_child_element_iterator_t` if successful and returns NULL on error.

axiom_element_extract_attributes

Synopsis:

```
AXIS2_EXTERN axutil_hash_t* axiom_element_extract_attributes
                            ( axiom_element_t *  om_element,
                              const axutil_env_t *  env,
                              axiom_node_t *  ele_node )
```

Description:

This function extracts attributes and returns a clones hash table of attributes. If the attributes are associated with a namespace, it is also cloned.

Parameters:

om_element
pointer to om_element.

env
is a pointer to the environment struct. The `env` parameter must not be NULL.

om_node
pointer to this element node.

Return Values:

axiom_element_get_attribute_value_by_name

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axiom_element_get_attribute_value_by_name
                           ( axiom_element_t *  om_ele,
                             const axutil_env_t *  env,
                             axis2_char_t *  attr_name  )
```

Description:

This function returns the attribute value as a string for the given element.

Parameters:

om_element
pointer to om_element.

env
is a pointer to the environment struct. The `env` parameter must not be NULL.

attr_name
the attribute name.

Return Values:

This function returns the attribute value as a string.

axiom_element_get_localname_str

Synopsis:

```
AXIS2_EXTERN axutil_string_t* axiom_element_get_localname_str
                              ( axiom_element_t *  om_element,
                                const axutil_env_t *  env )
```

Description:

This function returns the local name of the element.

Parameters:

om_element
pointer to `om_element`.

env
is a pointer to the environment struct. The `env` parameter must not be NULL.

om_node
pointer to this element node.

Return Values:

This function returns the local name of the element.

axiom_element_set_localname_str

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_element_set_localname_str
                            ( axiom_element_t *  om_element,
                              const axutil_env_t *  env,
                              axutil_string_t *  localname )
```

Description:

This function sets the local name of the element.

Parameters:

om_element
pointer to om_element

env
is a pointer to the environment struct. The env parameter must not be NULL.

localname
the local name of the element.

Return Values:

Namespace

axiom_namespace_create

Synopsis:

```
AXIS2_EXTERN axiom_namespace_t* axiom_namespace_create
                                ( const axutil_env_t *  env,
                                  const axis2_char_t *  uri,
                                  const axis2_char_t *  prefix )
```

Description:

This function creates a namespace structure

Parameters:

uri
namespace URI

prefix
namespace prefix

Return Values:

This function returns a pointer to newly created namespace struct.

axiom_namespace_free

Synopsis:

```
AXIS2_EXTERN void axiom_namespace_free
                  ( struct axiom_namespace *  om_namespace,
                    const axutil_env_t *  env )
```

Description:

This function frees given AXIOM namespcae.

Parameters:

om_namespace
namespace to be freed.

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns the status of the operation. AXIS2_SUCCESS on success else AXIS2_FAILURE.

axiom_namespace_equals

Synopsis:

```
AXIS2_EXTERN axis2_bool_t axiom_namespace_equals
                          ( struct axiom_namespace *  om_namespace,
                            const axutil_env_t *  env,
                            struct axiom_namespace *  om_namespace1 )
```

Description:

This function compares two namepsaces.

Parameters:

om_namespace
: first namespace to be compared

env
: is a pointer to the environment struct. The env parameter must not be NULL.

om_namespace1
: second namespace to be compared

Return Values:

This funtion returns AXIS2_TRUE if the two namespaces are equal, AXIS2_FALSE otherwise.

axiom_namespace_serialize

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_namespace_serialize
                            ( struct axiom_namespace *  om_namespace,
                              const axutil_env_t *  env,
                              axiom_output_t *  om_output )
```

Description:

This function serializes given namespace.

Parameters:

om_namespace
: namespace to be compared

env
: is a pointer to the environment struct. The env parameter must not be NULL.

om_output
: om_output

Return Values:

This function returns the status of the operation. It returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axiom_namespace_get_uri

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axiom_namespace_get_uri
                           ( struct axiom_namespace *  om_namespace,
                             const axutil_env_t *  env )
```

Description:

Parameters:

om_namespace
: pointer to om_namespace structure

env
: is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns namespace uri or NULL on error.

axiom_namespace_get_prefix

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axiom_namespace_get_prefix
                           ( struct axiom_namespace *  om_namespace,
                             const axutil_env_t *  env  )
```

Description:

Parameters:

om_namespace
pointer to om_namespace structure

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns the prefix or NULL on error.

axiom_namespace_to_string

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axiom_namespace_to_string
                           ( struct axiom_namespace *  om_namespace,
                             const axutil_env_t *  env )
```

Description:

to string , returns the string by combining namespace_uri, and prefix seperated by a '|' character

Parameters:

om_namespace

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns pointer to string , This is a property of namespace, should not be freed by user.

axiom_namespace_create_str

Synopsis:

```
AXIS2_EXTERN axiom_namespace_t* axiom_namespace_create_str
                                ( const axutil_env_t *  env,
                                  axutil_string_t *  uri,
                                  axutil_string_t *  prefix )
```

Description:

This funcion creates an AXIOM namespace from a URI and a Prefix.

Parameters:

om_namespace
pointer to the AXIOM namespace structure

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns the created AXIOM namespace.

axiom_namespace_set_uri_str

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_namespace_set_uri_str
                            ( axiom_namespace_t *  om_namespace,
```

```
                                    const axutil_env_t *  env,
                                    axutil_string_t *  uri )
```

Description:

This function sets the uri string.

Parameters:

om_namespace
pointer to the AXIOM namespace structure

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axiom_namespace_get_uri_str

Synopsis:

```
AXIS2_EXTERN axutil_string_t* axiom_namespace_get_uri_str
                                ( axiom_namespace_t *  om_namespace,
                                  const axutil_env_t *  env )
```

Description:

This function gets the uri as a string.

Parameters:

om_namespace
pointer to the AXIOM namespace structure

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns the uri as a string.

axiom_namespace_get_prefix_str

Synopsis:

```
AXIS2_EXTERN axutil_string_t* axiom_namespace_get_prefix_str
                                ( axiom_namespace_t *  om_namespace,
                                  const axutil_env_t *  env )
```

Description:

This function gets the prefix as a string.

Parameters:

om_namespace
pointer to the AXIOM namespace structure

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns the prefix as a string.

Node

axiom_node_create

Synopsis:

```
AXIS2_EXTERN axiom_node_t* axiom_node_create
                            ( const axutil_env_t *  env   )
```

Description:

This function creates a node struct.

Parameters:

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns a pointer to newly created node struct. NULL on error.

axiom_node_free_tree

Synopsis:

```
AXIS2_EXTERN void axiom_node_free_tree
                  ( axiom_node_t *  om_node,
                    const axutil_env_t *  env )
```

Description:

This function frees an AXIOM node and all of its children. Please note that the attached data_element will also be freed along with the node. If the node is still attached to a parent, it will be detached first, then freed.

Parameters:

om_node
node to be freed.

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns the status of the op. It returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_node_add_child
                            ( axiom_node_t *  om_node,
                              const axutil_env_t *  env,
                              axiom_node_t *  child )
```

Description:

This function adds a given node as child to parent. child should not have a parent if child has a parent it will be detached from existing parent.

Parameters:

om_node
parent node. cannot be NULL.

env
is a pointer to the environment struct. The env parameter must not be NULL.

child
child node.

Return Values:

This function returns the status of the operation. It returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axiom_node_insert_sibling_after

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_node_insert_sibling_after
                            ( axiom_node_t *  om_node,
```

```
                                    const axutil_env_t *  env,
                                    axiom_node_t *  node_to_insert )
```

Description:

This function inserts a sibling node after the given node.

Parameters:

om_node
: parent node. cannot be NULL.

env
: is a pointer to the environment struct. The `env` parameter must not be NULL.

node_to_insert
: the node to be inserted. Cannot be NULL.

Return Values:

This function returns the status of the op. It returns `AXIS2_SUCCESS` on success, else returns `AXIS2_FAILURE`.

axiom_node_insert_sibling_before

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_node_insert_sibling_before
                            ( axiom_node_t *  om_node,
                              const axutil_env_t *  env,
                              axiom_node_t *  node_to_insert )
```

Description:

This function inserts a sibling node before the given node.

Parameters:

om_node
: parent node. cannot be NULL.

env
: is a pointer to the environment struct. The `env` parameter must not be NULL.

node_to_insert
: the node to be inserted. Cannot be NULL.

Return Values:

This function returns the status of the op. It returns `AXIS2_SUCCESS` on success, else returns `AXIS2_FAILURE`.

axiom_node_serialize

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_node_serialize
                            ( axiom_node_t *  om_node,
                              const axutil_env_t *  env,
                              struct axiom_output *  om_output )
```

Description:

This function serializes the given node. This operation makes the node go through its children and serialize them in order.

Parameters:

om_node
: parent node. cannot be NULL.

env
: is a pointer to the environment struct. The `env` parameter must not be NULL.

om_output
AXIOM output handler to be used in serializing.

Return Values:

This function returns the status of the operation. This function returns AXIS2_SUCCESS on success, else AXIS2_FAILURE.

axiom_node_get_parent

Synopsis:

```
AXIS2_EXTERN axiom_node_t* axiom_node_get_parent
                           ( axiom_node_t *  om_node,
                             const axutil_env_t *  env  )
```

Description:

This function gets parent of om_node node.

Parameters:

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns a pointer to parent node of om_node, returns NULL if no parent exists or when an error occurs.

axiom_node_get_first_child

Synopsis:

```
AXIS2_EXTERN axiom_node_t* axiom_node_get_first_child
                           ( axiom_node_t *  om_node,
                             const axutil_env_t *  env   )
```

Description:

This function gets the first child of om_node.

Parameters:

om_node
node

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns a pointer to first child node , NULL is returned on error with error code set in environments error.

axiom_node_get_first_element

Synopsis:

```
AXIS2_EXTERN axiom_node_t* axiom_node_get_first_element
                           ( axiom_node_t *  om_node,
                             const axutil_env_t *  env  )
```

Description:

This function gets the first AXIOM_ELEMENT in om_node.

Parameters:

om_node
node

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns a pointer to first child AXIOM element, NULL is returned on error with error code set in environments error.

axiom_node_get_last_child

Synopsis:

```
AXIS2_EXTERN axiom_node_t* axiom_node_get_last_child
                           ( axiom_node_t *  om_node,
                             const axutil_env_t *  env   )
```

Description:

This function gets the last child of the node.

Parameters:

om_node
: node

env
: is a pointer to the environment struct. The `env` parameter must not be NULL.

Return Values:

This function returns a pointer to last child of this node , return NULL on error.

axiom_node_get_previous_sibling

Synopsis:

```
AXIS2_EXTERN axiom_node_t* axiom_node_get_previous_sibling
                           ( axiom_node_t *  om_node,
                             const axutil_env_t *  env  )
```

Description:

This function gets the previous sibling.

Parameters:

om_node
: `om_node` structure

env
: is a pointer to the environment struct. The `env` parameter must not be NULL.

Return Values:

This function returns a pointer to previous sibling , NULL if a previous sibling does not exits (happens when this node is the first child of a node).

axiom_node_get_next_sibling

Synopsis:

```
AXIS2_EXTERN axiom_node_t* axiom_node_get_next_sibling
                           ( axiom_node_t *  om_node,
                             const axutil_env_t *  env  )
```

Description:

This function gets the next sibling.

Parameters:

om_node
: om_node structure

env
: is a pointer to the environment struct. The `env` parameter must not be NULL.

Return Values:

This function returns the next sibling of this node.

axiom_node_get_node_type

Synopsis:

```
AXIS2_EXTERN axiom_types_t axiom_node_get_node_type
                           ( axiom_node_t *  om_node,
                             const axutil_env_t *  env  )
```

Description:

This function gets the node type of this element Node type can be one of AXIOM_ELEMENT, AXIOM_COMMENT, AXIOM_TEXT, AXIOM_DOCTYPE, AXIOM_PROCESSING_INSTRUCTION.

Parameters:

om_node
node of which node type is required

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns the node type.

axiom_node_get_data_element

Synopsis:

```
AXIS2_EXTERN void* axiom_node_get_data_element
                   ( axiom_node_t *  om_node,
                     const axutil_env_t *  env  )
```

Description:

This function gets the structure contained in the node IF the node is on type AXIOM_ELEMENT , this method returns a pointer to axiom_element_t structure contained.

Parameters:

om_node
node of which node type is required

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns a pointer to structure contained in the node returns NULL if no structure is contained.

Synopsis:

```
AXIS2_EXTERN struct axiom_document* axiom_node_get_document
                                    ( axiom_node_t *  om_node,
                                      const axutil_env_t *  env  )
```

Description:

This function returns the associated document, only valid if built using builder and for a node of type AXIOM_ELEMENT returns NULL if no document is available.

Parameters:

om_node
pointer to the AXIOM node structure

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns the AXIOM document of the node.

axiom_node_to_string

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axiom_node_to_string
                           ( axiom_node_t *  om_node,
                             const axutil_env_t *  env  )
```

Description:

This function

Parameters:

om_node
: pointer to the AXIOM node structure

env
: is a pointer to the environment struct. The `env` parameter must not be NULL.

Return Values:

This function returns the string representation of the node

Text

axiom_text_create

Synopsis:

```
AXIS2_EXTERN axiom_text_t* axiom_text_create
                           ( const axutil_env_t *  env,
                             axiom_node_t *  parent,
                             const axis2_char_t *  value,
                             axiom_node_t **  node   )
```

Description:

This function creates a new text struct.

Parameters:

env
: is a pointer to the environment struct. The `env` parameter must not be NULL.

parent
: parent of the new node. Optinal, can be NULL. The parent element must be of type `AXIOM_ELEMENT`.

value
: Text value. Optinal, can be NULL.

comment_node
: This is an out parameter. cannot be NULL. Returns the node corresponding to the text structure created. Node type will be set to `AXIOM_TEXT`.

Return Values:

This function returns a pointer to newly created text struct.

axiom_text_free

Synopsis:

```
AXIS2_EXTERN void axiom_text_free
                  ( struct axiom_text *  om_text,
                    const axutil_env_t *  env   )
```

Description:

This function frees an `axiom_text` struct.

Parameters:

env
is a pointer to the environment struct. The env parameter must not be NULL.

om_text
pointer to AXIOM text structure to be freed.

Return Values:

This function returns the status of the op. It returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE on error.

axiom_text_serialize

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_text_serialize
                            ( struct axiom_text *  om_text,
                              const axutil_env_t *  env,
                              axiom_output_t *  om_output   )
```

Description:

Serialize operation

Parameters:

env
is a pointer to the environment struct. The env parameter must not be NULL.

om_text
pointer to AXIOM text structure to be freed.

om_output
AXIOM output handler to be used in serializing.

Return Values:

This function returns the status of the operation. It returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE on error.

axiom_text_set_value

Synopsis:

```
AXIS2_EXTERN axis2_status_t axiom_text_set_value
                            ( struct axiom_text *  om_text,
                              const axutil_env_t *  env,
                              const axis2_char_t *  value   )
```

Description:

This function sets the text value.

Parameters:

om_text
om_text structure

env
is a pointer to the environment struct. The env parameter must not be NULL.

value
text

Return Values:

The function returns the status of the operation. It returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE on error.

axiom_text_get_value

Synopsis:

```
AXIS2_EXTERN const axis2_char_t* axiom_text_get_value
                                 ( struct axiom_text *  om_text,
                                   const axutil_env_t *  env   )
```

Description:

This function gets text value.

Parameters:

om_text

om_text structure

env

is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns the text value , NULL is returned if there is no text value.

axiom_text_get_text

Synopsis:

```
AXIS2_EXTERN const axis2_char_t* axiom_text_get_text
                                 ( axiom_text_t *  om_text,
                                   const axutil_env_t *  env  )
```

Description:

This function gets text value from the text node even when MTOM optimized.

Parameters:

om_text

om_text structure

env

is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns the text value base64 encoded text when MTOM optimized, NULL is returned if there is no text value.

Client API Module

client service

The client service APIs are as follows:

axis2_svc_client_get_svc

Synopsis:

```
AXIS2_EXTERN axis2_svc_t* axis2_svc_client_get_svc
                          ( const axis2_svc_client_t *  svc_client,
                            const axutil_env_t *  env
                          )
```

Description:

This function returns the axis2_svc_t this is a client for. This is useful when the service is created anonymously or from a WSDL.

Parameters:

svc_client

pointer to service client structure

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns a pointer to service context struct. The service client owns the returned pointer.

axis2_svc_client_set_options

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_svc_client_set_options
                            ( axis2_svc_client_t *  svc_client,
                              const axutil_env_t *  env,
                              const axis2_options_t *  options
                             )
```

Description:

This function sets the options to be used by service client.

Parameters:

svc_client
pointer to service client struct.

env
is a pointer to the environment struct. The env parameter must not be NULL.

options
pointer to options structure to be set.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_svc_client_get_options

Synopsis:

```
AXIS2_EXTERN const axis2_options_t* axis2_svc_client_get_options
                                    ( const axis2_svc_client_t *  svc_client,
                                      const axutil_env_t *  env
                                    )
```

Description:

This function gets options used by service client.

Parameters:

svc_client
pointer to service client struct.

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns a pointer to the options structure if options set, else returns NULL. It returns a reference and not a cloned copy.

axis2_svc_client_engage_module

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_svc_client_engage_module
                            ( axis2_svc_client_t *  svc_client,
                              const axutil_env_t *  env,
                              const axis2_char_t *  module_name
                            )
```

Description:

This function engages the named module. The engaged modules extend the message processing when consuming services. Modules help to apply QoS norms in messaging. After a module is engaged to a service client, the `axis2_engine` invokes the module for all the interactions between the client and the service.

Parameters:

`svc_client`
pointer to service client struct.

`env`
is a pointer to the environment struct. The `env` parameter must not be NULL.

`module_name`
name of the module to be engaged.

Return Values:

This function returns `AXIS2_SUCCESS` on success, else returns `AXIS2_FAILURE`.

axis2_svc_client_disengage_module

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_svc_client_disengage_module
                            ( axis2_svc_client_t *  svc_client,
                              const axutil_env_t *  env,
                              const axis2_char_t *  module_name )
```

Description:

This function disengages the specified module. Disengaging a module on a service client ensures that the `axis2_engine` does not invoke the specified module when sending and receiving messages.

Parameters:

`svc_client`
pointer to service client struct.

`env`
is a pointer to the environment struct. The `env` parameter must not be NULL.

`module_name`
name of the module to be engaged.

Return Values:

This function returns `AXIS2_SUCCESS` on success, else returns `AXIS2_FAILURE`.

axis2_svc_client_add_header

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_svc_client_add_header  \
                            ( axis2_svc_client_t *  svc_client,
                              const axutil_env_t *  env,
                              axiom_node_t *  header )
```

Description:

This function adds an XML element as a header to be sent to the server side. This enables users to go beyond the usual XML-in/XML-out pattern, and send custom SOAP headers. After being added, the service client owns the header and cleans up when the service client is freed.

Parameters:

`svc_client`
pointer to service client struct.

`env`
is a pointer to the environment struct. The `env` parameter must not be NULL.

header
AXIOM node representing the SOAP header in XML.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_svc_client_remove_all_headers

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_svc_client_remove_all_headers
                            ( axis2_svc_client_t *  svc_client,
                              const axutil_env_t *  env )
```

Description:

This function removes all the headers added to service client.

Parameters:

svc_client
pointer to service client struct.

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_svc_client_fire_and_forget_with_op_qname

Synopsis:

```
AXIS2_EXTERN void axis2_svc_client_fire_and_forget_with_op_qname
                  ( axis2_svc_client_t *  svc_client,
                    const axutil_env_t *  env,
                    const axutil_qname_t *  op_qname,
                    const axiom_node_t *  payload )
```

Description:

This function sends a message and forget about it. This method is used to interact with a service operation whose MEP is In-Only. That is, there is no opportunity to get an error from the service using this method; you may still get client-side errors, such as host unknown.

Parameters:

svc_client
pointer to service client structure

env
is a pointer to the environment struct. The env parameter must not be NULL.

op_qname
operation qname. NULL is equivalent to an operation name of "__OPERATION_OUT_ONLY__"

payload
pointer to AXIOM node representing the XML payload to be sent. The caller has control over the payload until the service client frees it.

Return Values:

None

axis2_svc_client_fire_and_forget

Synopsis:

```
AXIS2_EXTERN void axis2_svc_client_fire_and_forget
                 ( axis2_svc_client_t *  svc_client,
                   const axutil_env_t *  env,
                   const axiom_node_t *  payload )
```

Description:

This function sends a message and forget about it. This method is used to interact with a service operation whose MEP is In-Only. That is, there is no opportunity to get an error from the service via this method; one may still get client-side errors, such as host unknown etc.

Parameters:

svc_client
pointer to service client struct.

env
is a pointer to the environment struct. The `env` parameter must not be NULL.

payload
pointer to AXIOM node representing the XML payload to be sent. The caller has control over the payload until the service client frees it.

Return Values:

None

axis2_svc_client_send_receive

Synopsis:

```
AXIS2_EXTERN axiom_node_t* axis2_svc_client_send_receive
                          ( axis2_svc_client_t *  svc_client,
                            const axutil_env_t *  env,
                            const axiom_node_t *  payload )
```

Description:

This function sends an XML request and receives XML response. This method is used to interact with a service operation whose MEP is IN-OUT.

Parameters:

svc_client
pointer to service client structure

env
pointer to the environment structure. The `env` parameter must not be NULL.

payload
pointer to AXIOM node representing the XML payload to be sent. The caller has control over the payload until the service client frees it.

Return Values:

This function returns a pointer to AXIOM node representing the XML response. The caller owns the returned node.

axis2_svc_client_send_receive_with_op_qname

Synopsis:

```
AXIS2_EXTERN axiom_node_t* axis2_svc_client_send_receive_with_op_qname
                          ( axis2_svc_client_t *  svc_client,
                            const axutil_env_t *  env,
                            const axutil_qname_t *  op_qname,
                            const axiom_node_t *  payload )
```

Description:

This function sends an XML request and receives XML response. This method is used to interact with a service operation whose MEP is IN-OUT.

Parameters:

svc_client
pointer to service client structure

env
pointer to the environment structure

op_qname
operation qname. NULL is equivalent to an operation name of "__OPERATION_OUT_IN__".

payload
pointer to OM node representing the XML payload to be sent. The caller has control over the payload until the service client frees it.

Return Values:

This function returns a pointer to OM node representing the XML response. The caller owns the returned node.

axis2_svc_client_send_receive_non_blocking_with_op_qname

Synopsis:

```
AXIS2_EXTERN void axis2_svc_client_send_receive_non_blocking_with_op_qname
                ( axis2_svc_client_t *  svc_client,
                  const axutil_env_t *  env,
                  const axutil_qname_t *  op_qname,
                  const axiom_node_t *  payload,
                  axis2_callback_t *  callback )
```

Description:

This function sends XML request and receives XML response, but does not block for response. This method is used to interact in non-blocking mode with a service operation whose MEP is IN-OUT.

Parameters:

svc_client
pointer to service client structure

env
is a pointer to the environment struct. The `env` parameter must not be NULL.

op_qname
operation qname. NULL is equivalent to an operation name of "__OPERATION_OUT_IN__"

payload
pointer to AXIOM node representing the XML payload to be sent. The caller has control over the payload until the service client frees it.

Return Values:

None

axis2_svc_client_get_svc_ctx

Synopsis:

```
AXIS2_EXTERN axis2_svc_ctx_t* axis2_svc_client_get_svc_ctx
                             ( const axis2_svc_client_t *  svc_client,
                               const axutil_env_t *  env )
```

Description:

This function gets the service context.

Parameters:

svc_client
pointer to service client struct.

env
is a pointer to the environment structure. The `env` parameter must not be NULL.

Return Values:

This function returns a pointer to service context struct. The service client owns the returned pointer.

axis2_svc_client_free

Synopsis:

```
AXIS2_EXTERN void axis2_svc_client_free
                 ( axis2_svc_client_t *  svc_client,
                   const axutil_env_t *  env )
```

Description:

This function frees the service client.

Parameters:

svc_client
pointer to service client struct.

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_svc_client_create

Synopsis:

```
AXIS2_EXTERN axis2_svc_client_t* axis2_svc_client_create
                                 ( const axutil_env_t *  env,
                                   const axis2_char_t *  client_home )
```

Description:

This function creates a service client struct.

Parameters:

env
is a pointer to the environment struct. The env parameter must not be NULL.

client_home
name of the directory that contains the Axis2/C repository

Return Values:

This function returns a pointer to newly created service client struct, or returns NULL on error with the error code set in the environment's error.

Options

axis2_options_get_action

Synopsis:

```
AXIS2_EXTERN const axis2_char_t* axis2_options_get_action
                                 ( const axis2_options_t *  options,
                                   const axutil_env_t *  env )
```

Description:

This function gets Web Services Addressing (WSA) action.

Parameters:

options
pointer to options struct.

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns WSA action string if set, else returns NULL.

axis2_options_get_fault_to

Synopsis:

```
AXIS2_EXTERN axis2_endpoint_ref_t* axis2_options_get_fault_to
                                    ( const axis2_options_t *  options,
                                      const axutil_env_t *  env )
```

Description:

This function gets WSA fault to address.

Parameters:

options

pointer to options struct.

env

is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns a pointer to endpoint reference structure representing fault to address if set, else returns NULL.

axis2_options_get_from

Synopsis:

```
AXIS2_EXTERN axis2_endpoint_ref_t* axis2_options_get_from
                                    ( const axis2_options_t *  options,
                                      const axutil_env_t *  env )
```

Description:

This function gets WSA from address.

Parameters:

options

pointer to options struct.

env

is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This funtion returns a pointer to endpoint reference structure representing the from address if set, else returns NULL.

axis2_options_get_transport_receiver

Synopsis:

```
AXIS2_EXTERN axis2_transport_receiver_t* axis2_options_get_transport_receiver
                                          ( const axis2_options_t *  options,
                                            const axutil_env_t *  env )
```

Description:

This function gets the transport receiver.

Parameters:

options

pointer to options struct.

env

is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns pointer to transport receiver structure if set, else returns NULL.

axis2_options_get_transport_in

Synopsis:

```
AXIS2_EXTERN axis2_transport_in_desc_t* axis2_options_get_transport_in
                                        ( const axis2_options_t *  options,
                                          const axutil_env_t *  env )
```

Description:

This function gets transport `in`.

Parameters:

options

pointer to options struct.

env

is a pointer to the environment struct. The `env` parameter must not be NULL.

Return Values:

This function returns a pointer to transport `in` structure if set, else NULL.

axis2_options_get_transport_in_protocol

Synopsis:

```
AXIS2_EXTERN AXIS2_TRANSPORT_ENUMS axis2_options_get_transport_in_protocol
                                   ( const axis2_options_t *  options,
                                     const axutil_env_t *  env )
```

Description:

This function gets transport `in` protocol.

Parameters:

options

pointer to options struct.

env

is a pointer to the environment struct. The `env` parameter must not be NULL.

Return Values:

This function returns pointer to transport in protocol string if set, else returns NULL.

axis2_options_get_message_id

Synopsis:

```
AXIS2_EXTERN const axis2_char_t* axis2_options_get_message_id
                                 ( const axis2_options_t *  options_t,
                                   const axutil_env_t *  env )
```

Description:

This function gets the message `ID`.

Parameters:

options

pointer to options struct.

env

is a pointer to the environment struct. The `env` parameter must not be NULL.

Return Values:

This function returns a pointer to the message ID string if set, else returns NULL.

axis2_options_get_properties

Synopsis:

```
AXIS2_EXTERN axutil_hash_t* axis2_options_get_properties
                            ( const axis2_options_t *  options,
                              const axutil_env_t *  env  )
```

Description:

This function gets the properties hash map.

Parameters:

options
pointer to options struct.

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns a pointer to the properties hash map if set, else returns NULL.

axis2_options_get_property

Synopsis:

```
AXIS2_EXTERN void* axis2_options_get_property
                  ( const axis2_options_t *  options,
                    const axutil_env_t *  env,
                    const axis2_char_t *  key )
```

Description:

This function gets a property corresponding to the given *key*.

Parameters:

options
pointer to options struct.

env
is a pointer to the environment struct. The env parameter must not be NULL.

key
key of the property to be returned.

Return Values:

This function returns the value corresponding to the given key.

axis2_options_get_property

Synopsis:

```
AXIS2_EXTERN axis2_relates_to_t* axis2_options_get_property
                                 ( const axis2_options_t *  options,
                                   const axutil_env_t *  env  )
```

Description:

This function gets relates_to information.

Parameters:

options
pointer to options struct.

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns pointer to relates_to structure if set, else returns NULL.

axis2_options_get_reply_t

Synopsis:

```
AXIS2_EXTERN axis2_endpoint_ref_t* axis2_options_get_reply_to
                                    ( const axis2_options_t *  options,
                                      const axutil_env_t *  env )
```

Description:

This function gets the WSA `reply_to` address.

Parameters:

options

pointer to options structure

env

is a pointer to the environment struct. The `env` parameter must not be NULL.

Return Values:

This function returns a pointer to the endpoint reference structure representing reply to address if set, else returns NULL.

axis2_options_get_transport_out

Synopsis:

```
AXIS2_EXTERN axis2_transport_out_desc_t* axis2_options_get_transport_out
                                         ( const axis2_options_t *  options,
                                           const axutil_env_t *  env )
```

Description:

This function gets transport `out`.

Parameters:

options

pointer to options struct.

env

is a pointer to the environment struct. The `env` parameter must not be NULL.

Return Values:

This function returns pointer to transport `out` structure if set, else NULL.

axis2_options_get_sender_transport_protocol

Synopsis:

```
AXIS2_EXTERN AXIS2_TRANSPORT_ENUMS axis2_options_get_sender_transport_protocol
                                   ( const axis2_options_t *  options,
                                     const axutil_env_t *  env )
```

Description:

This function gets transport `out` protocol.

Parameters:

options

pointer to options struct.

env

is a pointer to the environment struct. The `env` parameter must not be NULL.

Return Values:

This function returns pointer to transport `out` protocol string if set, else returns NULL.

xis2_options_get_soap_version_uri

Synopsis:

```
AXIS2_EXTERN const axis2_char_t* axis2_options_get_soap_version_uri
                                 ( const axis2_options_t *  options,
                                   const axutil_env_t *  env  )
```

Description:

This function gets the SOAP version URI.

Parameters:

`options`
pointer to options struct.

`env`
is a pointer to the environment struct. The `env` parameter must not be NULL.

Return Values:

This function returns the string representing SOAP version URI.

axis2_options_get_timeout_in_milli_seconds

Synopsis:

```
AXIS2_EXTERN long axis2_options_get_timeout_in_milli_seconds
                 ( const axis2_options_t *  options,
                   const axutil_env_t *  env  )
```

Description:

This function gets the wait time after which a client times out in a blocking scenario. The default is `AXIS2_DEFAULT_TIMEOUT_MILLISECONDS`.

Parameters:

`options`
pointer to options struct.

`env`
is a pointer to the environment struct. The `env` parameter must not be NULL.

Return Values:

This function returns timeout in milliseconds.

axis2_options_get_parent

Synopsis:

```
AXIS2_EXTERN axis2_options_t* axis2_options_get_parent
                              ( const axis2_options_t *  options,
                                const axutil_env_t *  env )
```

Description:

This function gets the parent options.

Parameters:

`options`
pointer to options struct.

`env`
is a pointer to the environment struct. The `env` parameter must not be NULL.

Return Values:

This function returns a pointer to the parent options structure if set, else returns NULL.

axis2_options_set_parent

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_parent
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              const axis2_options_t *  parent )
```

Description:

This function sets the parent options.

Parameters:

options

pointer to options struct.

env

is a pointer to the environment struct. The `env` parameter must not be NULL.

parent

pointer to parent options struct.

Return Values:

This function returns `AXIS2_SUCCESS` on success, else returns `AXIS2_FAILURE`.

axis2_options_set_action

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_action
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              const axis2_char_t *  action )
```

Description:

This function sets the WSA action.

Parameters:

options

pointer to options struct.

env

is a pointer to the environment struct. The `env` parameter must not be NULL.

action

pointer to action string.

Return Values:

This function returns `AXIS2_SUCCESS` on success, else returns `AXIS2_FAILURE`.

axis2_options_set_fault_to

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_fault_to
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              axis2_endpoint_ref_t *  fault_to )
```

Description:

This function sets the `fault_to` address.

Parameters:

options

pointer to options struct.

env

is a pointer to the environment struct. The `env` parameter must not be NULL.

fault_to
pointer to endpoint reference structure representing fault_to address. *options* takes over the ownership of the struct.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_set_from

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_from
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              axis2_endpoint_ref_t *  from )
```

Description:

This function sets from address.

Parameters:

options
pointer to options struct.

env
is a pointer to the environment struct. The env parameter must not be NULL.

from
pointer to endpoint reference structure representing from to address. *options* takes over the ownership of the struct.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_set_to

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_to
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              axis2_endpoint_ref_t *  to )
```

Description:

This function sets the to address.

Parameters:

options
pointer to options structure

env
is a pointer to the environment struct. The env parameter must not be NULL.

to
pointer to endpoint reference structure representing to address. *options* takes over the ownership of the struct.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_set_transport_receiver

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_transport_receiver
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              axis2_transport_receiver_t *  receiver )
```

Description:

This function sets transport receiver.

Parameters:

options
pointer to options struct.

env
is a pointer to the environment struct. The env parameter must not be NULL.

receiver
pointer to transport receiver struct options takes over the ownership of the struct.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_set_transport_in

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_transport_in
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              axis2_transport_in_desc_t *  transport_in )
```

Description:

This function sets transport_in description.

Parameters:

options
pointer to options structure

env
is a pointer to the environment struct. The env parameter must not be NULL.

transport_in
pointer to transport_in struct. options takes over the ownership of the struct.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_set_transport_in_protocol

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_transport_in_protocol
                           ( axis2_options_t *  options,
                             const axutil_env_t *  env,
                             const AXIS2_TRANSPORT_ENUMS  transport_in_protocol )
```

Description:

This function sets the transport_in_protocol.

Parameters:

options
pointer to options struct.

env
is a pointer to the environment struct. The env parameter must not be NULL.

in_protocol
pointer to in_protocol struct.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_set_message_id

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_message_id
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              const axis2_char_t *  message_id )
```

Description:

This function sets message ID.

Parameters:

options

pointer to options structure

env

is a pointer to the environment struct. The env parameter must not be NULL.

message_id

pointer to message_id structure

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_set_properties

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_properties
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              axutil_hash_t *  properties )
```

Description:

This function sets the properties hash map.

Parameters:

options

pointer to options struct.

env

is a pointer to the environment struct. The env parameter must not be NULL.

properties

pointer to properties hash map. options takes over the ownership of the hash struct.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_set_property

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_property
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              const axis2_char_t *  property_key,
                              const void *  property )
```

Description:

This function sets a property with the given key value.

Parameters:

options

pointer to options struct.

env
is a pointer to the environment struct. The env parameter must not be NULL.

property_key
property key string.

property
pointer to property to be set.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_set_reply_to

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_reply_to
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              axis2_endpoint_ref_t *  reply_to )
```

Description:

This function sets reply to address.

Parameters:

options
pointer to options structure

env
is a pointer to the environment struct. The env parameter must not be NULL.

reply_to
pointer to endpoint reference structure representing reply to address. options takes over the ownership of the struct.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_set_transport_out

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_transport_out
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              axis2_transport_out_desc_t *  transport_out )
```

Description:

This function sets the transport out description.

Parameters:

options
pointer to options structure

env
is a pointer to the environment struct. The env parameter must not be NULL.

transport_out
pointer to transport out description struct. options takes over the ownership of the struct.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_set_sender_transport

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_sender_transport
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              const AXIS2_TRANSPORT_ENUMS  sender_transport,
                              axis2_conf_t *  conf )
```

Description:

This function sets the sender transport.

Parameters:

options
pointer to options struct.

env
is a pointer to the environment struct. The env parameter must not be NULL.

sender_transport
name of the sender transport to be set.

conf
pointer to conf struct, it is from the conf that the transport is picked with the given name.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_set_soap_version_uri

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_soap_version_uri
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              const axis2_char_t *  soap_version_uri )
```

Description:

This function sets the SOAP version URI.

Parameters:

options
pointer to options structure

env
is a pointer to the environment struct. The env parameter must not be NULL.

soap_version_uri
URI of the SOAP version to be set, can be either AXIOM_SOAP11_SOAP_ENVELOPE_NAMESPACE_URI or AXIOM_SOAP12_SOAP_ENVELOPE_NAMESPACE_URI.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_set_timeout_in_milli_seconds

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_timeout_in_milli_seconds
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              const long  timeout_in_milli_seconds )
```

Description:

This function sets timeout in milliseconds.

Parameters:

options
pointer to options structure

env
is a pointer to the environment struct. The env parameter must not be NULL.

timeout_in_milli_seconds
timeout in milli seconds. The value can range from 0 to 2,147,483,647 milli seconds.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_set_transport_info

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_transport_info
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              const AXIS2_TRANSPORT_ENUMS  sender_transport,
                              const AXIS2_TRANSPORT_ENUMS  receiver_transport,
                              const axis2_bool_t  use_separate_listener )
```

Description:

This function sets transport information. Transport information includes the name of the sender transport, name of the receiver transport and if a separate listener to be used to receive response.

Parameters:

options
pointer to options struct.

env
is a pointer to the environment struct. The env parameter must not be NULL.

sender_transport
name of sender transport to be used.

receiver_transport
name of receiver transport to be used.

use_separate_listener
bool value indicating whether to use a separate listener or not.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_get_manage_session

Synopsis:

```
AXIS2_EXTERN axis2_bool_t axis2_options_get_manage_session
                           ( const axis2_options_t *  options,
                             const axutil_env_t *  env )
```

Description:

This function gets manage session bool value.

Parameters:

options
pointer to options structure

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns AXIS2_TRUE if session is managed, else returns AXIS2_FALSE.

axis2_options_set_manage_session

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_manage_session
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              const axis2_bool_t  manage_session )
```

Description:

This function sets manage session bool value.

Parameters:

options

pointer to options struct.

env

is a pointer to the environment struct. The env parameter must not be NULL.

manage_session

manage session bool value.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_set_msg_info_headers

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_msg_info_headers
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              axis2_msg_info_headers_t *  msg_info_headers )
```

Description:

This function sets WSA message information headers.

Parameters:

options

pointer to options struct.

env

is a pointer to the environment struct. The env parameter must not be NULL.

pointer

to message information headers struct.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_get_msg_info_headers

Synopsis:

```
AXIS2_EXTERN axis2_msg_info_headers_t* axis2_options_get_msg_info_headers
                                       ( const axis2_options_t *  options,
                                         const axutil_env_t *  env )
```

Description:

This function gets WSA message information headers.

Parameters:

options

pointer to options structure

env

is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns pointer to message information headers structure if set, else returns NULL.

axis2_options_get_soap_version

Synopsis:

```
AXIS2_EXTERN int axis2_options_get_soap_version
                ( const axis2_options_t *  options,
                  const axutil_env_t *  env )
```

Description:

This function gets SOAP version.

Parameters:

options

pointer to options structure

env

is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns AXIOM_SOAP11 if SOAP version 1.1 is in use, else returns AXIOM_SOAP12.

axis2_options_set_soap_version

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_soap_version
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              const int  soap_version )
```

Description:

This function sets the SOAP version.

Parameters:

options

pointer to options structure

env

is a pointer to the environment struct. The env parameter must not be NULL.

soap_version

soap version, either AXIOM_SOAP11 or AXIOM_SOAP12.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_get_soap_action

Synopsis:

```
AXIS2_EXTERN axutil_string_t* axis2_options_get_soap_action
                              ( const axis2_options_t *  options,
                                const axutil_env_t *  env  )
```

Description:

This function gets the SOAP action.

Parameters:

options

pointer to options struct.

env

is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns SOAP Action string if set, else returns NULL.

axis2_options_set_soap_action

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_soap_action
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              axutil_string_t *  soap_action )
```

Description:

This function sets the SOAP action.

Parameters:

options

pointer to options struct.

env

is a pointer to the environment struct. The env parameter must not be NULL.

action

pointer to SOAP action string.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_free

Synopsis:

```
AXIS2_EXTERN void axis2_options_free
                  ( axis2_options_t *  options,
                    const axutil_env_t *  env )
```

Description:

This function frees the options struct.

Parameters:

options

pointer to options structure

env

is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns void.

axis2_options_set_http_headers

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_options_set_http_headers
                            ( axis2_options_t *  options,
                              const axutil_env_t *  env,
                              axutil_array_list_t *  http_header_list )
```

Description:

This function sets the additional HTTP headers to be sent.

Parameters:

options

pointer to options struct.

env

is a pointer to the environment struct. The env parameter must not be NULL.

http_header_list
array list containing HTTP Headers.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

axis2_options_create

Synopsis:

```
AXIS2_EXTERN axis2_options_t* axis2_options_create
                                ( const axutil_env_t *  env   )
```

Description:

This function creates the options struct.

Parameters:

env
is a pointer to the environment struct. The env parameter must not be NULL.

Return Values:

This function returns a pointer to newly created options struct, or returns NULL on error with the error code set in the environment's error.

Context Hierarchy

message context

The axis2_msg_ctx_get_base() Function

Synopsis:

```
AXIS2_EXTERN axis2_ctx_t* axis2_msg_ctx_get_base
                          ( const axis2_msg_ctx_t *  msg_ctx,
                            const axutil_env_t *  env )
```

Description:

This function gets the base, which is of type context.

Parameters:

msg_ctx
is the message context

env
is a pointer to an environment structure. It cannot have a NULL value.

Return Values:

This function returns a pointer to base context structure.

The axis2_msg_ctx_get_parent() Function

Synopsis:

```
AXIS2_EXTERN struct axis2_op_ctx* axis2_msg_ctx_get_parent
                                  ( const axis2_msg_ctx_t *  msg_ctx,
                                    const axutil_env_t *  env )
```

Description:

This function gets the parent. Parent of a message context is of type operation context.

Parameters:

msg_ctx
is the message context

env
is a pointer to an environment structure. It cannot have a NULL value.

Return Values:

This function returns a pointer to operation context which is the parent.

The axis2_msg_ctx_set_parent() Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_msg_ctx_set_parent
                            ( axis2_msg_ctx_t *  msg_ctx,
                              const axutil_env_t *  env,
                              struct axis2_op_ctx *  parent )
```

Description:

This function sets the parent. Parent of a message context is of type operation context.

Parameters:

`msg_ctx`
is the message context

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

`parent`
pointer to parent operation context

Return Values:

- `AXIS2_SUCCESS` on success.
- `AXIS2_FAILURE` on failure.

The axis2_msg_ctx_free() Function

Synopsis:

```
AXIS2_EXTERN void axis2_msg_ctx_free
                  ( axis2_msg_ctx_t *  msg_ctx,
                    const axutil_env_t *  env )
```

Description:

This function frees the message context.

Parameters:

`msg_ctx`
is the message context

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

Return Values:

Void.

The axis2_msg_ctx_get_soap_envelope() Function

Synopsis:

```
AXIS2_EXTERN struct axiom_soap_envelope* axis2_msg_ctx_get_soap_envelope
                                        ( const axis2_msg_ctx_t *  msg_ctx,
                                          const axutil_env_t *  env )
```

Description:

This function gets the SOAP envelope. This SOAP envelope could be either a request SOAP envelope or the response SOAP envelope, based on the state the message context is in.

Parameters:

`msg_ctx`
is the message context

env
is a pointer to an environment structure. It cannot have a NULL value.

Return Values:

This function returns a pointer to the SOAP envelope stored within message context.

The axis2_msg_ctx_get_response_soap_envelope() Function

Synopsis:

```
AXIS2_EXTERN struct axiom_soap_envelope* axis2_msg_ctx_get_response_soap_envelope  ( const axis2_msg_ctx_t *
msg_ctx,
                                        const axutil_env_t *  env )
```

Description:

This function gets the SOAP envelope of the response.

Parameters:

msg_ctx
is the message context

env
is a pointer to an environment structure. It cannot have a NULL value.

Return Values:

This function returns a pointer to response SOAP envelope stored within message context.

The axis2_msg_ctx_get_fault_soap_envelope() Function

Synopsis:

```
AXIS2_EXTERN struct axiom_soap_envelope* axis2_msg_ctx_get_fault_soap_envelope
                                        ( const axis2_msg_ctx_t *  msg_ctx,
                                          const axutil_env_t *  env )
```

Description:

This function gets the fault SOAP envelope.

Parameters:

msg_ctx
is the message context

env
is a pointer to an environment structure. It cannot have a NULL value.

Return Values:

This function returns a pointer to the fault SOAP envelope stored within message context.

The axis2_msg_ctx_set_msg_id() Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_msg_ctx_set_msg_id
                                ( const axis2_msg_ctx_t *  msg_ctx,
                                  const axutil_env_t *  env,
                                  axis2_char_t *  msg_id )
```

Description:

This function sets the message ID.

Parameters:

msg_ctx
is the message context

env
is a pointer to an environment structure. It cannot have a NULL value.

msg_id

Return Values:

- AXIS2_SUCCESS on success.
- AXIS2_FAILURE on failure.

The axis2_msg_ctx_get_msg_id() Function

Synopsis:

```
AXIS2_EXTERN const axis2_char_t* axis2_msg_ctx_get_msg_id
                                  ( const axis2_msg_ctx_t *  msg_ctx,
                                    const axutil_env_t *  env )
```

Description:

This function gets the message ID.

Parameters:

msg_ctx
: is the message context

env
: is a pointer to an environment structure. It cannot have a NULL value.

Return Values:

This function returns the message ID string corresponding to the message context.

The axis2_msg_ctx_get_server_side() Function

Synopsis:

```
AXIS2_EXTERN axis2_bool_t axis2_msg_ctx_get_server_side
                          ( const axis2_msg_ctx_t *  msg_ctx,
                            const axutil_env_t *  env )
```

Description:

Parameters:

msg_ctx
: is the message context

env
: is a pointer to an environment structure. It cannot have a NULL value.

Return Values:

- AXIS2_SUCCESS on success.
- AXIS2_FAILURE on failure.

The axis2_msg_ctx_set_soap_envelope() Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_msg_ctx_set_soap_envelope
                            ( axis2_msg_ctx_t *  msg_ctx,
                              const axutil_env_t *  env,
                              struct axiom_soap_envelope *  soap_envelope )
```

Description:

This function sets the SOAP envelope. The fact that if it is the request SOAP envelope or that of response depends on the current status represented by message context.

Parameters:

msg_ctx
: is the message context

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

`soap_envelope`
is a pointer to the SOAP envelope. The message context assumes ownership of SOAP envelope.

Return Values:

- `AXIS2_SUCCESS` on success.
- `AXIS2_FAILURE` on failure.

The `axis2_msg_ctx_set_response_soap_envelope()` Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_msg_ctx_set_response_soap_envelope  ( axis2_msg_ctx_t *  msg_ctx,
  const axutil_env_t *  env,
  struct axiom_soap_envelope *  soap_envelope
 )
```

Description:

This function sets the response SOAP envelope.

Parameters:

`msg_ctx`
is the message context

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

`soap_envelope`
is a pointer to the SOAP envelope. The message context assumes ownership of SOAP envelope.

Return Values:

- `AXIS2_SUCCESS` on success.
- `AXIS2_FAILURE` on failure.

The `axis2_msg_ctx_set_fault_soap_envelope()` Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_msg_ctx_set_fault_soap_envelope
                            ( axis2_msg_ctx_t *  msg_ctx,
                              const axutil_env_t *  env,
                              struct axiom_soap_envelope *  soap_envelope )
```

Description:

This function sets the fault SOAP envelope.

Parameters:

`msg_ctx`
is the message context.

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

`soap_envelope`
is a pointer to the SOAP envelope. The message context assumes ownership of SOAP envelope.

Return Values:

- `AXIS2_SUCCESS` on success.
- `AXIS2_FAILURE` on failure.

The axis2_msg_ctx_set_message_id() Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_msg_ctx_set_message_id
                            ( axis2_msg_ctx_t *  msg_ctx,
                              const axutil_env_t *  env,
                              const axis2_char_t *  message_id )
```

Description:

This function sets the message ID.

Parameters:

`msg_ctx`
is the message context.

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

`message_id`

Return Values:

- `AXIS2_SUCCESS` on success.
- `AXIS2_FAILURE` on failure.

The axis2_msg_ctx_set_server_side() Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_msg_ctx_set_server_side  ( axis2_msg_ctx_t *  msg_ctx,
  const axutil_env_t *  env,
  const axis2_bool_t  server_side
 )
```

Description:

This function sets the boolean value indicating if it is the server side or the client side.

Parameters:

`msg_ctx`
is the message context.

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

`server_side`

- `AXIS2_TRUE` if it is server side
- `AXIS2_FALSE` if it is client side

Return Values:

- `AXIS2_SUCCESS` on success.
- `AXIS2_FAILURE` on failure.

The axis2_msg_ctx_get_msg_info_headers() Function

Synopsis:

```
AXIS2_EXTERN axis2_msg_info_headers_t* axis2_msg_ctx_get_msg_info_headers
                                       ( const axis2_msg_ctx_t *  msg_ctx,
                                         const axutil_env_t *  env )
```

Description:

This function gets the WS-Addressing message information headers.

Parameters:

`msg_ctx`
is the message context.

env
is a pointer to an environment structure. It cannot have a NULL value.

Return Values:

This function returns a pointer to message information headers structure with WS-Addressing information. It returns a reference, not a cloned copy.

The axis2_msg_ctx_is_keep_alive() Function

Synopsis:

```
AXIS2_EXTERN axis2_bool_t axis2_msg_ctx_is_keep_alive
                          ( const axis2_msg_ctx_t *  msg_ctx,
                            const axutil_env_t *  env )
```

Description:

This function gets the boolean value indicating the keep alive status. It is possible to keep alive the message context by any handler. By calling this method one can see whether it is possible to clean the message context.

Parameters:

msg_ctx
is the message context.

env
is a pointer to an environment structure. It cannot have a NULL value.

Return Values:

- AXIS2_TRUE if the message context is kept alive
- AXIS2_FALSE if the message context is not alive

The axis2_msg_ctx_set_keep_alive() Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_msg_ctx_set_keep_alive
                            ( axis2_msg_ctx_t *  msg_ctx,
                              const axutil_env_t *  env,
                              const axis2_bool_t  keep_alive )
```

Description:

This function

Parameters:

msg_ctx
is the message context.

env
is a pointer to an environment structure. It cannot have a NULL value.

keep_alive

Return Values:

- AXIS2_TRUE if the message context is kept alive
- AXIS2_FALSE if the message context is not alive

The axis2_msg_ctx_get_op_ctx() Function

Synopsis:

```
AXIS2_EXTERN struct axis2_op_ctx* axis2_msg_ctx_get_op_ctx
                                  ( const axis2_msg_ctx_t *  msg_ctx,
                                    const axutil_env_t *  env )
```

Description:

This function gets operation context related to the operation that this message context is related to.

Parameters:

`msg_ctx`
is the message context.

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

Return Values:

This function returns a pointer to operation context structure.

The `axis2_msg_ctx_get_svc_ctx_id()` Function

Synopsis:

```
AXIS2_EXTERN const axis2_char_t* axis2_msg_ctx_get_svc_ctx_id
                                 ( const axis2_msg_ctx_t *  msg_ctx,
                                   const axutil_env_t *  env )
```

Description:

This function gets the ID of service context that relates to the service that is related to the message context.

Parameters:

`msg_ctx`
is the message context.

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

Return Values:

This function returns the service context ID string.

The `axis2_msg_ctx_set_svc_ctx_id()` Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axis2_msg_ctx_set_svc_ctx_id
                            ( axis2_msg_ctx_t *  msg_ctx,
                              const axutil_env_t *  env,
                              const axis2_char_t *  svc_ctx_id )
```

Description:

This function sets the ID of the service context that relates to the service that is related to the message context.

Parameters:

`msg_ctx`
is the message context.

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

`svc_ctx_id`
is the service context ID string

Return Values:

- `AXIS2_TRUE` if the message context is kept alive
- `AXIS2_FALSE` if the message context is not alive

The axis2_msg_ctx_get_conf_ctx() Function

Synopsis:

```
AXIS2_EXTERN struct axis2_conf_ctx* axis2_msg_ctx_get_conf_ctx
                                    ( const axis2_msg_ctx_t *  msg_ctx,
                                      const axutil_env_t *  env )
```

Description:

This function gets the configuration context.

Parameters:

`msg_ctx`
is the message context.

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

Return Values:

This function returns pointer to configuration context.

The axis2_msg_ctx_get_svc_ctx() Function

Synopsis:

```
AXIS2_EXTERN struct axis2_svc_ctx* axis2_msg_ctx_get_svc_ctx
                    ( const axis2_msg_ctx_t *  msg_ctx,
                      const axutil_env_t *  env )
```

Description:

This function

Parameters:

`msg_ctx`
is the message context.

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

Return Values:

Service Skeleton API

AXIS2_SVC_SKELETON_INIT

Synopsis:

```
#define AXIS2_SVC_SKELETON_INIT  ( svc_skeleton,
  env    )     ((svc_skeleton)->ops->init (svc_skeleton, env))
```

Description:

Initialize the svc skeleton.

AXIS2_SVC_SKELETON_INIT_WITH_CONF

Synopsis:

```
#define AXIS2_SVC_SKELETON_INIT_WITH_CONF  ( svc_skeleton,
  env,
  conf    )     ((svc_skeleton)->ops->init_with_conf (svc_skeleton, env, conf))
```

Description:

Initialize the svc skeleton with axis2c configuration struct.

AXIS2_SVC_SKELETON_FREE

Synopsis:

```
#define AXIS2_SVC_SKELETON_FREE  ( svc_skeleton,
  env    )     ((svc_skeleton)->ops->free (svc_skeleton, env))
```

Description:

Frees the svc skeleton.

AXIS2_SVC_SKELETON_INVOKE

Synopsis:

```
#define AXIS2_SVC_SKELETON_INVOKE  ( svc_skeleton,
  env,
  node,
  msg_ctx    )     ((svc_skeleton)->ops->invoke (svc_skeleton, env, node, msg_ctx))
```

Description:

Invokes axis2 service skeleton.

AXIS2_SVC_SKELETON_ON_FAULT

Synopsis:

```
#define AXIS2_SVC_SKELETON_ON_FAULT  ( svc_skeleton,
  env,
  node    )     ((svc_skeleton)->ops->on_fault (svc_skeleton, env, node))
```

Description:

Called on fault.

Utilities

NonStop SOAP 4 also provides APIs for Logging, error reporting, string operations, array list, and hash tables.

Utilities For Logging

The following functions can be used to log messages during run time.

The axutil_log_impl_log_critical() Function

Synopsis:

```
AXIS2_EXTERN void  axutil_log_impl_log_critical
                  (axutil_log_t *log, const axis2_char_t *filename,
                   const int linenumber, const axis2_char_t *format,...)
```

Description:

This function logs the Critical Level logs in the specified log file.

Parameters:

`log`
is a pointer to the log structure.

`filename`
is the name of the file in which logs are recorded.

`linenumber`
specifies the line number where the message is generated.

`format`
is the string format of the message

The axutil_log_impl_log_error() Function

Synopsis:

```
AXIS2_EXTERN void  axutil_log_impl_log_error
                   (axutil_log_t *log, const axis2_char_t *filename,
                    const int linenumber, const axis2_char_t *format,...)
```

Description:

This function logs the error messages in the specified log file.

Parameters:

`log`
is a pointer to the log structure.

`filename`
is the name of the file in which logs are recorded.

`linenumber`
specifies the line number where the message is generated.

`format`
is the string format of the message

The axutil_log_impl_log_warning() Function

Synopsis:

```
AXIS2_EXTERN void  axutil_log_impl_log_warning
                   (axutil_log_t *log, const axis2_char_t *filename,
                    const int linenumber, const axis2_char_t *format,...)
```

Description:

This function logs the warning messages in the specified log file.

Parameters:

`log`
is a pointer to the log structure.

`filename`
is the name of the file in which logs are recorded.

`linenumber`
specifies the line number where the message is generated.

`format`
is the string format of the message

The axutil_log_impl_log_info() Function

Synopsis:

```
AXIS2_EXTERN void  axutil_log_impl_log_info
                   (axutil_log_t *log, const axis2_char_t *filename, const int linenumber, const axis2_char_t
*format,...)
```

Description:

This function logs information messages in the specified log file.

Parameters:

`log`
is a pointer to the log structure.

`filename`
is the name of the file in which logs are recorded.

`linenumber`
is the line number where the message is generated.

`format`
is the string format of the message

The `axutil_log_impl_log_user()` Function

Synopsis:

```
AXIS2_EXTERN void  axutil_log_impl_log_user
                  (axutil_log_t *log, const axis2_char_t *filename,
                   const int linenumber, const axis2_char_t *format,...)
```

Description:

This function logs user-level debug messages in the specified log file.

Parameters:

`log`
is a pointer to the log structure.

`filename`
is the name of the file in which logs are recorded.

`linenumber`
specifies the line number where the message is generated.

`format`
is the string format of the message

The `axutil_log_impl_log_debug()` Function

Synopsis:

```
AXIS2_EXTERN void  axutil_log_impl_log_debug
                  (axutil_log_t *log, const axis2_char_t *filename,
                   const int linenumber, const axis2_char_t *format,...)
```

Description:

This function logs the debug level logs. It logs all the information in the specified log file.

Parameters:

`log`
is a pointer to the log structure.

`filename`
is the name of the file in which logs are recorded.

`linenumber`
specifies the line number where the message is generated.

`format`
is the string format of the message

The `axutil_log_impl_log_trace()` Function

Synopsis:

```
AXIS2_EXTERN void  axutil_log_impl_log_trace
                   (axutil_log_t *log, const axis2_char_t *filename,
                    const int linenumber, const axis2_char_t *format,...)
```

Description:

This function logs the trace level logs. It is enabled with the compiler time option `AXIS2_TRACE`.

Parameters:

`log`
is a pointer to the log structure.

`filename`
is the name of the file in which logs are recorded.

`linenumber`
specifies the line number where the message is generated.

`format`
is the string format of the message

The `axutil_log_free()` Function

Synopsis:

```
AXIS2_EXTERN void  axutil_log_free
                   (axutil_allocator_t *allocator,
                    struct axutil_log *log)
```

Description:

This function releases the log files.

Parameters:

`allocator`
is the allocator to be used. It is a mandatory parameter and cannot have a `Null` value.

`log`
is a pointer to the log structure that is freed.

Return Values:

Void.

The `axutil_log_create()` Function

Synopsis:

```
AXIS2_EXTERN axutil_log_t *  axutil_log_create
                             (axutil_allocator_t *allocator,
                              axutil_log_ops_t *ops,
                              const axis2_char_t *stream_name)
```

Description:

This function creates a log structure.

Parameters:

`allocator`
is the allocator to be used. It is a mandatory parameter and cannot have a `Null` value.

`ops`
is an options structure to set the log options. It is a mandatory parameter and cannot have a `Null` value.

`stream_name`
is the name of the file to which the log is returned.

Return Values:

This function returns a pointer to the newly created log structure.

The `axutil_log_create_default()` Function

Synopsis:

```
AXIS2_EXTERN axutil_log_t *  axutil_log_create_default
                             (axutil_allocator_t *allocator)
```

Description:

Creates the log structure with default settings.

Parameters:

`-allocator`
is the allocator to be used. It is a mandatory parameter and cannot have a `Null` value.

Return Values:

This function returns a pointer to the newly created log structure.

Utilities For Error Reporting

The following functions set the error code and error message during run time.

The axutil_error_free() Function

Synopsis:

```
AXIS2_EXTERN void axutil_error_free
                  ( struct axutil_error *  error)
```

Description:

This function de-allocates an error structure instance.

Parameters:

`error`
is a pointer to an error structure instance to be freed.

Return Values:

Void.

The axutil_error_get_message() Function

Synopsis:

```
AXIS2_EXTERN const axis2_char_t* axutil_error_get_message
                                  ( const struct axutil_error *  error )
```

Description:

This function gets the error message corresponding to the last error occurred.

Parameters:

`error`
is a pointer to an error structure instance.

Return Values:

This function returns the error message.

The axutil_error_set_error_message() Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axutil_error_set_error_message
                            ( struct axutil_error *  error,
                              axis2_char_t *  message )
```

Description:

This function sets error message to the given value.

Parameters:

`error`
is a pointer to an error structure instance.

`message`
is the error message to be set.

Return Values:

- `AXIS2_SUCCESS` on success.
- `AXIS2_FAILURE` on failure.

The axutil_error_set_error_number() Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axutil_error_set_error_number
                            ( struct axutil_error *  error,
                              axutil_error_codes_t  error_number )
```

Description:

This function sets the error number to the given value.

Parameters:

`error`
is a pointer to an error structure instance.

`error_number`
is the error number to be set.

Return Values:

This function returns AXIS2_SUCCESS on success, else returns AXIS2_FAILURE.

The axutil_error_set_status_code() Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axutil_error_set_status_code
                            ( struct axutil_error *  error,
                              axis2_status_codes_t  status_code)
```

Description:

This function sets the status code to the given value.

Parameters:

error
is a pointer to an error structure instance.

status_code
is the status code to be set

Return Values:

- AXIS2_SUCCESS on success.
- AXIS2_FAILURE on failure.

The axutil_error_get_status_code() Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axutil_error_get_status_code
                            ( struct axutil_error *  error )
```

Description:

This function gets the status code.

Parameters:

`error`
is a pointer to an error structure instance.

Return Values:

This function returns the last status code set in the error structure instance.

Utilities For String Operations

The following functions enable you to perform string operations in NonStop SOAP 4.

The axutil_strdup() Function

Synopsis:

```
AXIS2_EXTERN void *  axutil_strdup
                     (const axutil_env_t *env, const void *ptr)
```

Description:

This function duplicates the string passed to it.

Parameters:

`env`
: is a pointer to an environment structure. It cannot have a `NULL` value.

`ptr`
: is a pointer to the string that is duplicated.

Return Values:

This function returns a pointer to the newly created string.

The axutil_strndup() Function

Synopsis:

```
AXIS2_EXTERN void* axutil_strndup
                   ( const axutil_env_t *  env,
                     const void *  ptr,
                     int  n )
```

Description:

This function duplicates the first n characters of a string in the memory.

Parameters:

`env`
: is a pointer to an environment structure. It cannot have a `NULL` value.

`ptr`
: The string to duplicate.

`n`
: The number of characters to duplicate.

Return Values:

This function returns the new string.

The axutil_strmemdup() Function

Synopsis:

```
AXIS2_EXTERN void* axutil_strmemdup
                   ( const void *  ptr,
                     size_t  n,
                     const axutil_env_t *  env )
```

Description:

This function creates a null-terminated string by making a copy of a sequence of characters and appending a null byte. This is a faster alternative to `axis2_strndup`, for use when you know that the string being duplicated has 'n' or more characters. If the string contains fewer characters, use `axis2_strndup`.

Parameters:

`ptr`
: is the block of characters that should be duplicated.

`n`
: is the number of characters that should be duplicated.

env
is an input parameter and is a pointer to the environment structure. It cannot have a NULL value.

Return Values:

This function returns the new string.

The axutil_strcmp() Function

Synopsis:

```
AXIS2_EXTERN int  axutil_strcmp
                  (const axis2_char_t *s1,
                   const axis2_char_t *s2)
```

Description:

This function compares the strings passed to it.

Parameters:

s1
is the first string passed to the function.

s2
is the second string passed to the function.

Return Values:

- The function returns zero if the two strings are identical.
- The function returns a non-zero integer if the two strings are different.

The axutil_strncmp() Function

Synopsis:

```
AXIS2_EXTERN int  axutil_strncmp
                  (const axis2_char_t *s1,
                   const axis2_char_t *s2, int n)
```

Description:

This function compares the first n characters of a string in the memory.

Parameters:

s1
is the first string passed to the function.

s2
is the second string passed to the function.

n
is the number of characters to be compared.

Return Values:

- The function returns zero if the two strings are identical.
- The function returns a non-zero integer if the two strings are different.

The axutil_strlen() Function

Synopsis:

```
AXIS2_EXTERN axis2_ssize_t  axutil_strlen
                            (const axis2_char_t *s)
```

Description:

This function gets the length of the string passed to it .

Parameters:

`s`
is the string passed.

Return Values:

This function returns the length of the string.

The `axutil_strcasecmp()` Function

Synopsis:

```
AXIS2_EXTERN int  axutil_strcasecmp
                  (const axis2_char_t *s1,
                   const axis2_char_t *s2)
```

Description:

This function compares two strings.

Parameters:

`s1`
is the first string to be compared.

`s2`
is the second string to be compared.

Return Values:

- The function returns zero if the two strings are identical.
- The function returns a non-zero integer if the two strings are different.

The `axutil_strncasecmp()` Function

Synopsis:

```
AXIS2_EXTERN int  axutil_strncasecmp
                  (const axis2_char_t *s1,
                   const axis2_char_t *s2, const int n)
```

Description:

This function compares two strings. It ignores the case of both arguments.

Parameters:

`s1`
is the first string to be compared.

`s2`
is the second string to be compared.

Return Values:

- The function returns zero if the two strings are identical.
- The function returns a non-zero integer if the two strings are different.

The `axutil_stracat()` Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t *  axutil_stracat
                             (const axutil_env_t *env,
                              const axis2_char_t *s1,
                              const axis2_char_t *s2)
```

Description:

This function concatenates two strings.

Parameters:

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`s1`
is the first string.

`s2`
is the second string.

Return Values:

This function returns the new string.

The `axutil_strcat()` Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axutil_strcat
                            ( const axutil_env_t *  env,
                                ...    )
```

Description:

This function concatenates multiple strings.

Parameters:

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`...`
strings, separated by commas, to be concatenated.

Return Values:

This function returns the new string.

The `axutil_strstr()` Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t *  axutil_strstr
                             (const axis2_char_t *haystack,
                              const axis2_char_t *needle)
```

Description:

This function finds finds the first occurrence of the substring needle in the string haystack.

Parameters:

`haystack`
is the string in which the given string is to be found.

`needle`
is the string to be found in haystack.

Return Values:

This function returns pointer to the beginning of the substring, or `NULL` if the substring is not found.

The `axutil_strcasestr()` Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axutil_strcasestr
                            ( const axis2_char_t *  haystack,
                              const axis2_char_t *  needle )
```

Description:

This function finds the first occurrence of the substring needle in the string haystack. It ignores the case of both arguments.

Parameters:

`haystack`
is the string in which the given string is to be found.

`needle`
is the string to be found in haystack.

Return Values:

This function returns pointer to the beginning of the substring, or `NULL` if the substring is not found.

The `axutil_strchr()` Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axutil_strchr
                           ( const axis2_char_t *  s,
                             axis2_char_t  ch )
```

Description:

This function finds the first occurrence of a character in a string.

Parameters:

`s`
is the string in which the character is searched.

`ch`
is the character to be searched.

Return Values:

This function returns pointer to the first occurrence of the character in the string.

The `axutil_replace()` Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t *  axutil_replace
                             (const axutil_env_t *env,
                              axis2_char_t *str,
                              int s1, int s2)
```

Description:

This function replaces a string by another string.

Parameters:

`env`
is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`str`
is the string in which the input string (`s1`) is searched .

`s1`
is the string to be searched in `str`.

`s2`
is the string that replaces `s1`.

Return Values:

This function returns the new string.

The axutil_strltrim() Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t *  axutil_strltrim
                             (const axutil_env_t *env,
                              const axis2_char_t *s,
                              const axis2_char_t *trim)
```

Description:

This function trims the sequence or pattern specified in `trim` from the left of the string.

Parameters:

`env`
: is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`s`
: is the string which is trimmed.

`trim`
: specifies the sequence or pattern to be trimmed.

Return Values:

This function returns the new string.

The axutil_strrtrim() Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t *  axutil_strrtrim
                            (const axutil_env_t *env,
                             const axis2_char_t *_s,
                             const axis2_char_t *_trim)
```

Description:

This function trims the sequence or pattern specified in `trim` from the right of the string.

Parameters:

`env`
: is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`s`
: is the string which is trimmed.

`trim`
: specifies the sequence or pattern to be trimmed.

Return Values:

This function returns the new string.

The axutil_strtrim() Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t *  axutil_strtrim
                             (const axutil_env_t *env,
                              const axis2_char_t *_s,
                              const axis2_char_t *_trim)
```

Description:

This function trims the sequence or pattern specified in `trim` from left or right of the string.

Parameters:

`env`
: is an input parameter and is a pointer to the environment structure. It cannot have a `NULL` value.

`s`
is the string which is trimmed.

`trim`
specifies the sequence or pattern to be trimmed.

Return Values:

This function returns the new string.

The `axutil_string_replace()` Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axutil_string_replace
                           ( axis2_char_t *  str,
                             axis2_char_t  old_char,
                             axis2_char_t  new_char )
```

Description:

This function replaces the given `axis2_character` with a new one.

Parameters:

`str`
is the string in which an old character is replaced by a new character

`old_char`
is the old character that is replaced.

`new_char`
is the new character that replaces the new character.

Return Values:

This function returns the replaced string.

The `axutil_string_substring_starting_at()` Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axutil_string_substring_starting_at
                           ( axis2_char_t *  str,
                             int  s )
```

Description:

This function gives a substring starting from a given index.

Parameters:

`str`
is the source string.

`c`
is the starting index.

Return Values:

This function returns the substring.

The `axutil_string_substring_ending_at()` Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axutil_string_substring_ending_at
                           ( axis2_char_t *  str,
                             int  e )
```

Description:

This function gives a substring ending with a given index.

Parameters:

`str`
is the source string.

`c`
is the ending index.

Return Values:

This function returns the substring.

The `axutil_string_tolower()` Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axutil_string_tolower
                           ( axis2_char_t *  str   )
```

Description:

This function sets a string to lowercase.

Parameters:

`str`
is the string to be converted to lowercase.

Return Values:

This function returns the string with lowercase.

The `axutil_string_toupper()` Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axutil_string_toupper
                           ( axis2_char_t *  str   )
```

Description:

This function sets a string to uppercase.

Parameters:

`str`
is the string to be converted to uppercase.

Return Values:

This function returns the string with uppercase.

Utilities For Array List

The following functions enable you to perform array operations in NonStop SOAP 4.

The `axutil_array_list_create()` Function

Synopsis:

```
AXIS2_EXTERN axutil_array_list_t* axutil_array_list_create
                                  ( const axutil_env_t *  env,
                                    int  capacity )
```

Description:

This function constructs a new array list with the supplied initial capacity. If capacity is invalid (<= 0), the default capacity is used

Parameters:

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

`capacity`
is the initial capacity of thes `array_list`.

Return Values:

This function returns the newly created array list.

The axutil_array_list_size() Function

Synopsis:

```
AXIS2_EXTERN int axutil_array_list_size
                  ( struct axutil_array_list *  array_list,
                    const axutil_env_t *  env )
```

Description:

This function returns the number of elements in this list.

Parameters:

`array_list`
is a pointer to the array list.

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

Return Values:

This function returns the list size.

The axutil_array_list_is_empty() Function

Synopsis:

```
AXIS2_EXTERN axis2_bool_t axutil_array_list_is_empty
                          ( struct axutil_array_list *  array_list,
                            const axutil_env_t *  env )
```

Description:

This function checks if the list is empty.

Parameters:

`array_list`
is a pointer to the array list

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

Return Values:

- `AXIS2_SUCCESS` on success.
- `AXIS2_FAILURE` on failure.

The axutil_array_list_contains() Function

Synopsis:

```
AXIS2_EXTERN axis2_bool_t axutil_array_list_contains
                           ( struct axutil_array_list *  array_list,
                             const axutil_env_t *  env,
                             void *  e  )
```

Description:

This function returns `true` if the element is in the `array_list`.

Parameters:

`array_list`
is a pointer to the array list.

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

e
is the element whose inclusion in the list is being tested.

Return Values:

This function returns `true` if the list contains `e`.

The `axutil_array_list_add()` Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axutil_array_list_add
                            ( struct axutil_array_list *  array_list,
                              const axutil_env_t *  env,
                              const void *  e  )
```

Description:

This function appends the supplied element to the end of this list. The element, `e`, can be a pointer of any type or `NULL`.

Parameters:

`array_list`
is a pointer to the array list.

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

`e`
is the element whose inclusion in the list is being tested

Return Values:

- `AXIS2_SUCCESS` on success.
- `AXIS2_FAILURE` on failure.

The `axutil_array_list_add_at()` Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axutil_array_list_add_at
                            ( struct axutil_array_list *  array_list,
                              const axutil_env_t *  env,
                              const int  index,
                              const void *  e )
```

Description:

This function adds the supplied element at the specified index, shifting all elements currently at that index or higher one to the right. The element, `e`, can be a pointer of any type or `NULL`.

Parameters:

`array_list`
is a pointer to the array list.

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

`index`
is the index at which the element is being added.

`e`
is the element whose inclusion in the list is being tested.

Return Values:

- `AXIS2_SUCCESS` on success.
- `AXIS2_FAILURE` on failure.

The axutil_array_list_remove() Function

Synopsis:

```
AXIS2_EXTERN void* axutil_array_list_remove
                   ( struct axutil_array_list *  array_list,
                     const axutil_env_t *  env,
                     int  index )
```

Description:

This function removes the element at the user-supplied index.

Parameters:

`array_list`
is a pointer to the array list.

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

`index`
is the index at which the element is being added.

Return Values:

This function returns the the removed `void*` pointer.

The axutil_array_list_free() Function

Synopsis:

```
AXIS2_EXTERN void axutil_array_list_free
                  ( struct axutil_array_list *  array_list,
                    const axutil_env_t *  env )
```

Parameters:

`array_list`
is a pointer to the array list.

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

Return Values:

Void.

Utilities For Hash Tables

The axutil_hash_first() Function

Synopsis:

```
AXIS2_EXTERN axutil_hash_index_t* axutil_hash_first
                                   ( axutil_hash_t *  ht,
                                     const axutil_env_t *  env )
```

Description:

Parameters:

Return Values:

The axutil_hash_next() Function

Synopsis:

```
AXIS2_EXTERN axutil_hash_index_t* axutil_hash_next
                                   ( const axutil_env_t *  env,
                                     axutil_hash_index_t *  hi )
```

Description:

Continue iterating over the entries in a hash table.

Parameters:

`hi`

The iteration state.

Return Values:

This function returns a pointer to the updated iteration state. It returns `NULL` if there are no more entries.

The `axutil_hash_count()` Function

Synopsis:

```
AXIS2_EXTERN unsigned int axutil_hash_count
                            ( axutil_hash_t *  ht )
```

Description:

This function gets the number of key/value pairs in the hash table.

Parameters:

`ht`

is the hash table.

Return Values:

This function returns the number of key/value pairs in the hash table.

The `axutil_hash_free()` Function

Synopsis:

```
AXIS2_EXTERN void axutil_hash_free
                  ( axutil_hash_t *  ht,
                    const axutil_env_t *  env )
```

Description:

Parameters:

`ht`

hash table to be freed

`env`

is a pointer to an environment structure. It cannot have a `NULL` value.

Return Values:

This function returns the status code.

The `axutil_hash_contains_key()` Function

Synopsis:

```
AXIS2_EXTERN axis2_bool_t axutil_hash_contains_key
                          ( axutil_hash_t *  ht,
                            const axutil_env_t *  env,
                            const axis2_char_t *  key )
```

Description:

Query whether the hash table provided as parameter contains the key provided as parameter.

Parameters:

`ht`

hash table to be queried for key

Return Values:

This function returns whether the hash table contains key

The axutil_hash_make() Function

Synopsis:

```
AXIS2_EXTERN axutil_hash_t* axutil_hash_make
                            ( const axutil_env_t *  env )
```

Description:

This function creates a hash table.

Parameters:

`env`
is a pointer to an environment structure. It cannot have a `NULL` value.

Return Values:

This function returns the newly created hash table.

The axutil_hash_get() Function

Synopsis:

```
AXIS2_EXTERN void* axutil_hash_get
                  ( axutil_hash_t *  ht,
                    const void *  key,
                    axis2_ssize_t  klen )
```

Description:

This function looks up the value associated with a key in a hash table.

Parameters:

`ht`
The hash table

`key`
Pointer to the key

`klen`
Length of the key. Can be AXIS2_HASH_KEY_STRING to use the string length.

Return Values:

This function returns NULL if the key is not present.

The axutil_hash_set() Function

Synopsis:

```
AXIS2_EXTERN void axutil_hash_set
                 ( axutil_hash_t *  ht,
                   const void *  key,
                   axis2_ssize_t  klen,
                   const void *  val )
```

Description:

This function associates a value with a key in a hash table.

Parameters:

`ht`
The hash table

`key`
Pointer to the key

`klen`
Length of the key. Can be AXIS2_HASH_KEY_STRING to use the string length.

`val`
Value to associate with the key.

Return Values:

If the value is NULL the hash entry is deleted.

qname

The axutil_qname_create() Function

Synopsis:

```
AXIS2_EXTERN axutil_qname_t* axutil_qname_create
                                ( const axutil_env_t *  env,
                                  const axis2_char_t *  localpart,
                                  const axis2_char_t *  namespace_uri,
                                  const axis2_char_t *  prefix )
```

Description:

This function creates a qname structure returns a pointer to a qname structure mandatory mandatory optional The prefix. Must not be null. Use "" (empty string) to indicate that no namespace URI is present or the namespace URI is not relevant if null is passed for prefix and uri , ""(empty string) will be assinged to those fields.

Return Values:

This function a pointer to newly created qname structure

The axutil_qname_create_from_string() Function

Synopsis:

```
AXIS2_EXTERN axutil_qname_t* axutil_qname_create_from_string
                                ( const axutil_env_t *  env,
                                  const axis2_char_t *  string )
```

Description:

Parameters:

Return Values:

This function returns a newly created qname using a string genarated from axutil_qname_to_string method freeing the returned qname is users responsibility

The axutil_qname_free() Function

Synopsis:

```
AXIS2_EXTERN void axutil_qname_free
                  ( struct axutil_qname *  qname,
                    const axutil_env_t *  env )
```

Description:

This function frees a qname structure

Parameters:

Return Values:

This function returns the status code.

The axutil_qname_equals() Function

Synopsis:

```
AXIS2_EXTERN axis2_bool_t axutil_qname_equals  ( const struct axutil_qname *  qname,
  const axutil_env_t *  env,
  const struct axutil_qname *  qname1
 )
```

Description:

This function compare two qname. The prefix is ignored when comparing. If ns_uri and localpart of qname1 and qname2 are equal, returns true.

Parameters:

Return Values:

This function returns true if qname1 equals qname2, false otherwise.

The axutil_qname_clone() Function

Synopsis:

```
AXIS2_EXTERN struct axutil_qname* axutil_qname_clone
                                  ( struct axutil_qname *  qname,
                                    const axutil_env_t *  env )
```

Description:

This function clones a given qname.

Parameters:

qname,qname

structure instance to be cloned environment , double pointer to environment

Return Values:

This function returns the newly cloned qname structure instance

The axutil_qname_get_uri() Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t *  axutil_qname_get_uri
                             (const struct axutil_qname *qname,
                              const axutil_env_t *env)
```

Description:

Parameters:

Return Values:

The axutil_qname_get_prefix() Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t *  axutil_qname_get_prefix
                             (const struct axutil_qname *qname,
                              const axutil_env_t *env)
```

Description:

Parameters:

klen

val

Return Values:

The axutil_qname_get_localpart() Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t *  axutil_qname_get_localpart
                             (const struct axutil_qname *qname,
                              const axutil_env_t *env)
```

Description:

Parameters:

Return Values:

The axutil_qname_to_string() Function

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axutil_qname_to_string
                           ( struct axutil_qname *  qname,
                             const axutil_env_t *  env )
```

Description:

Parameters:

Return Values:

This function returns a unique string created by concatenating namespace uri and localpart. The string is of the form localpart|url. When the qname free function is called, the returned char* is freed.

parameter

The axutil_param_create() Function

Synopsis:

```
AXIS2_EXTERN axutil_param_t* axutil_param_create
                             ( const axutil_env_t *  env,
                               axis2_char_t *  name,
                               void *  value )
```

Description:

This function creates the param struct.

Parameters:

Return Values:

Synopsis:

```
AXIS2_EXTERN axis2_char_t* axutil_param_get_name
                           ( struct axutil_param *  param,
                             const axutil_env_t *  env )
```

Description:

Parameter name accessor.

Parameters:

Return Values:

This function returns name of the param.

Synopsis:

```
AXIS2_EXTERN void* axutil_param_get_value
                  ( struct axutil_param *  param,
                    const axutil_env_t *  env )
```

Description:

Parameter value accessor

Parameters:

Return Values:

Thjis function returns the object.

Synopsis:

```
AXIS2_EXTERN axis2_status_t axutil_param_set_name
                            ( struct axutil_param *  param,
                              const axutil_env_t *  env,
                              const axis2_char_t *  name )
```

Description:

param name mutator

Parameters:

name

Return Values:

Synopsis:

```
AXIS2_EXTERN axis2_status_t axutil_param_set_value
                            ( struct axutil_param *  param,
                              const axutil_env_t *  env,
                              const void *  value )
```

Description:

Method setValue

Parameters:

value

Return Values:

Synopsis:

```
AXIS2_EXTERN axis2_bool_t axutil_param_is_locked
                           ( struct axutil_param *  param,
                             const axutil_env_t *  env )
```

Description:

Method isLocked

Parameters:

Return Values:

This fuction returns a boolean value.

The `axutil_param_set_locked()` Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t axutil_param_set_locked
                            ( struct axutil_param *  param,
                              const axutil_env_t *  env,
                              axis2_bool_t  value )
```

Description:

Method setLocked

Parameters:

value

Return Values:

The axutil_param_free() Function

Synopsis:

```
AXIS2_EXTERN void  axutil_param_free
                   (struct axutil_param *param,
                    const axutil_env_t *env)
```

Description:

Parameters:

Return Values:

The axutil_param_set_attributes() Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t  axutil_param_set_attributes
                             (struct axutil_param *param,
                              const axutil_env_t *env,
                              axutil_hash_t *attrs)
```

Description:

Parameters:

Return Values:

Synopsis:

```
AXIS2_EXTERN axutil_hash_t *  axutil_param_get_attributes
                              (struct axutil_param *param,
                               const axutil_env_t *env)
```

Description:

Parameters:

Return Values:

The axutil_param_set_value_free() Function

Synopsis:

```
AXIS2_EXTERN axis2_status_t  axutil_param_set_value_free
                             (struct axutil_param *param,
                              const axutil_env_t *env,
                              AXIS2_PARAM_VALUE_FREE free_fn)
```

Description:

Parameters:

Return Values:

E Transaction Management Model in NonStop SOAP 3 and NonStop SOAP 4

The transaction management model followed by NonStop SOAP 4 is different from the transaction management model of NonStop SOAP 3.

"Transaction Management Models in NonStop SOAP 3 and NonStop SOAP 4" (page 398) highlights the differences between NonStop SOAP 3 Transaction Management Model and the NonStop SOAP 4 Transaction Management model.

Table 26 Transaction Management Models in NonStop SOAP 3 and NonStop SOAP 4

NonStop SOAP 3 Transaction Management Model	NonStop SOAP 4 Transaction Management Model
Persistent session object encapsulates all transactions	Dummy session ID has no bearing on transactions
Session ID is exposed to the client	Transaction ID is exposed to the client.
Cookie file is maintained	Cookie file is not maintained.
Single SOAP header block is used for handling sessions and transactions	Two distinct SOAP header blocks are used for handling sessions and transactions.
Session management is permanently implemented in the SOAP server object.	Transaction management, implemented as a module, can be detached from the NonStop SOAP 4 server.
The `TMFTransactionSupport` attribute accepts the following values: • `yes` • `no`	The `TMFTransactionSupport` attribute accepts the following values: • `Required` • `Supports` • `Never`

Glossary

A

administrator For a NonStop system, the person responsible for the installation and configuration of a software subsystem on a NonStop node. Contrast with operator.

ANSI The American National Standards Institute.

API See application program interface.

application program interface A set of services (such as programming language functions or procedures) that are called by an application program to communicate with other software components.

attribute An item of descriptive data associated with an XML element. An attribute has a name and a value.

B

Business Logic A set of the functional algorithms that handle information exchange between a data store and a user interface.

C

cache A temporary storage buffer.

CGI See Common Gateway Interface (CGI)..

client An application program that requests services to be performed. In discussions of the Pathway environment, this term is used to refer to the part of an application that runs on some other vendor's hardware, such as a personal computer, Macintosh computer, UNIX workstation, or mainframe computer system, and makes requests of a server process.

Client APIs A set of function calls that can be used to create client applications.

client stub files Template files generated using the `WSDL2C` tool, to design the NonStop SOAP application client.

client/server model A model for distributing applications. In general, but not always, in this model the client process resides on a workstation and the server process resides on a second workstation, minicomputer, or mainframe system. Communication takes the form of request and reply pairs, which are initiated by the client and serviced by the server. (A server can make requests of another server, thus acting as a client.) Client/server computing is often used to connect different types of workstations or personal computers to a host computer system by means of supported communications protocols.

command line client A NonStop SOAP 4 client that can be run from the OSS command prompt.

Common Gateway Interface (CGI). A standard protocol used as the interface between Web servers and the programs these servers use to process requests from Web clients.

configuration The definition or alteration of characteristics of an object.

configuration context The configuration data needed to start an instance of NonStop SOAP 4.

configuration file A file that includes configuration information for NonStop SOAP 4 server and itp WebServer and underlying Web services. The configuration files for NonStop SOAP 4 include `axis2.xml`, `module.xml`, `services.xml`, and `itp_axis2.config`.

context free A style of application access in which the application regards each request as independent of all other requests. As a result, a different process can handle each request in a set or series.

context sensitive A style of application access in which a series of requests are logically related, requiring the same server process to handle all requests in the series.

Contract-First development This approach of application development considers a WSDL definition as the starting point of the development process. The WSDL becomes a contract between the service provider and the client. The WSDL definition is created using Web services standards and the service implementation is performed, based on this contract.

D

Data Definition Language (DDL). The set of data definition statements within the Structured Query Language (SQL). An HP product for defining data objects in Enscribe files on NonStop™ systems and translating object definitions into source code.

data type A categorization of values, operations, and arguments that usually cover both behavior and representation.

deployment directory This is an OSS directory that can be considered as the working directory where all the services, modules, and log files are available.

deployment script A script file used to deploy the NonStop SOAP 4 server and setup the deploy environment.

directory A type of Open System Services (OSS) special file that contains directory entries that name links to other files. No two directory entries in the same directory can have the same name.

DLL A DLL is a library that contains code and data, which can be used by more than one program at the same time.

E

element The basic unit of information in an XML document. An element has a name and can have content and attributes.

envelope The elements of a SOAP message that precede and follow the body of the message.

environment variable An element of a system or product-specific configuration file used to specify the path to a processing component or to control some aspect of system or product operation. For example, you use an environment variable in the OSS profile file to specify the location of the NonStop configuration file.

Event Management Service (EMS) A part of DSM used to provide event collection, event logging, and event distribution facilities. It provides for different descriptions of events for people and for programs, lets an operator or application select specific event message data, and allows for flexible distribution of event messages within a system or network. EMS has an SPI-based programmatic interface for reporting and retrieving events.

event message A special kind of SPI message that describes an event occurring in the system or network. Event messages are collected, logged, and distributed by EMS.

F

file An object to which data can be written or from which data can be read. A file has attributes, such as access permissions and a file type. In the Open System Services (OSS) environment, file types include regular file, character special file, block special file, FIFO, and directory.

file descriptor A non-negative integer that uniquely identifies a single open of a file to a running process. Each file descriptor is associated with an open file description that contains data about the file.

file identifier (file ID) In the OSS environment, a portion of the internal information used to identify a file in the OSS file system. In the Guardian environment, the portion of a file name following the subvolume name. The two identifiers are not comparable.

file name In the Guardian environment, the set of node name, volume name, subvolume name, and file identifier characters that uniquely identifies a file. This name is used to open a file and thereby provide a connection between the opening process and the file. In the Open System Services (OSS) environment, a component of a pathname containing any valid characters other than a slash (/) or a null.

file system (1) In the Guardian environment, the application program interface for communication between a process and a file. A file can be a disk file, a device other than a disk, or another process. (2) In the OSS environment, a collection of files and file attributes. A file system provides the namespace for the file serial numbers that uniquely identify its files.

File Transfer Protocol (FTP). The Internet standard, high-level protocol for transferring files from one machine to another. Usually implemented as application-level programs, FTP uses the TELNET and Transmission Control Protocol (TCP) protocols. The server side requires a Web client to supply a login identifier and password before it will honor requests.

File Utility Program (FUP). An HP product on NonStop™ systems that allows users to create, copy, purge, and otherwise manipulate disk files interactively.

Flows A Flow is a sequence of activities where messages flow in and out of NonStop SOAP 4.

FTP See File Transfer Protocol (FTP).

fully qualified file name. The complete name of a file in the Guardian environment. For a permanent disk file, this consists of a node name (system name), volume name, subvolume name, and file identifier (file ID). In interactive interfaces, such as PATHCOM and TACL, the parts of a file name are separated by periods.

G

generated class A class, derived from one of the base SOAP classes, but specific to a TS/MP service or NonStop process defined in an SDL file.

generated file A file created by the `SoapAdminCL` tool and included in a custom SOAP server for TS/MP or NonStop process.

graphical user interface (GUI). A type of screen interface that typically includes pull-down menus, icons, dialog boxes, and online help.

Guardian An environment available for interactive or programmatic use with the NonStop™ Kernel operating system. Processes that run in the Guardian environment use the Guardian system procedure calls as their application program interface, and might also use related APIs, such as the Pathsend and TMF procedure calls.

Guardian services An application program interface (API) to the HP NonStop operating system, plus the tools and utilities associated with that API.

H

handler A handler is the smallest unit of invocation within a phase. Handlers are an independent unit of execution, which perform specific activities, such as logging a message, and consuming addressing headers.

Hot deployment Hot deployment is the process of deploying Web services while the NonStop SOAP 4 server is actively serving requests.

Hypertext Markup Language (HTML). The tagging language used to format Hypertext documents on the World Wide Web. It is built on top of Standard Generalized Markup Language (SGML).

Hypertext Transport Protocol (HTTP) The communications protocol used for transmitting data between servers and Web clients (browsers) on the World Wide Web.

I

Internationalization

Internationalization (or "I18N") is the provision for conversion of data to the encoding a client requires.

J

Java A programming language for browser-based internet applications.

M

marshal To serialize a C++/Java object into a data stream, for example, to convert a C++/Java object into an XML document.

MEP See message exchange pattern.

Message Context An object for every message, regardless of whether it is from the client or server or processed within NonStop SOAP 4. It stores information related to a particular message in the object.

message exchange pattern A message exchange pattern (MEP) describes the pattern of messages required by a communications protocol to establish or use a communication channel.

Message Receiver User Functions (MRUFs) These are a set functions that can be used to customize the Pathway message receiver process flow.

N

NonStop Process A process running on a NonStop server.

NonStop SOAP configuration file An XML document that defines the environment variables used by a SOAP server at runtime.

O

Open System Services (OSS) An open system environment available for interactive or programmatic use with the NonStop™ operating system. Processes that run in the OSS environment use the OSS application program interface; interactive users of the OSS environment use the OSS shell for their command interpreter.

Operation Context Operation context is the runtime representation of an operation; it is the direct map to the operation that is being invoked by a given message.

OSS environment See Open System Services (OSS).

OSS pathname The string of characters that uniquely identifies a file within its file system in the OSS environment. A pathname can be either absolute or relative.

P

PATHCOM (1) The command interface to the PATHMON process, through which users enter commands to configure and manage Pathway applications when Pathway/XM is not used. (2) The process that provides and supports the PATHCOM interface. In a Pathway/XM environment, PATHCOM processes are used internally only, to interpret management commands directed to individual PATHMON processes. You configure a Pathway/XM environment by using PXMCFG and start up, maintain, and shut down that environment by using PXMCOM.

PATHMON The central controlling process for a NonStop TS/MP application.

PATHMON environment The set of servers, server classes, TCPs, terminals, SCREEN COBOL programs, and tell messages that run together under the control of one PATHMON process.

PATHMON process (1) In a Pathway environment without Pathway/XM, the central controlling process for the environment. The PATHMON process maintains configuration-related data; grants links to server classes in response to requests from TCPs and LINKMON processes; and performs all process control (starting, monitoring, restarting, and stopping) of server processes and TCPs. (2) In a Pathway/XM environment, a process that monitors and manages a set of requester and server objects such as TCPs, TERM objects, and server classes (SERVER objects). A Pathway/XM environment can include multiple PATHMON processes, and all are configured and managed centrally through the SuperCTL file.

pathname See OSS pathname

Pathway The former name of NonStop TS/MP, a product providing transaction services for persistent, scalable, transaction-processing applications.

Pathway application A set of programs that perform online transaction processing tasks in the Guardian environment on NonStop™ systems , using interfaces defined by Compaq software. A Pathway application can include SCREEN COBOL requesters, Pathsend requesters, and Pathway servers. It can also include GDSX front-end processes and clients that use RSC/MP.

Pathway environment A run-time environment consisting of transaction-processing products provided by HP for the Guardian operating environment on NonStop™ systems . This term is often shortened to “Pathway environment.” Depending on the needs of your application, your Pathway environment could include NonStop™ TS/MP, the run-time portions of Pathway/iTS (the TCP and the SCREEN COBOL run-time environment), and processes from related products, such as NonStop™ TM/MP, GDSX, and RSC/MP.

Pathway server. A server process or program in the Pathway transaction processing environment.

Payload The payload is the actual content in the message that is sent to an application.

Phase A Phase is a stage of processing or a time interval in the message process. A collection of phases forms a Flow.

process A running entity that is managed by the operating system, as opposed to a program, which is a collection of code and data. When a program is taken from a file on a disk and run in a processor, the running entity is called a process.

R

request message buffer A character buffer created by the NonStop SOAP 4 server to be sent to the Pathway application or NonStop process.

response message buffer A character buffer received by the NonStop SOAP 4 server from the Pathway application or NonStop process.

response selection criteria NonStop SOAP 4 facilitates multiple non-fault response structures for a single request. To select which response structure must be selected the NonStop SOAP 4 server uses the response buffer value based on the values mentioned in the services.xml file, and generate a proper SOAP response. The values that are stored in the services.xml file like StartIndex, EndIndex, BufVal, ComparisionOp, ConversionOp, FaultResponse, and DefaultResponse are collectively known as the response selection criteria.

root directory An OSS directory associated with a process that the system uses for pathname resolution when a pathname begins with a slash (/) character. See also OSS pathname.

S

SDL Service Definition Language.

SDR Service Definition Repository.

server A process or set of processes that satisfy requests from Web clients in a clientserver environment.

server class A set of duplicate copies of a single server process, all of which execute the same object program. Server classes are configured through the PATHMON process or the PXMCFG utility.

SERVERCLASS_SEND The procedure call used to send a request to a TS/MP server class, defined as part of the TS/MP Pathsend facility.

Service Artifacts The files generated by the `SoapAdminCL` tool while deploying a service.

Service Context The Service Context holds all the operation contexts that are created, invoking the operations within that service.

Service skeleton files These files are template files, generated using the `WSDL2PWY` tool, to design the Pathway server class.

service-first development An application developmentt approach, where the development of the service starts from a code. The service code is written without the WSDL and the business logic is written later. This approach is also commonly known as the Code-First approach.

session A logical grouping of a requests used for to permit transaction control and to support access to context-sensitive classes.

session context Information a SOAP server maintains to represent a session.

session ID A handle used by the SOAP client and server to relate requests and responses to a session context.

Simple Object Access Protocol (SOAP) SOAP is a simple XML-based protocol that enables applications to exchange information over HTTP.

SOAP See Simple Object Access Protocol (SOAP)

SOAP server A process that converts a SOAP or XML request into the format required by an external service, and converts the reply from the external service into SOAP or XML.

soft link This is a type of file that contains a reference to another file or directory in the form of an absolute or relative path and affects pathname resolution.

subsession A series of interactions between a client and a SOAP server, corresponding to dialog between a SOAP server and a context-sensitive TS/MP server class or NonStop process. A session can include a series of subsessions, each with a different target server class.

T

TMF See Transaction Management Facility subsystem.

TMF transaction See Transaction Management Facility subsystem and transaction.

transaction An operation or a series of operations that retrieves and updates information to reflect an exchange of goods or services. In the process of retrieving and updating information, a transaction transforms a database from one consistent state to another. The TMF subsystem treats a transaction as a single unit; either all of the changes made by a transaction are made permanent (the transaction is committed) or none of the changes are made permanent (the transaction is aborted).

transaction management To coordinate transaction control functions, such as beginning and ending transactions, committing or aborting transactions, and recovering transactions.

Transaction Management Facility (TMF) subsystem The major component of the NonStop™ TM/MP product, which protects databases in online transaction processing environments. To furnish this service, the TMF subsystem manages database transactions, keeps track of database activity through audit trails, and provides database recovery methods.

TS/MP TS/MP software manages server pools by establishing links between clients and servers, balancing the workload across servers, automatically creating and deleting servers in response to changes in request traffic, and restarting servers after processor or process failures.

U

Url_pattern This is a unique string that specifies the Web address of the NonStop SOAP 4 deployment location.

user-exits These are functions used to customize NonStop SOAP 3 server.

V

volume A disk drive, or a pair of disk drives that forms a mirrored disk, in the Guardian environment. The name of a volume is the second of the four parts of a file name.

W

Web server Web servers are programs that execute on a variety of server platforms. Web server functions can be divided into two parts. A file server part performs normal file server functions, such as file transfer and buffering. A message switching facility allows messages from Web clients to be forwarded to application programs.

WS-Security WS-Security provides a platform to secure your services beyond transport level protocols, such as HTTPS. HTTPS performs a secure message transfer from one end point to another. However, in the real world, the message is transferred over multiple domains and you must preserve the identity, integrity, and security of the message across multiple trusted domains or points. WS-Security provides an end-to-end solution for Web service security.

WSDL Web Services Description Language (WSDL) is an XML-based language that captures the mechanical information a client needs to access a Web service: definitions of message formats, SOAP details, and a destination URL. WSDL provides a simple way for service providers to describe the basic format of requests to their systems regardless of the underlying protocol (such as Simple Object Access Protocol or XML).

X

XML Extensible Markup Language, a standard for tagging data in an HTML document, which describes as to provide semantic information about content elements.

XML schemas XML schemas express shared vocabularies and allow machines to carry out rules made by people. They provide a means for defining the structure, content, and semantics of XML documents.

Index

A

Abort
- AbortTransactionOnFault attribute, 162
- configuring transactions to Abort on fault, 248
- Session, 255
- timeout, 246
- Transaction, 244

Attaching
- module at global level, 181
- MRUF at global level, 181

C

Client APIs
- Basic client APIs, 105
- to generate Request Node and consume Response Node, 106
- to implement the Client Interface, 105

Communicating with Non-Pathway process, 231

configuring
- Abort on fault, 248
- level of transaction support, 245
- service as a DLL, 183
- service as a Pathway application, 183
- service as a process, 183
- transaction timeout, 247

Contract-first Approach, 260

Customizing Message Processing
- Overview, 124
- using Handlers, 126
- using MRUFs, 137

D

DDL comments, 217

Deploying
- adminserver, 50
- NonStop SOAP 4 Pathway web service, 269
- sample web service, 44
- Web Service developed as a NonStop Process, 83
- Web Service developed as a Pathway Application, 77

E

Error and Warning Messages, 285

F

For Contract-first Approach, 260

H

Hot-Deployment, 229

I

Internationalization and Encoding, 230

L

Logging
- Client APIs
  - creating log file structure, 115
  - logging levels, 115
  - releasing log structure, 116
- defining log levels, 178
- log levels, 178
- Service APIs, 95
  - creating log file structure, 95

M

MEP
- setting operation-specific MEP, 189
- setting up, 180

message processing
- customizing, 124
- specifying order of Phase invocation, 182

Message Receiver User Functions, 125

Migrating NonStop SOAP 3 services, 52
- prerequisites, 52
- services, 52
- transactions, 58
- user-exits, 61

MRUFs, 125

Multiple Deployment
- Setting up, 43

N

NonStop SOAP
- Architecture, 29
- Features, 28
- Introduction, 27
- Prerequisites, 36
- tools, 194

NonStop SOAP 4
- configuration files, 177
- configuring, 177
- Getting Started, 77
- Installing, 36
- Supported Standards, 27
- Transaction Management, 236

NonStop SOAP 4 Tools
- Administration Utility, 215
- SOAPAdminCL, 194
- WSDL2C, 211
- WSDL2PWY, 203

P

Phases
- user-defined, 131

Prerequisites, 36

R

Rampart Specific Assertions, 275

S

SDL
- attributes, 153

Creating SDL for service, 98
elements, 153
service types, 154
Service APIs
Basic service APIs, 85
developing services, 95
logging, 95
to extract input parameters and return response, 89
to implement the Service Interface, 85
to implement the Service Skeleton Structure interface, 87
Session
Abort Transaction in Session, 255
Begin, 250
Begin Transaction in Session, 251
Commit Transaction in Session, 254
Continue Transaction in Session, 253
End, 256
subsessions, 259
timeout, 258
Session Management, 249

T

Timeout
configuring Transaction timeout, 247
Session, 258
Transaction, 246
TMF transaction
controlling, 188
Tools
WSDL2C
NonStop SOAP 4 Non-Pathway web service, 270
WSDL2PWY
Develop a NonStop SOAP 4 Pathway web service, 262
Transaction
Abort, 244
transactions to Abort on fault, 248
Begin, 239
Commit, 242
Continue, 241
timeouts, 246
configuring transaction timeout, 247
Transaction Management
client, 238
Transaction Managment
server, 238
Transaction Module Configuration, 237

W

web service
develop using Service APIs, 95
WS-Security
Scenarios, 281
using, 273
WSDL
considerations, 260
pre-defined, 260