

Natural

Natural API Server

Version 9.2.5

October 2026

This document applies to Natural Version 9.2.5 and all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 1979-2026 Software GmbH, Darmstadt, Germany and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software GmbH product names are either trademarks or registered trademarks of Software GmbH and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software GmbH and/or its subsidiaries is located at <https://softwareag.com/licenses>.

Use of this software is subject to adherence to Software GmbH's licensing conditions and terms. These terms are part of the product documentation, located at <https://softwareag.com/licenses> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software GmbH Products / Copyright and Trademark Notices of Software GmbH Products". These documents are part of the product documentation, located at <https://softwareag.com/licenses> and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software GmbH.

Document ID: NATMF-NATNPI-925-20261009

Table of Contents

Preface	v
1 About this Documentation	1
Document Conventions	2
Online Information and Support	2
Data Protection	3
I Overview	5
2 Overview	7
What is Natural API Server	8
Architecture	8
About Natural API Server	9
II Natural Subprogram Server on z/OS	11
3 Introduction	13
Natural Subprogram Server Concept	14
NATRNP Front-end Stub	15
Front-end	16
Gateway Module	16
Server Monitor	16
Product Interaction	17
4 Installation	19
Prerequisites	20
Content of the Natural Subprogram Server Distribution Medium	20
Installation of Natural Subprogram Server on z/OS	21
5 Configuration	25
Configuration Requirements	26
External Security Configuration	27
Configuration File	27
Configuration Parameters	27
Configuration File Example	33
Natural Subprogram Server Datasets	33
6 Operation	35
Starting the Natural Subprogram Server	36
Terminating the Natural Subprogram Server	36
Monitoring the Natural Subprogram Server	37
Runtime Trace Facility	38
7 Security	39
Natural Security	40
External Security	40
III Natural OpenAPI Server on Tomcat	41
8 Installation	43
Installing the Natural OpenAPI Server	44
9 Configuration	45
General Configuration	46
Natural Error to HTTP Mapping	47

10 Operation	49
Deploying Web Application with Tomcat	50
Deploying OpenAPI Resources	50
11 Monitoring	53
12 Security	55
Secure Communication between Natural OpenAPI Server and Subprogram Server	56
Secure Communication between Client and Natural OpenAPI Server	56
Authentication	57
IV Natural OpenAPI Server on Liberty	59
13 Installation	61
Installing the Natural OpenAPI Server	62
14 Configuration	63
General Configuration	64
Natural Error to HTTP Mapping	65
15 Operation	67
Deploying Web Application with Liberty	68
Deploying OpenAPI Resources	69
16 Monitoring	71
17 Security	73
Secure Communication between Natural OpenAPI Server and Subprogram Server	74
Secure Communication between Client and Natural OpenAPI Server	74
Authentication	75
V Developer Overview	77
18 Installation of the NaturalONE OpenAPI Plugin	79
19 Configuration of the NaturalONE OpenAPI Plugin	81
20 Generate and Deploy Services	85
Generating OpenAPI Resources	86
Viewing OpenAPI Resources via Editor	87
21 Special Considerations	91
Developing Natural Subprograms for Natsubsrv	92

Preface

This documentation describes the Natural API Server

Overview
Natural Subprogram Server on z/OS
Natural OpenAPI Server on Tomcat
Natural OpenAPI Server on Liberty
Developer Overview

1

About this Documentation

■ Document Conventions	2
■ Online Information and Support	2
■ Data Protection	3

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Monospace font	Identifies service names and locations in the format <i>folder.subfolder.service</i> , APIs, Java classes, methods, properties.
<i>Italic</i>	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
Monospace font	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the symbol.
[]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at <https://documentation.softwareag.com>.

Product Training

You can find helpful product training material on our Learning Portal at <https://learn.software-ag.com>.

Tech Community

You can collaborate with Software GmbH experts on our Tech Community website at <https://tech-community.softwareag.com>. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software GmbH news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at <https://github.com/softwareag> and <https://hub.docker.com/publishers/softwareag> and discover additional Software GmbH resources.

Product Support

Support for Software GmbH products is provided to licensed customers via our Empower Portal at <https://empower.softwareag.com>. Many services on this portal require that you have an account. If you do not yet have one, you can request it at <https://empower.softwareag.com/register>. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

I Overview

2 Overview

■ What is Natural API Server	8
■ Architecture	8
■ About Natural API Server	9

This document covers the following topics:

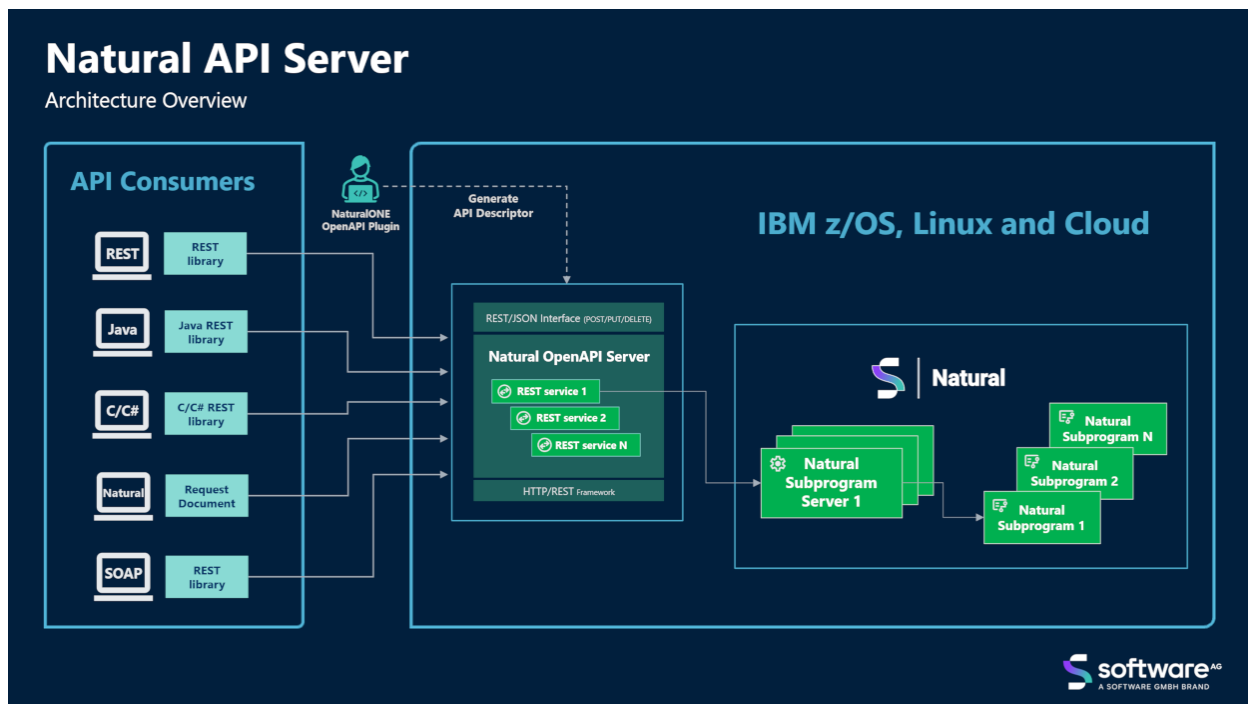
What is Natural API Server

The Natural API Server provides access to Natural subprograms through REST API calls based on OpenAPI standard. This allows external applications to access and reuse Natural business logic without rewriting existing Natural subprograms. The OpenAPI descriptions can be generated automatically.

Architecture

The Natural API Server consists of the following components:

- Natural Subprogram Server for z/OS or Linux and Cloud.
- Natural OpenAPI Server as interface between REST API clients and the Natural Subprogram Server. This component runs standalone as Embedded Tomcat® or is deployed in a Java Application Servers like Apache Tomcat® or Liberty®.
- NaturalONE OpenAPI Plugin to generate and deploy Natural OpenAPI services.



About Natural API Server

The Natural API Server is available as an installed component of your Natural for z/OS or Linux environment, or as an Eclipse Plugin for NaturalONE.

II Natural Subprogram Server on z/OS

The Natural Subprogram Server exposes Natural subprograms as services for integration into modern application architectures.



Note: It does not provide a direct REST API for client applications. The access is always performed via Natural OpenAPI Server, which is the middleware component that exposes a REST API and forwards requests to the Natural Subprogram Server using an internal protocol.

This document covers the following topics:

Introduction
Installation
Configuration
Operation
Security

3 Introduction

■ Natural Subprogram Server Concept	14
■ NATRNPI Front-end Stub	15
■ Front-end	16
■ Gateway Module	16
■ Server Monitor	16
■ Product Interaction	17

This document covers the following topics:

Natural Subprogram Server Concept

A Natural Subprogram Server is a multi-user, multi-tasking application. It can host Natural sessions for multiple users and execute their requests concurrently.

The concept is based on the "serverized" Natural runtime system. Its architecture comprises a server front-end stub (NATRNP I subprogram server stub) that uses the Natural front-end to dispatch Natural sessions and to execute subprograms within these sessions.

The architecture basically consists of:

- **Front-end stub**

The NATRNPI stub is launched to initialize a Natural Subprogram Server. It listens for incoming requests and initializes a Natural session for executing the subprogram.

- **Front-end**

The front-end is called (together with the Natural runtime system) by the front-end stub for session initialization/termination and request execution.

- **Gateway module**

The NATGWNPI module provides for interaction between the Natural runtime system and the front-end stub. NATGWNPI is already included in the Natural nucleus and is called by the Natural runtime system to exchange the necessary request data.

- **Natural driver**

Natural is driven by the Natural Driver, NATOS / NATOSL.



Note: NATOSL is an extension of NATOS that supports IBM's Language Environment (LE). LE allows different programming languages to be used within a single application. Because the server is implemented in C and requires LE, the Natural front end must always include both NATOS and NATOSL.

- **Server monitor**

A monitor task allows the administrator to control the server activities and to terminate the entire server, etc.

NATRNPI Front-end Stub

NATRNPI is a multi-user and multi-tasking front-end stub, and it is launched to initialize a Natural Subprogram Server.

This document covers the following topics:

- [Stub Description](#)
- [Natural System Variables Used](#)
- [Natural I/O Handling](#)

Stub Description

The task executing the server initialization, `TMain`, is the main listener which waits for incoming requests from the Natural OpenAPI Server. It owns a session directory to manage multiple clients (users) and their corresponding remote Natural sessions. `TMain` has the task to accept all incoming requests and to dispatch them to other subtasks (`TWork`). The process is as follows:

- First, the Natural OpenAPI Server connects to `TMain` to establish a connection.
- Next, `TMain` inserts the client into its session directory, attaches a new `TWork` subtask, and passes the connection to `TWork`.
- `TWork` initializes a new Natural session, processes the request, replies the result to the client, terminates the Natural session, and terminates itself.

Each request is stateless. For each request, a separate Natural session is started. However, the session initialization is accelerated by cloning new sessions from a master session. At server initialization, a master session is initialized and kept in a user-independent state. For a new session, the master is just copied and connected with the user-ID of the request.

Worker Pool

It is expensive to create a new worker task for each request. It is possible to keep created workers in a pool to reuse them for successive requests. If the configuration variable `WORKER_DELAY` is set to a value greater than zero, a worker waits after request execution for the defined time in seconds. The worker pool automatically adjusts its size based on the server's workload. When there are many parallel requests, the pool grows to handle the increased load. As the workload decreases, the pool shrinks accordingly.

Natural System Variables Used

Within a Natural OpenAPI Server session, the following Natural system variables are used:

- *TPSYS contains SERVSTUB.
- *DEVICE contains BATCH.
- *SERVER-TYPE contains CALLNAT.

Natural I/O Handling

The called subprogram must not do any screen-io. If a screen-io happens, the server dumps the content of the screen to its trace file and cancels the screen-io.

Front-end

The Natural front-end required for a Natural Subprogram Server is a Natural batch front-end driver, which should be LE enabled. See sample installation jobs for details.

Gateway Module

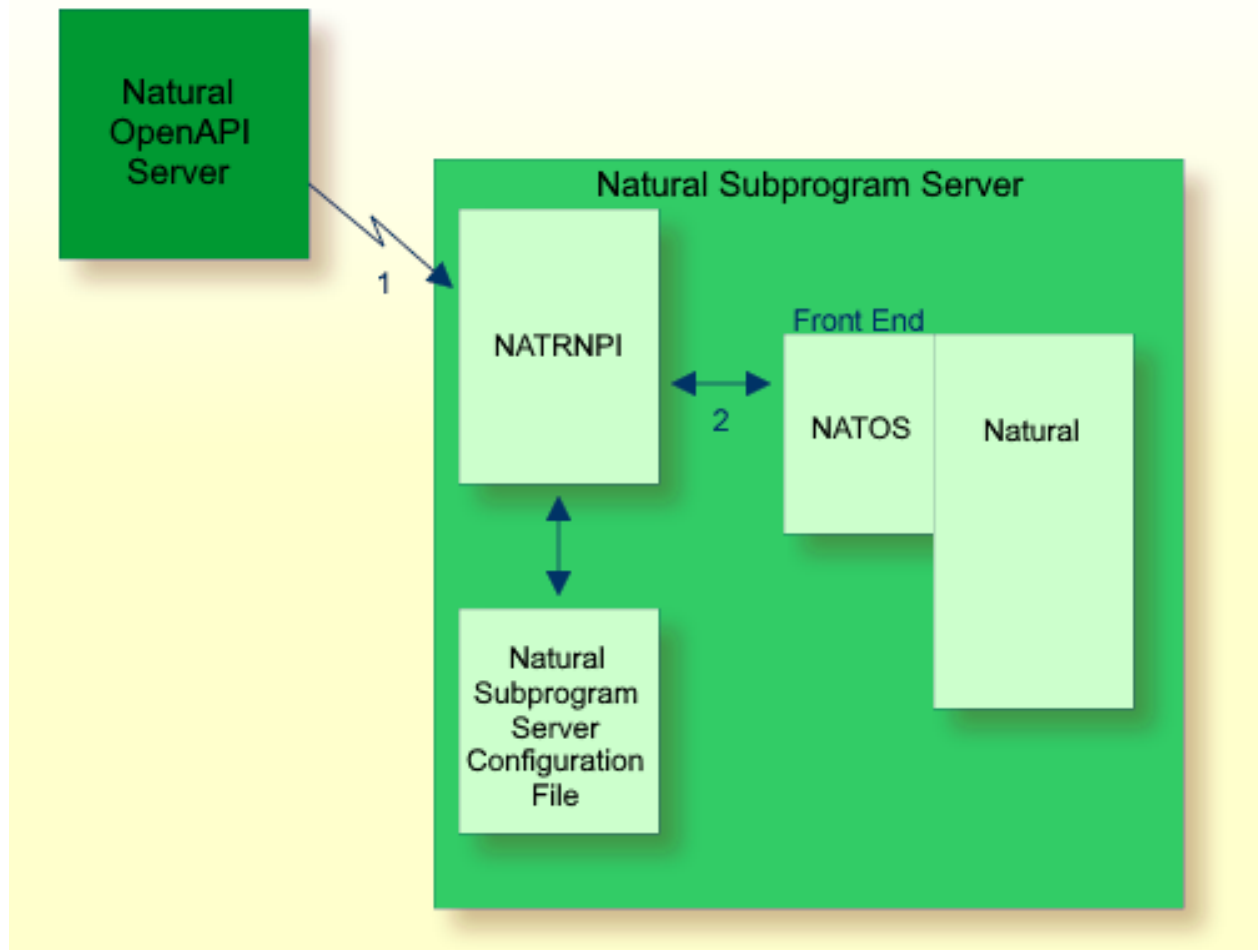
The NATGWNPI gateway module is already included in the Natural nucleus.

Server Monitor

To enable the administrator to monitor the status of the Natural Subprogram Server, a monitor task is provided and is automatically initialized at server startup. Using the monitor commands, the administrator can control the server activities, terminate the entire server, etc. For more information, see *Natural API Server > Natural Subprogram Server on z/OS > Operation > Monitoring the Natural API Server > [Monitor Communication](#)*.

Product Interaction

The following graphic illustrates the interaction of the Natural OpenAPI Server with a Natural Subprogram Server.



1. Natural OpenAPI Server (the client) gets a REST call. It converts it into a proprietary protocol and sends it to the Natural Subprogram Server using the port number specified with the Natural OpenAPI Server.
2. The Natural Subprogram Server initializes (clones) a Natural session using the Natural front-end you have specified with `FRONTEND_NAME`, the Natural Subprogram Server configuration parameter. In this example is NATOS.

4 Installation

■ Prerequisites	20
■ Content of the Natural Subprogram Server Distribution Medium	20
■ Installation of Natural Subprogram Server on z/OS	21

This document describes how to install the Natural Subprogram Server on z/OS.

This document covers the following topics:

Prerequisites

- The currently applicable version of Natural for z/OS must be installed. For more information, see <https://empower.softwareag.com/>.



Note: Any user-written exits not written in Natural and are used within a subprogram server environment must be reentrant and thread-safe (capable to run in a multi-tasking environment).

- Natural Subprogram Server version 1.1.1 is compatible with Natural for z/OS version 9.2.5 or later.
- To prevent the formation of endless loops in user programs running under the Natural Subprogram Server, specify a reasonable value for the Natural profile parameter, MT (Maximum CPU Time).

Content of the Natural Subprogram Server Distribution Medium

The installation medium contains the datasets listed in the table below. The sequence of the datasets and the number of library blocks needed are displayed in the Product Delivery Report, which accompanies the installation medium.

Dataset Name	Content
NP1vrs.OBJS	Contains the object modules of the Natural Subprogram Server.
NP1vrs.JOBS	Example of installation jobs.

The notation `vrs` in the dataset name represents the version, release, and system maintenance level of the product.

For the currently applicable versions, refer to <https://empower.softwareag.com/>.

Installation of Natural Subprogram Server on z/OS

The installation of the Natural Subprogram Server on z/OS is described in the following steps:

Step 1: Create a Natural Subprogram Server configuration file

(Job I009, Step 9110)

Step 9110 creates an NPICONFG sample for the Natural Subprogram Server.

The following parameters of the configuration file have to be defined. For other parameters, the default values may be used:

FRONTEND_NAME	Specifies the name of the Natural Subprogram Server front-end module you generate in Step 3 .
PORT_NUMBER	Specifies the TCP/IP port number under to which the server can be connected.

For more information on the parameters, see *Natural API Server > Natural Subprogram Server on z/OS > Configuration > [Configuration Parameters](#)*.

Step 2: Link the object modules into the Natural Subprogram Server load library

(Job I058, Step 9120)

The Natural Subprogram Server object modules must be linked with the necessary runtime extensions of your batch installations into executable load modules.

See NPI058 sample job on NPIvrs.JOBS dataset.

Step 3: Create the Natural Subprogram Server front-end module

(Job I060, Steps 9110, 9120, 9130)

- Job I060, Step 9110, start the IEBUPDATE batch program and store NPIPARM.
- Job I060, Step 9120, assemble and link the NPIPARM parameter module.
- Job I060, Step 9130, link the Natural front-end module.



Notes:

1. If you have Natural subproducts for IBM database access installed, for example, Natural for DB2, Natural for DL/I, or Natural for VSAM, you need to adjust the link job to include ATRCSS from SYS1.CSSLIB to enable the use of SYNCPOINT / ROLLBACK features.
2. The Natural front-end must be link-edited for reentrant or reusable execution.

Step 4: Create server startup JCL

(Job I200, Step 9115)

Described in the [Configuration](#) section. See NPISTART sample member on NPIvrs.JOBS dataset.

Example:

```
//          PROC  SRV=SAGNPI
//NDV       EXEC  PGM=NATRNPI,
// REGION=4000K,TIME=1440,PARM='POSIX(ON),TRAP(ON,NOSPIE)/&SRV'
//STEPLIB   DD    DISP=SHR,DSN=NPIvrs.LOAD
//          DD    DISP=SHR,DSN=SAGLIB.SMALOAD
//CEEDUMP   DD    SYSOUT=X
//CMPRINT   DD    SYSOUT=X
//STGCONFIG DD    DISP=SHR,
//          DSN=NPI.CONFIG(&SRV)
//STGTRACE  DD    SYSOUT=X
//STGSTDO   DD    SYSOUT=X
//STGSTDE   DD    SYSOUT=X
```



Note: The Natural Subprogram Server account must be defined in the z/OS Linux System Services (OE segment). If the server account is not defined, the server ends with U4093 and CEE5101C system message in the trace file.

Step 5: Clients must be defined in Natural Security

If Natural Security (NSC) is installed:

- The Natural Subprogram Server initial user ID (default ID is SYSTEM) must be defined in Natural Security with a valid default library. For more information, see *Natural API Server > Natural Subprogram Server on z/OS > Configuration > Configuration Parameters > [INITIAL_USERID](#)*.
- Each client user ID must be defined in Natural Security.

If the initial user ID is not defined, the Natural Subprogram Server initialization aborts with a NAT0856 error message.

If a client is not defined, the server returns an NSC error.

Step 6: Clients must be defined to the server host

If you configure the Natural Subprogram Server to use an external security system (see *Natural API Server > Natural Subprogram Server on z/OS > Configuration > Configuration Parameters > [SECURITY_MODE](#)*), the clients must be defined to the external security system.

5 Configuration

■ Configuration Requirements	26
■ External Security Configuration	27
■ Configuration File	27
■ Configuration Parameters	27
■ Configuration File Example	33
■ Natural Subprogram Server Datasets	33

This document describes how to configure a Natural Subprogram Server for z/OS (Batch).

This document covers the following topics:

Configuration Requirements

A Natural Subprogram Server for z/OS (Batch) requires the following z/OS language environment parameter configuration:

Parameter	Definition
POSIX(ON)	Enables a Natural Subprogram Server to access the POSIX functionality of z/OS. If you start a Natural Subprogram Server with POSIX(OFF), it terminates immediately with a U4093 user abend and the EDC5167 system message. IBM supplies the default OFF.
TRAP(ON,NOSPIE)	Defines the abend handling of the LE/370 environment: ON Enables the Language Environment condition handler. NOSPIE Specifies that the Language Environment will handle program interrupts and abends via an ESTAE. This means the Natural abend handler will receive control to handle program interrupts and abends. If you do not specify TRAP(ON,NOSPIE), the Natural abend handling will not work properly. IBM supplies the default (ON,SPIE).
TERMTHDACT(UADUMP)	Defines the level of information that is produced in case of an abend. The UADUMP option generates a Language Environment, CEEDUMP, and system dump of the user address space. The CEEDUMP does not contain the Natural relevant storage areas. IBM supplies the default (TRACE).
ENVAR(TZ=...)	The ENVAR option enables you to set Linux environment variables. The only environment variable applicable for the Natural Subprogram Server is TZ (time zone). This variable allows you to adjust the timestamp within the Natural Subprogram Server's trace file to your local time. Example (minus 1 hour daylight saving time): ENVAR(TZ=CET-1DST) CET

You can set the z/OS language environment parameters:

- With the PARM parameter specified in the EXEC card of the Natural Subprogram Server startup job. The length of the options is constrained by the maximum length of the PARM parameter.
- Assemble an LE/370 runtime option module CEEUOPT and link it to the Natural Subprogram Server load module.

External Security Configuration

If you configure the Natural Subprogram Server to impersonate the clients in the Natural Subprogram Server (configuration parameter `SECURITY_MODE` set to `IMPERSONATE_LOCAL`), the Natural Subprogram Server must run "program-controlled". Under RACF, the following definitions are required for the Natural Subprogram Server:

- The `BPX.SERVER` resource must be defined and the Natural Subprogram Server account must have `READ` access to this resource
- The `LOAD` datasets defined in the startup job definition must be defined to the program class "***"

```
ralt program ** addmem('natural load library') uacc(read)
ralt program ** addmem('NPI load library'//NOPADCHK) uacc(read)
ralt program ** addmem('user load library'//NOPADCHK) uacc(read)
```

- `SETR WHEN(PROGRAM) REFRESH`

Additionally, each client connecting to the server must be defined in RACF and be granted to use the z/OS Linux System Services.

Configuration File

A configuration file is allocated to the name `STGCONFIG`.

The configuration file is a text file located on a dataset or on an HFS file under z/OS.

The configuration file contains the server configuration parameters in the form of a `keyword=value` syntax. In addition, it may contain comments whose beginning is marked with a hash symbol (`#`).

For more information, see the [Configuration File Example](#) section below.

Configuration Parameters

The following configuration parameters are available:

FRONTEND_NAME

This configuration parameter specifies the name of the Natural front-end to be used to start a Natural session. The front-end resides on a PDS member. It needs to be linked re-entrant or re-usable.

Value	Explanation
frontend-name	Natural front-end to be used. Maximum length is 8 characters.
	No default value is provided.

Example:

```
FRONTEND_NAME=NATvrsSV
```

where `vrs` stands for the version, release, and system maintenance number.

FRONTEND_OPTIONS

The values of this configuration parameter may be used to specify additional options for the Natural front-end.

Value	Explanation
40	Do not use Clone. To accelerate the Natural session initialization, each new session is cloned from the master session created at server initialization. If you activate this option, each new session is initialized from scratch. This will have a significant impact on the response time of the server.

Example:

```
FRONTEND_OPTIONS=40
```

FRONTEND_PARAMETER

This optional configuration parameter contains additional Natural front-end parameters as specified in the Startup Parameter Area.

Value	Explanation
parameter-name	You can define multiple parameters. Each parameter specification is a pair of 8-character strings, with the first containing the parameter keyword and the second containing the parameter value.
	No default value is provided.

Example:

```
FRONTEND_PARAMETER='MSGCLASSX      '
```

The setting in this example specifies that the default output class for `CMPRINT` is `X`.

For more information, see *Operations > Natural as a Server*.

HANDLE_ABEND

If an abend occurs during server processing outside of Natural processing, Natural abend handling does not trap the abend. For this reason, the Natural Subprogram Server has its own abend recovery.

Value	Explanation
YES	Traps abends in the server processing, write a snap dump and abort the affected user. This is the default value. It is recommended to use the default value to limit the impact of an abend to a single user.
NO	Suspends the server abend handling. Any abend in the server processing terminates the complete server processing, affecting all users on that server.

Example:

```
HANDLE_ABEND=NO
```

HOST_NAME

This optional configuration parameter is necessary only if the server host supports multiple TCP/IP stacks.

Value	Explanation
host_name	If HOST_NAME is specified, the server listens on the particular stack specified by HOST_NAME, otherwise the server listens on all stacks. No default value is provided.

Example:

```
HOST_NAME=node1
```

or

```
HOST_NAME=157.189.160.55
```

IGNORE_PRESENT_SERVER

A Natural Subprogram Server allocates a so-called server environment which contains the server dependent common resources. This environment is unique for each server and relates to the Natural Subprogram Server name.

If a Natural Subprogram Server ends abnormally, it may leave the server environment in an inconsistent state, causing subsequent server restarts to fail with a NAT9913 error.

If you start a Natural Subprogram Server with IGNORE_PRESENT_SERVER=YES, it might damage an already running server which is using the same server name.

Value	Explanation
YES	Terminates the existing server environment.
NO	Aborts server initialization if a server environment already exists. This is the default value.

Example:

```
IGNORE_PRESENT_SERVER=YES
```

INITIAL_USERID

At server initialization, the Natural Subprogram Server creates a temporary Natural session to obtain the properties of the installed Natural environment.

This configuration parameter specifies the user ID to be used for this Natural session.

Value	Explanation
userid	The specified value must not exceed 8 characters, otherwise it is truncated.
SYSTEM	This is the default value.

Example:

```
INITIAL_USERID=NPIINITU
```

See also *Natural API Server > Natural Subprogram Server on z/OS > Installation > Installation of Natural Subprogram Server on z/OS > [Step 5: Clients must be defined in Natural Security](#)*.

PORT_NUMBER

This configuration parameter defines the TCP/IP port number to which the server can be connected.

Value	Explanation
1 - 65535	TCP/IP port number. No default value is provided.

Example:

```
PORT_NUMBER=3140
```

SECURITY_MODE

The Natural Subprogram Server offers a security concept that also covers the operating system resources. The client credentials are validated at the operating-system-dependent security system and the client request is executed under the client's account data.

Using the `SECURITY_MODE` parameter, you can specify whether clients run impersonated on z/OS. In this case, the client's subtask runs under the client's account.

Value	Explanation
IMPERSONATE_LOCAL	Impersonation is done within the Natural Subprogram Server environment. The client must be defined in the external security system of the Natural Subprogram Server (for example, RACF). No default value is provided.

**Notes:**

1. The `SECURITY_MODE` parameter requires Natural Security (NSC).
2. Applicable to Natural for DB2 (Batch Server only): In order to run the Natural Subprogram Server with impersonation enabled, you must have the `DSNRLI` DB2 interface module (instead of `DSNALI`) linked to the Natural nucleus.

Example:

```
SECURITY_MODE=IMPERSONATE
```

SESSION_PARAMETER

This optional configuration parameter defines dynamic session parameters that apply to all sessions running in the Natural Subprogram Server.

Value	Explanation
parameter-string	This string may extend across several lines. A plus sign (+) at the end of a string line denotes that another line follows. No default value is provided.

Example:

```
SESSION_PARAMETER='NUCNAME=NATNUCvr,' +
'FNAT=(10,930),FSEC=(10,310),' +
'FUSER=(10,32),' +
'PROFILE=(NPIPARM,18006,48),ADAMODE=1,' +
'BPI=(TYPE=NAT,SIZE=6044)'
```

where `vr` stands for the version and release number.

THREAD_NUMBER

This configuration parameter specifies the number of physical storage threads to be allocated by the Natural front-end, that is, the number of sessions that can be executed in parallel.

Value	Explanation
thread-number	Number of physical storage threads to be allocated. Note: This number does not limit the number of sessions within the server, but the number of sessions which can be in execution status concurrently. The number of sessions is limited by the size of the Natural swap medium.
3	This is the default value.

Example:

```
THREAD_NUMBER=5
```

THREAD_SIZE

This configuration parameter specifies the size of each physical storage thread which contains the Natural session data at execution time.

Value	Explanation
thread-size	Size (in KB) of each physical storage thread.
500	This is the default value.

Example:

```
THREAD_SIZE=800
```

TRACE_LEVEL

Value	Explanation
trace-level	See <i>Natural API Server > Natural Subprogram Server on z/OS > Operation > Runtime Trace Facility > Trace Level</i> .
0	This is the default value.

Example:

```
TRACE_LEVEL=0x00000011
```

or

```
TRACE_LEVEL=31+27
```

The setting in the example switches on Bits 31 and 27.

WORKER_DELAY

This configuration parameter specifies the time (in seconds) a worker waits for successive requests. See also [Worker Pool](#).

Value	Explanation
worker-delay	Value (in seconds) a worker waits for successive requests.
0	This is the default value. No worker pool is used.

Example:

```
WORKER_DELAY=10
```

Configuration File Example

```
# This is a comment
SESSION_PARAMETER= +
'profile=(npqa,10,930),' +
'fuser=(10,32)'
THREAD_NUMBER=2
THREAD_SIZE=700
FRONTEND_NAME=NATOS92L      # and another comment
PORT_NUMBER=4711
```

Natural Subprogram Server Datasets

The Natural Subprogram Server requires the following datasets:

STGCONFIG	Defines the server configuration file.
STGTRACE	The server trace output.
STGSTDO	The stdo dataset.
STGSTDE	The stde error output.

6 Operation

■ Starting the Natural Subprogram Server	36
■ Terminating the Natural Subprogram Server	36
■ Monitoring the Natural Subprogram Server	37
■ Runtime Trace Facility	38

This document describes how to operate a Natural Subprogram Server under z/OS (Batch).

This document covers the following topics:

Starting the Natural Subprogram Server

The Natural Subprogram Server can be started as a "started task":

```
//NPISRV  PROC
//KSPSRV  EXEC PGM=NATRNPI,REGION=4000K,TIME=1440,
//  PARM=('POSIX(ON)/NPISRV1')
//STEPLIB DD DISP=SHR,DSN=NP1vrs.LOAD
//        DD DISP=SHR,DSN=NATvrs.LOAD
//CMPRINT DD SYSOUT=X
//STGCONF DD DISP=SHR,DSN=NP1vrs.CONFIG(SRV1)
//STGTRACE DD SYSOUT=X
//STGSTDO DD SYSOUT=X
//STGSTDE DD SYSOUT=X
```

where `vrs` is the version, release, and system maintenance level number of the Natural Subprogram Server.

For the currently applicable versions, refer to <https://empower.softwareag.com/>.



Notes:

1. `PARM=('POSIX(ON)/NPISRV1')` - `POSIX(ON)` is required for a proper LE370 initialization, with `NPISRV1` being the server name. This server name is used for referencing server-related resources.
2. The name of the started task must be defined under RACF and the z/OS Linux System Services.

Terminating the Natural Subprogram Server

The Natural Subprogram Server can be terminated using z/OS operator commands.

`STOP NPISRV` stops the server with the started task name `NPISRV`.

`F NPISRV,APPL=TERMINATE` sends the `TERMINATE` command to the started task. This also stops the Natural Subprogram Server. For more information, see the [Monitor Communication](#) section.

Active Natural Sessions are canceled during the Natural Subprogram Server shutdown.

Monitoring the Natural Subprogram Server

To enable the administrator to monitor the status of the Natural Subprogram Server, a monitor task is provided and is automatically initialized at server startup. Using the monitor commands described below, the administrator can control the server activities, cancel particular user sessions, terminate the entire server, etc.

This document covers the following topics:

- [Monitor Communication](#)
- [Monitor Commands](#)

Monitor Communication

To communicate with the monitor, you can use the `MODIFY` operator command. The commands in the [Monitor Commands](#) section can be used to execute the monitor commands. The output of the executed monitor command will be written to the system log.

Example:

```
F jobname,APPL=ping
```

sends the `ping` command to the Natural Subprogram Server running under the `jobname` job.

Monitor Commands

The Natural Subprogram Server supports the following monitor commands:

Monitor Command	Explanation
<code>ping</code>	Verifies whether the server is active. The server responds and sends the string <code>I'm still up</code>
<code>terminate</code>	Terminates the server.
<code>list sessions</code>	Returns a list of active Natural sessions within the server. For each session, the server returns information about the user who owns the session, the session initialization time, the last activity time, and an internal session identifier (<code>session-id</code>).
<code>help</code>	Returns help information about the supported monitor commands.

Runtime Trace Facility

For debugging purposes, the server code has a built-in trace facility which can be switched on.

This document covers the following topics:

- [Trace Medium](#)
- [Trace Configuration](#)
- [Trace Level](#)

Trace Medium

A remote Natural Subprogram Server writes its runtime trace to the logical system file `SYSOUT` of the `FSIO` task.

Trace Configuration

The trace is configured by a trace level, which defines the details of the trace. See also [Trace Level](#).

Every Natural session is provided with a 32-bit trace status word (TSW) which defines the trace level for this session. The value of the TSW is set in the `TRACE_LEVEL` configuration parameter. The trace is switched off if the value is zero.

Trace Level

Each bit of the TSW is responsible for certain trace information, starting with the rightmost bit:

Bit 31	Trace main events (server initialization/termination, client request/result).
Bit 30	Detailed functions (session allocation, detailed request processing).
Bit 29	Dump internal storage areas.
Bit 28	Session directory access.
Bit 27	Free.
Bit 26	Free.
Bit 25	Free.
Bit 24	Free.
Bit 23	Callback requests (Natural runtime calls front-end stub).
Bit 22	Callback memory functions (Natural runtime calls front-end stub for memory allocation).
Bit 21	Trace NSC Logon processing.
Bit 20	Worker pool events (add/delete/post worker).
Bit 19	Worker pool content traced at each add/delete worker.
Bit 18-00	Free.

7 Security

■ Natural Security	40
■ External Security	40

This document describes the security environment of a Natural Subprogram Server under z/OS (Batch).

This document covers the following topics:

Natural Security

Each request contains the user identification and the user password. The Natural Subprogram Server performs a LOGON with these credentials. Natural security validates the user-ID and the password. All restrictions set for this user in Natural Security take effect.

From the operating system's perspective, the user operates anonymously under the account of the Natural Subprogram Server.

External Security

The Natural Subprogram Server can be configured to run with impersonation. For more information, see *Natural API Server > Natural Subprogram Server on z/OS > Configuration > Configuration Parameters > [SECURITY-MODE](#)*.

If the server is configured to run with impersonation, every client will be validated by the z/OS security system. The z/OS subtask of the client runs under the account of the client. All restrictions set for this user in RACF will take effect. Every client must be defined in the z/OS security system. For more information, see *Natural API Server > Natural Subprogram Server on z/OS > Configuration > [External Security Configuration](#)*.

To use impersonation, Natural Security (NSC) is a prerequisite. If a user is validated by the z/OS security system, the user is accepted as a “trusted” user by NSC. That is, the user must be defined in NSC but no password is validated during logon. So even with impersonation, all restrictions set for this user in Natural Security will take effect.

III

Natural OpenAPI Server on Tomcat

This document covers the following topics:

Installation
Configuration
Operation
Monitoring
Security

8 Installation

■ Installing the Natural OpenAPI Server	44
---	----

Instructions on how to install the Natural OpenAPI Server.

Installing the Natural OpenAPI Server

➤ To install the Natural OpenAPI Server

- 1 Open the Software AG Installer and navigate to the installation tree.
- 2 Expand the **Natural** node and select **Natural OpenAPI Server**.
- 3 Follow the onscreen instructions to complete the installation.

9 Configuration

■ General Configuration	46
■ Natural Error to HTTP Mapping	47

This document covers the following topics:

General Configuration

The following configurations have to be done before running the Natural OpenAPI server.

In the *application.properties* file, define the following properties:

Parameter	Description
<code>com.softwareag.natural.openapi.server.config.host.address</code>	The IP address of the Natural Subprogram Server.
<code>com.softwareag.natural.openapi.server.config.host.port</code>	The port to connect to the Natural Subprogram Server.
<code>natural.integration.services.files</code>	A list of JSON files that should be accessed by the Natural OpenAPI Server (full file name and path). The files should be separated by comma. For examples, <code>/opt/softwareag/NaturalOpenAPIServer/services/naturalIntegrationServicesFiles1.json</code> <code>/opt/softwareag/NaturalOpenAPIServer/services/naturalIntegrationServicesFiles2.json</code>
<code>natural.integration.services.directory</code>	The location of folder that contains all the JSON definition files.
<code>com.softwareag.natural.openapi.server.config.host.ssl.enabled</code>	Define whether to use SSL or not.
<code>com.softwareag.natural.openapi.server.config.host.ssl.certsChain[0]</code>	Location of the public certificate files to be used by the Natural Subprogram Server using SSL.
<code>com.softwareag.natural.openapi.web.config.authenticationType</code>	The Authentication used to connect to the Natural Subprogram Server.
<code>com.softwareag.natural.openapi.web.config.oAuth2Config.tokenUrl</code>	The token URL of the credentials provider. The token URL is required for Bearer token when testing with the Swagger client.
<code>com.softwareag.natural.openapi.web.config.oAuth2Config.scopes</code>	The scopes to request from the credentials provider. The scopes are required for the Bearer token when testing with the Swagger client.
<code>com.softwareag.natural.openapi.server.config.host.charset</code>	The charset to use when encoding the response of the Natural Subprogram Server.
<code>management.endpoints.web.exposure.include</code>	This parameter controls the exposure of the service endpoint. If not defined, this parameter is added with the value <code>reloadService</code> . If <code>reloadService</code> is defined, append <code>reloadServices</code> to the list of comma-separated values.



Notes:

1. You can define a file list in the `natural.integration.services.files` property and define another folder in the `natural.integration.services.directory` property. The application will use the resources from both locations.
2. All the JSON files created by the Natural OpenAPI Eclipse Plug-In should be copied to the service folder (defined in the `application.properties` file by the `natural.integration.services.directory` property, or they should be defined in the `natural.integration.services.files` property).

Natural Error to HTTP Mapping

The Natural OpenAPI Server can map backend Natural error codes to HTTP status codes.

The mapping resolution order is:

1. Exact match (for example, NAT3009)
2. Matching numeric range (for example, 0..99)
3. Global default mapping

Default mappings are predefined in the Natural OpenAPI Server.

You can override or extend mappings in the `application.properties` file using the `error-mapping.*` prefix.

Example: Override one Natural specific code

```
error-mapping.sources.natural.specific.NAT3009.http-code=422
error-mapping.sources.natural.specific.NAT3009.http-message=Unprocessable Entity
error-mapping.sources.natural.specific.NAT3009.category=VALIDATION_ERROR
error-mapping.sources.natural.specific.NAT3009.log-level=WARN
```

Example: Define a Natural range mapping

```
error-mapping.sources.natural.ranges.syntax.from=0
error-mapping.sources.natural.ranges.syntax.to=49
error-mapping.sources.natural.ranges.syntax.http-code=400
error-mapping.sources.natural.ranges.syntax.http-message=Bad Request
error-mapping.sources.natural.ranges.syntax.category=SYNTAX_ERROR
error-mapping.sources.natural.ranges.syntax.log-level=WARN
```

Example: Override global fallback

```
error-mapping.default-mapping.http-code=500
error-mapping.default-mapping.http-message=Internal Server Error
error-mapping.default-mapping.category=SYSTEM_ERROR
error-mapping.default-mapping.log-level=ERROR
```

In addition to Natural code mappings, transport failures are mapped by the WebFlux error handler:

- Backend connection failure: 502 Bad Gateway
- Backend timeout: 504 Gateway Timeout
- Unknown/unmapped errors: 500 Internal Server Error

10 Operation

■ Deploying Web Application with Tomcat	50
■ Deploying OpenAPI Resources	50

This document covers the following topics:

Deploying Web Application with Tomcat

The following are some specific configurations for Apache Tomcat® if you want to deploy the Natural OpenAPI Web application with your own Tomcat.

- [Deploying War File](#)
- [Configuring Server.XML](#)

Deploying War File

The *NaturalOpenAPIServer.war* must be copied to the */tomcat/webapps/* directory.

In addition, the `SPRING_CONFIG_LOCATION` environment variable must be configured to point to the location of the *application.properties* file.

Configuring Server.XML

Alternatively, the following configuration can be added to the Tomcat's *server.xml* file when deploying the Natural OpenAPI Server.

```
<Context path="/noa1" docBase="noa1" override="true">
    <Environment name="SPRING_CONFIG_LOCATION" ↵
value="/your-path/conf/application.properties" type="java.lang.String" />
</Context>
```

Deploying OpenAPI Resources

To deploy the services described in the *Developer Overview* > [Generating OpenAPI Resources](#) section, the Natural OpenAPI Server must be configured with the location of the service files.

The service file location can be configured in the following ways:

- Using `natural.integration.services.directory` – Set this property in the *application.properties* file to specify the directory containing the service files. Place the service files in the specified directory.
- Using `natural.integration.services.files` – Set this property to specify the names of the specific service files to be deployed. The specified files must be located in the classpath of the application.

When the Natural OpenAPI Server starts, it loads the configured service files and registers the OpenAPI services defined in those files.

Each service file contains the definition and mapping information required to expose a Natural subprogram as a REST API service. Once the server has successfully started, all services defined in the loaded service files are available to clients through the Natural OpenAPI Server. For more information, see the `management.endpoints.web.exposure.include` parameter in the [General Configuration](#) section.

Adding or modifying a service therefore requires updating the corresponding service file and restarting the server or using the reload actuator endpoint for the changes to be loaded and the services to become available.

11

Monitoring

OpenTelemetry is an open standard for collecting and managing observability data from applications and services. It provides a common framework that helps monitor and analyze the behavior, performance, and health of a system.

Our product uses the three main types of telemetry data defined by OpenTelemetry:

- **Traces** – Provides information about the flow of a request through the system. Traces help follow an operation across different components and identify where time is spent or where failures occur.
- **Metrics** – Provides numerical measurements about the system, such as resource usage, request counts, response times, and other performance-related values. Metrics can be used to monitor the overall behavior and health of the application.
- **Logs** – Provides detailed records of events that occur during application execution. Logs can help investigate errors, warnings, and other relevant runtime events.

By collecting traces, metrics, and logs, OpenTelemetry provides a more complete view of the system and makes it easier to monitor application behavior, identify performance issues, and troubleshoot problems.

12

Security

■ Secure Communication between Natural OpenAPI Server and Subprogram Server	56
■ Secure Communication between Client and Natural OpenAPI Server	56
■ Authentication	57

The Natural OpenAPI Server provides configuration options to secure communication:

- between the Natural OpenAPI server and the Subprogram Server.
- between the client and the Natural OpenAPI Server.

This document covers the following topics:

Secure Communication between Natural OpenAPI Server and Subprogram Server

The Natural OpenAPI Server can use SSL/TLS to encrypt communication between the Natural OpenAPI Server and the subprogram servers.

SSL/TLS can be enabled using the following property:

```
com.softwareag.natural.openapi.server.config.host.ssl.enabled=true
```

When SSL/TLS is enabled, the server uses the configured certificate chain to establish secure connections. The certificate can be configured using the `certsChain` property:

```
com.softwareag.natural.openapi.server.config.host.ssl.certsChain[0]=classpath:natsubsrv.server.cert.crt
```

The property specifies the location of the server certificate. The `classpath:` prefix indicates that the certificate is loaded from the application's classpath.

Multiple certificates can be specified as part of the certificate chain by using additional indexed properties, for example, `certsChain[1]`, `certsChain[2]`, and so on.



Note: The SSL certificate configuration is relevant only when

`com.softwareag.natural.openapi.server.config.host.ssl.enabled` is set to `true`.

Secure Communication between Client and Natural OpenAPI Server

The Natural OpenAPI Server supports secure communication using SSL/TLS. When SSL/TLS is enabled, clients connect to the server using HTTPS instead of HTTP.

The default port used by the server is 8080. A different port can be configured for HTTPS, for example, 8443, by setting the following property:

```
server.port=8443
```

To enable SSL/TLS, set the following property:

```
server.ssl.enabled=true
```

The following properties configure the SSL certificate and key store used by the server:

Key Store Path

This specifies the path to the key store containing the SSL certificate and private key. For example:

```
server.ssl.key-store=file:///opt/softwareag/NaturalOpenAPIServer/conf/keystore.p12
```

Key Store Password

This specifies the password required to access the key store. For example:

```
server.ssl.key-store-password=password
```

Key Store Type

This specifies the type of key store. The supported types are JKS and PKCS12. For example:

```
server.ssl.key-store-type=PKCS12
```

Key Store Alias

This specifies the alias used to identify the certificate and private key within the key store. For example:

```
server.ssl.key-alias=noa-server
```



Note: The key store must contain a valid SSL/TLS certificate and its corresponding private key.

Authentication

The Natural OpenAPI server does not authenticate the client sending the request by itself, but rely on the authentication module of the Natural Subprogram Server. However, if the Natural Subprogram Server is configured to use an authentication method, you must define the appropriate configuration in the Natural OpenAPI Server. Defining the correct configuration will create the related definition in the OpenAPI document located at http://localhost:OpenAPIServer_port/v3/api-docs.

There are 3 different ways to define the authentication:

Using Basic Authentication

Setting the `com.softwareag.natural.openapi.web.config.authenticationType` property to `basic` will require the client application to send username and password encoded with Base64 encoding (defined in RFC-7617).

Using OAuth2 Token

Token retrieval is the responsibility of the client application, while token validation is performed by the Natural Subprogram Server authentication module.

Setting the `com.softwareag.natural.openapi.web.config.authenticationType` property to `oauth2` adds a requirement to the OpenAPI document for the client application to send a Bearer token.

The client application is required to include the special HTTP header *X-Impersonate-User* to specify the user to impersonate, alongside the provided token.

The following configuration can be used when testing with the Swagger UI web client. The Swagger UI embedded in the Natural OpenAPI Server currently supports only the Client Credentials flow. When OAuth2 authentication is enabled, you can define the `tokenURL` and required scopes. These values are exposed in the OpenAPI document but are not used by the Natural OpenAPI Server for authentication. In the case of impersonation, it must be performed after a successful token verification.

Using Anonymous Requests

When an authentication type is not defined in the *application.properties* file, the requests are considered anonymous by default and should not be authenticated. It is also possible to execute the request as another user using the special HTTP header *X-Impersonate-User*.

IV

Natural OpenAPI Server on Liberty

This document covers the following topics:

Installation
Configuration
Operation
Monitoring
Security

13

Installation

■ Installing the Natural OpenAPI Server	62
---	----

This document covers the following topics:

Installing the Natural OpenAPI Server

➤ To install the Natural OpenAPI Server

- 1 Open the Software AG Installer and navigate to the installation tree.
- 2 Expand the **Natural** node and select **Natural OpenAPI Server**.
- 3 Follow the onscreen instructions to complete the installation.

14

Configuration

■ General Configuration	64
■ Natural Error to HTTP Mapping	65

This document covers the following topics:

General Configuration

The following configurations have to be done before running the Natural OpenAPI server.

In the *application.properties* file, define the following properties:

Parameter	Description
<code>com.softwareag.natural.openapi.server.config.host.address</code>	The IP address of the Natural Subprogram Server.
<code>com.softwareag.natural.openapi.server.config.host.port</code>	The port to connect to the Natural Subprogram Server.
<code>natural.integration.services.files</code>	A list of JSON files that should be accessed by the Natural OpenAPI Server (full file name and path). Separate the files with a comma. For examples, <code>/opt/softwareag/NaturalOpenAPIServer/services/naturalIntegrationServices.json</code> <code>/opt/softwareag/NaturalOpenAPIServer/services/naturalIntegrationServices2.json</code>
<code>natural.integration.services.directory</code>	The location of folder that contains all the JSON definition files.
<code>com.softwareag.natural.openapi.server.config.host.ssl.enabled</code>	Define whether to use SSL or not.
<code>com.softwareag.natural.openapi.server.config.host.ssl.certsChain[0]</code>	Location of the public certificate files to be used by the Natural Subprogram Server using SSL.
<code>com.softwareag.natural.openapi.web.config.authenticationType</code>	The Authentication used to connect to the Natural Subprogram Server.
<code>com.softwareag.natural.openapi.web.config.oAuth2Config.tokenUrl</code>	The token URL of the credentials provider. The token is a Bearer token when testing with the Swagger client.
<code>com.softwareag.natural.openapi.web.config.oAuth2Config.scopes</code>	The scopes to request from the credentials provider. The token is a Bearer token when testing with the Swagger client.
<code>com.softwareag.natural.openapi.server.config.host.charset</code>	The charset to use when encoding the request and response of the Natural Subprogram Server.
<code>management.endpoints.web.exposure.include</code>	This parameter controls the exposure of the management service endpoint. If not defined, this parameter is added with the value <code>reloadService</code> . If <code>reloadService</code> is defined, append <code>reloadServices</code> to the list of comma-separated values.



Notes:

1. You can define a file list in the `natural.integration.services.files` property and define another folder in the `natural.integration.services.directory` property. The application will use the resources from both locations.
2. All the JSON files created by the Natural OpenAPI Eclipse Plug-In should be copied to the service folder (defined in the `application.properties` file by the `natural.integration.services.directory` property, or they should be defined in the `natural.integration.services.files` property).

Natural Error to HTTP Mapping

The Natural OpenAPI Server can map backend Natural error codes to HTTP status codes.

The mapping resolution order is:

1. Exact match (for example, NAT3009)
2. Matching numeric range (for example, 0..99)
3. Global default mapping

Default mappings are predefined in the Natural OpenAPI Server.

You can override or extend mappings in the `application.properties` file using the `error-mapping.*` prefix.

Example: Override one Natural specific code

```
error-mapping.sources.natural.specific.NAT3009.http-code=422
error-mapping.sources.natural.specific.NAT3009.http-message=Unprocessable Entity
error-mapping.sources.natural.specific.NAT3009.category=VALIDATION_ERROR
error-mapping.sources.natural.specific.NAT3009.log-level=WARN
```

Example: Define a Natural range mapping

```
error-mapping.sources.natural.ranges.syntax.from=0
error-mapping.sources.natural.ranges.syntax.to=49
error-mapping.sources.natural.ranges.syntax.http-code=400
error-mapping.sources.natural.ranges.syntax.http-message=Bad Request
error-mapping.sources.natural.ranges.syntax.category=SYNTAX_ERROR
error-mapping.sources.natural.ranges.syntax.log-level=WARN
```

Example: Override global fallback

```
error-mapping.default-mapping.http-code=500
error-mapping.default-mapping.http-message=Internal Server Error
error-mapping.default-mapping.category=SYSTEM_ERROR
error-mapping.default-mapping.log-level=ERROR
```

In addition to Natural code mappings, transport failures are mapped by the WebFlux error handler:

- Backend connection failure: 502 Bad Gateway
- Backend timeout: 504 Gateway Timeout
- Unknown/unmapped errors: 500 Internal Server Error

15 Operation

■ Deploying Web Application with Liberty	68
■ Deploying OpenAPI Resources	69

This document covers the following topics:

Deploying Web Application with Liberty

The following are some specific configurations for Liberty if you want to deploy the Natural OpenAPI Web application with your own Liberty.

Deploying War File

The *NaturalOpenAPIServer.war* must be copied to the */config/apps/* directory.

Configuring Server.XML

The following configuration should be added to the Liberty *server.xml* file to deploy the Natural OpenAPI Server.

In addition, the `SPRING_CONFIG_LOCATION` environment variable must be configured to point to the location of the *application.properties* file.

```
<?xml version="1.0" encoding="UTF-8"?>
    <server description="Natural _OpenAPI">

        <featureManager>
            <feature>servlet-6.0</feature>
            <feature>mpConfig-3.1</feature>
        </featureManager>

        <httpEndpoint id="defaultHttpEndpoint" httpPort="9080" ↵
httpsPort="9443"/>

        <!--
        appProperties is Liberty's per-application config mechanism: values ↵
here
        are scoped strictly to the <webApplication> that declares them and are
exposed through MicroProfile Config - MpConfigEnvironmentInitializer ↵
in
        the app reads them from there.
        -->
        <webApplication id="app1" name="app1" ↵
location="/config/apps/NaturalOpenAPIServer.war" contextRoot="/app1">
            <appProperties>
                <property name="SPRING_CONFIG_LOCATION" ↵
value="file:///config/application.properties"/>
            </appProperties>
        </webApplication>
    </server>
```

Deploying OpenAPI Resources

To deploy the services described in the *Developer Overview* > [Generating OpenAPI Resources](#) section, the Natural OpenAPI Server must be configured with the location of the service files.

The service file location can be configured in the following ways:

- Using `natural.integration.services.directory` – Set this property in the *application.properties* file to specify the directory containing the service files. Place the service files in the specified directory.
- Using `natural.integration.services.files` – Set this property to specify the names of the specific service files to be deployed. The specified files must be located in the classpath of the application.

When the Natural OpenAPI Server starts, it loads the configured service files and registers the OpenAPI services defined in those files.

Each service file contains the definition and mapping information required to expose a Natural subprogram as a REST API service. Once the server has successfully started, all services defined in the loaded service files are available to clients through the Natural OpenAPI Server. For more information, see the `management.endpoints.web.exposure.include` parameter in the [General Configuration](#) section.

Adding or modifying a service therefore requires updating the corresponding service file and re-starting the server or using the reload actuator endpoint for the changes to be loaded and the services to become available.

16

Monitoring

OpenTelemetry is an open standard for collecting and managing observability data from applications and services. It provides a common framework that helps monitor and analyze the behavior, performance, and health of a system.

Our product uses the three main types of telemetry data defined by OpenTelemetry:

- **Traces** – Provides information about the flow of a request through the system. Traces help follow an operation across different components and identify where time is spent or where failures occur.
- **Metrics** – Provides numerical measurements about the system, such as resource usage, request counts, response times, and other performance-related values. Metrics can be used to monitor the overall behavior and health of the application.
- **Logs** – Provides detailed records of events that occur during application execution. Logs can help investigate errors, warnings, and other relevant runtime events.

By collecting traces, metrics, and logs, OpenTelemetry provides a more complete view of the system and makes it easier to monitor application behavior, identify performance issues, and troubleshoot problems.

17

Security

■ Secure Communication between Natural OpenAPI Server and Subprogram Server	74
■ Secure Communication between Client and Natural OpenAPI Server	74
■ Authentication	75

The Natural OpenAPI Server provides configuration options to secure communication:

- between the Natural OpenAPI server and the Subprogram Server.
- between the client and the Natural OpenAPI Server.

This document covers the following topics:

Secure Communication between Natural OpenAPI Server and Subprogram Server

The Natural OpenAPI Server can use SSL/TLS to encrypt communication between the Natural OpenAPI Server and the subprogram servers.

SSL/TLS can be enabled using the following property:

```
com.softwareag.natural.openapi.server.config.host.ssl.enabled=true
```

When SSL/TLS is enabled, the server uses the configured certificate chain to establish secure connections. The certificate can be configured using the `certsChain` property:

```
com.softwareag.natural.openapi.server.config.host.ssl.certsChain[0]=classpath:natsubsrv.server.cert.crt
```

The property specifies the location of the server certificate. The `classpath:` prefix indicates that the certificate is loaded from the application's classpath.

Multiple certificates can be specified as part of the certificate chain by using additional indexed properties, for example, `certsChain[1]`, `certsChain[2]`, and so on.



Note: The SSL certificate configuration is relevant only when

`com.softwareag.natural.openapi.server.config.host.ssl.enabled` is set to `true`.

Secure Communication between Client and Natural OpenAPI Server

The Natural OpenAPI Server supports secure communication using SSL/TLS. When SSL/TLS is enabled, clients connect to the server using HTTPS instead of HTTP.

The default port used by the server is 8080. A different port can be configured for HTTPS, for example, 8443, by setting the following property:

```
server.port=8443
```

To enable SSL/TLS, set the following property:

```
server.ssl.enabled=true
```

The following properties configure the SSL certificate and key store used by the server:

Key Store Path

This specifies the path to the key store containing the SSL certificate and private key. For example:

```
server.ssl.key-store=file:///opt/softwareag/NaturalOpenAPIServer/conf/keystore.p12
```

Key Store Password

This specifies the password required to access the key store. For example:

```
server.ssl.key-store-password=password
```

Key Store Type

This specifies the type of key store. The supported types are JKS and PKCS12. For example:

```
server.ssl.key-store-type=PKCS12
```

Key Store Alias

This specifies the alias used to identify the certificate and private key within the key store. For example:

```
server.ssl.key-alias=noa-server
```



Note: The key store must contain a valid SSL/TLS certificate and its corresponding private key.

Authentication

The Natural OpenAPI server does not authenticate the client sending the request by itself, but rely on the authentication module of the Natural Subprogram Server. However, if the Natural Subprogram Server is configured to use an authentication method, you must define the appropriate configuration in the Natural OpenAPI Server. Defining the correct configuration will create the related definition in the OpenAPI document located at http://localhost:OpenAPIServer_port/v3/api-docs.

There are 3 different ways to define the authentication:

- [Using Basic Authentication](#)
- [Using OAuth2 Token](#)
- [Using Anonymous Requests](#)

Using Basic Authentication

Setting the `com.softwareag.natural.openapi.web.config.authenticationType` property to `basic` will require the client application to send username and password encoded with Base64 encoding (defined in RFC-7617).

Using OAuth2 Token

Token retrieval is the responsibility of the client application, while token validation is performed by the Natural Subprogram Server authentication module.

Setting the `com.softwareag.natural.openapi.web.config.authenticationType` property to `oauth2` adds a requirement to the OpenAPI document for the client application to send a Bearer token.

The client application is required to include the special HTTP header *X-Impersonate-User* to specify the user to impersonate, alongside the provided token.

The following configuration can be used when testing with the Swagger UI web client. The Swagger UI embedded in the Natural OpenAPI Server currently supports only the Client Credentials flow. When OAuth2 authentication is enabled, you can define the `tokenURL` and required scopes. These values are exposed in the OpenAPI document but are not used by the Natural OpenAPI Server for authentication. In the case of impersonation, it must be performed after a successful token verification.

Using Anonymous Requests

When an authentication type is not defined in the *application.properties* file, the requests are considered anonymous by default and should not be authenticated. It is also possible to execute the request as another user using the special HTTP header *X-Impersonate-User*.

V Developer Overview

This document covers the following topic:

Installation of the NaturalONE OpenAPI Plugin
Configuration of the NaturalONE OpenAPI Plugin
Generate and Deploy Services
Special Considerations

18

Installation of the NaturalONE OpenAPI Plugin

The NaturalONE OpenAPI Plugin is required to create and manage Natural OpenAPI services.

The plugin runs in the NaturalONE IDE. If NaturalONE is not already installed, install it when installing the NaturalONE OpenAPI Plugin.

➤ To install the NaturalONE OpenAPI Plugin

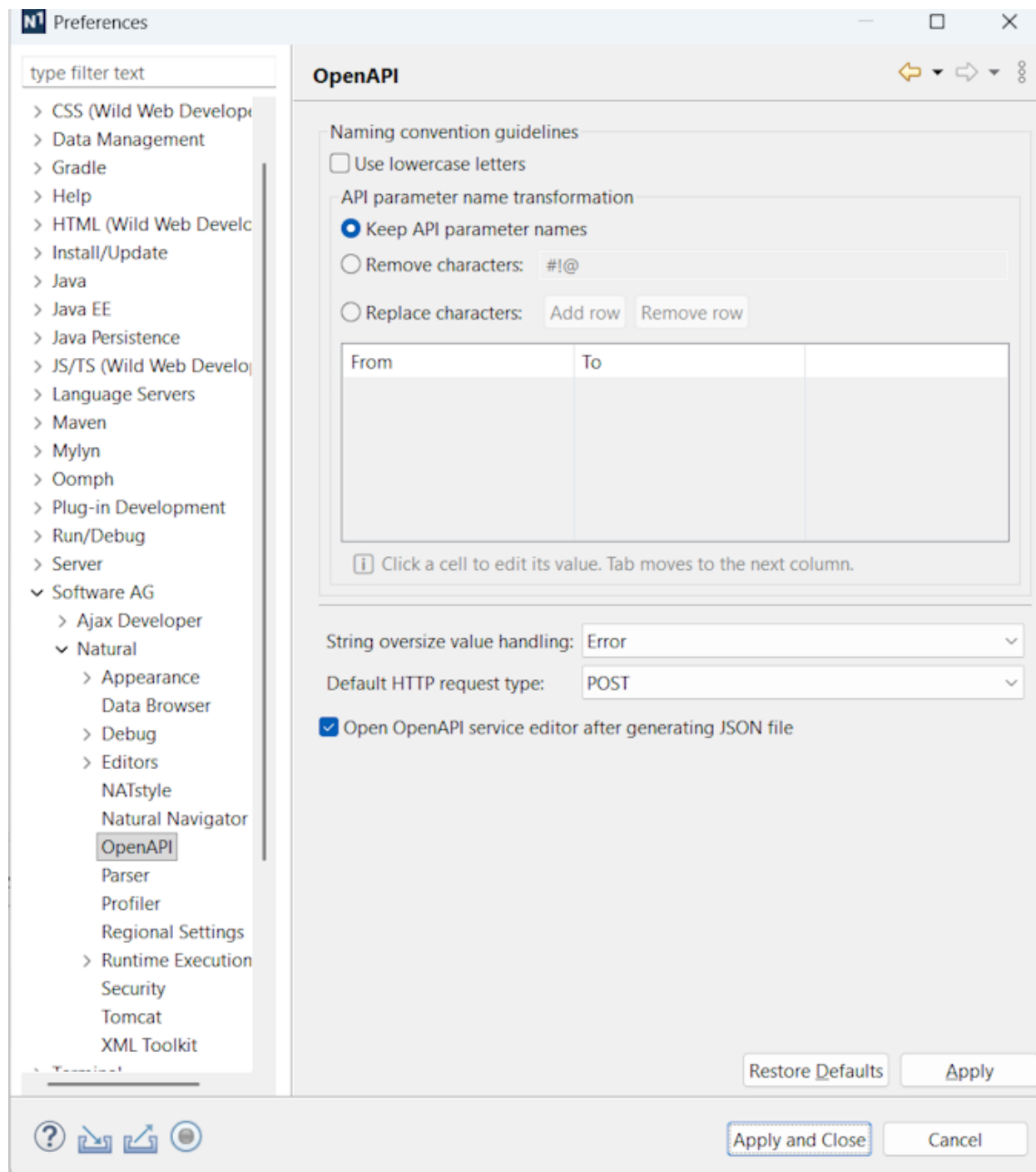
- 1 Open the Software AG Installer and navigate to the installation tree.
- 2 Expand the **NaturalONE** node and select **OpenAPI Development**.

19

Configuration of the NaturalONE OpenAPI Plugin

The global default settings for OpenAPI service generation can be configured in the Eclipse **Preferences** window. These settings apply to all service files generated by the plugin.

To access these settings, go to **Window** on the main menu and select **Preferences**. On the navigation tree, expand the **Software AG** node, **Natural** node, and then select **OpenAPI**.



The settings in the **Preferences** window define the default values used during OpenAPI service generation when a corresponding property has not been explicitly defined in the subprogram itself.

The following settings are available:

Use lowercase letters

Select this checkbox to automatically convert parameter names to lowercase during the OpenAPI service generation process.

API parameter name transformation

This setting provides options to automatically transform OpenAPI parameter names. The following options are available:

- **Keep API parameter names** – This option keeps the parameter names unchanged.
- **Remove characters** – This option removes specified characters from parameter names. An accompanying input field allows you to specify the characters to remove (for example, #, !, or @).
- **Replace characters** – This option allows you to define mappings that specify which characters or strings to replace and the characters or strings to use as replacements instead of removing them.

String oversize value handling

This setting defines the default behavior for handling oversized string values. This setting can be overridden for an individual service in the OpenAPI Editor.

The following options are available:

- **Truncate** – This option truncates the value when it exceeds the allowed size.
- **Error** – This option generates an error when the value exceeds the allowed size.

Default HTTP request type

This setting defines the default HTTP request method that is selected when the OpenAPI Editor is opened, if no HTTP request type was defined in the subprogram.

Open OpenAPI service editor after generating JSON file

This setting controls whether the OpenAPI Editor opens automatically when `Generate OpenAPI Service` is triggered.

By default, this checkbox is selected. When it is cleared, the service file is generated immediately without opening the editor.

When multiple subprograms are selected and `Generate OpenAPI Service` is triggered, this setting is bypassed and the OpenAPI Editor is not opened.

20

Generate and Deploy Services

■ Generating OpenAPI Resources	86
■ Viewing OpenAPI Resources via Editor	87

This document covers the following topics:

Generating OpenAPI Resources

Creating OpenAPI Services

To create OpenAPI services, the Natural OpenAPI Server needs to read the definition of a Natural subprogram. This information is modeled and stored in a service JSON file.

Each service JSON file represents a single Natural subprogram and contains the following information:

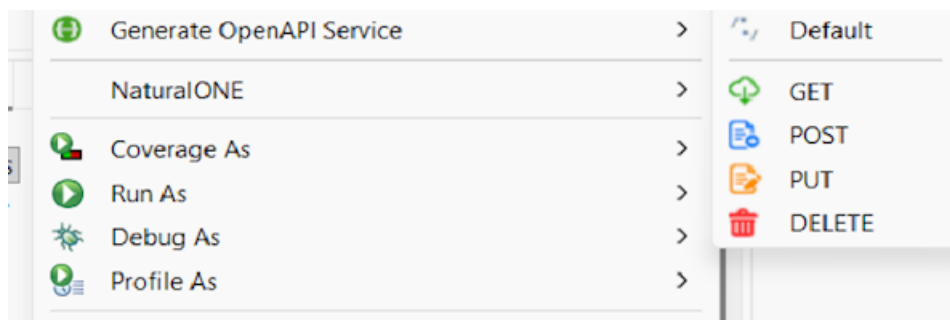
- Natural subprogram and library names
- OpenAPI schema mappings between OpenAPI parameters and Natural input and output parameters
- Query parameters that can also be mapped to Natural parameters



Note: Query parameters currently do not support array mappings.

➤ To create an OpenAPI service

- 1 Right-click the desired subprogram.
- 2 Select **Generate OpenAPI Service**. The following menu is displayed with these options:

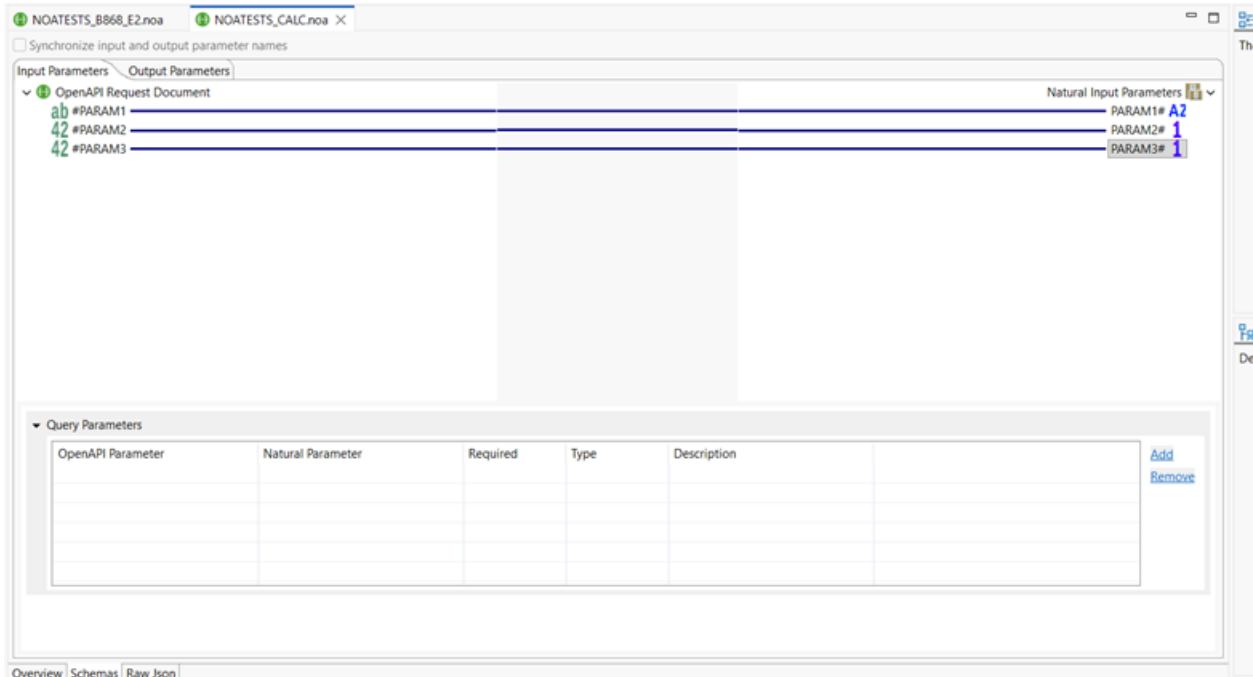


- **Default** – Uses the HTTP method defined in the subprogram. If no HTTP method is defined in the subprogram, the value configured in **Preferences** is used.
- **GET / POST / PUT / DELETE** – Selecting a specific HTTP method overrides the default method and defines the method that the service will use.



Note: Array parameters currently do not support the **GET** and **DELETE** HTTP methods.

Viewing OpenAPI Resources via Editor



The OpenAPI Service Editor displays information and configuration settings of the services. It consists of three tabs:

- Overview Tab
- Schemas Tab
- Raw Json Tab

Overview Tab

The **Overview** tab displays general information and configuration settings for the service.

This tab also allows you to define additional general information, such as summary, description, and tags.

HTTP Request Type

Indicates the HTTP method used by the service. The supported methods are **GET**, **POST**, **PUT**, and **DELETE**.

Behavior When a Value Is Too Large

Defines how the service handles an incoming OpenAPI value that exceeds the size of the corresponding mapped Natural parameter.

This setting can be configured for each individual service in the editor. The default value is based on the option selected at *String oversize value handling* in the Eclipse **Preferences** window.

Schemas Tab

This tab consists of two subtabs: **Input Parameters** and **Output Parameters**. The names of the properties on these two tabs can be synchronized by selecting the **Synchronize input and output parameter names** checkbox.

Input Parameters Tab

This tab displays the OpenAPI Schema layout for the Natural input parameters. The layout provides a mapping interface between Natural input parameters and OpenAPI request document.

The OpenAPI Schema layout consists of three areas and one section:

1. **OpenAPI Request Document** (left tree area): Displays the OpenAPI document. Right-clicking an OpenAPI element in the tree displays the following options:
 - **Rename** – Changes the name of the OpenAPI element.
 - **Restore Mapping** – Restores a mapping that was previously removed.
 - **Delete** – Deletes the OpenAPI parameter if it is not currently mapped.
2. Mapping Controls area: Provides the option to remove a mapping. Removing a mapping means that the OpenAPI element is no longer used. The corresponding Natural parameter receives its default value (for example, an empty string for alphanumeric parameters or 0 for numeric parameters).
3. **Natural Input Parameters** (right tree area): Displays the Natural parameters. Right-clicking a Natural parameter displays the following options:
 - **Map to Document** – Creates or restores a mapping to an element of the OpenAPI document.
 - **Map to Query Parameter** – Creates a query parameter mapped to the Natural parameter (enabled only if the Natural parameter is not mapped).
4. **Query Parameters** section: Allows you to add and/or remove mappings between query parameters and Natural input parameters. Query parameters are passed as part of the URL and are not included in the REST request body.

Output Parameters Tab

This tab displays the OpenAPI Schema layout for the Natural output parameters. The layout provides a mapping interface between Natural output parameters and OpenAPI response document.

Raw Json Tab

This tab displays the JSON file of the service.

21

Special Considerations

■ Developing Natural Subprograms for Natsubsvr	92
--	----

This section covers topics of relevance for developers.

Developing Natural Subprograms for Natsubsrv

This document lists the differences when a Natural subprogram is executed by the Natural Subprogram Server (Natsubsrv) instead of by a terminal session.

It is not an introduction to Natural programming — for that, see the Natural documentation.

These differences are caused by two properties:

- No terminal. The session is a server session; there is no screen and no keyboard.
- No state between calls. One session serves the requests of many end users, one after another, and is reset at every request boundary.

This document covers the following topics:

- [Execution Model](#)
- [Mapping OpenAPI Names](#)
- [Restricted Statements](#)
- [Statelessness](#)
- [Transactions and Work Files](#)
- [Error Handling](#)
- [System Variables](#)
- [Checklist](#)

Execution Model

- The subprogram is entered by a `CALLNAT` and leaves by its normal end or by `ESCAPE ROUTINE`.
- The `DEFINE DATA PARAMETER` block is the only data channel into and out of the subprogram.
- Every request names its own library and its own user; both are established by the server before the `CALLNAT`.
- The session is reset at the request boundary, so every call starts from a defined state and leaves nothing behind.

Mapping OpenAPI Names

In a subprogram, you can define the name of the API parameter that will be mapped to a Natural parameter.

This is done by adding the prefix `oapi.n:` followed by the desired property name in the comment of the field.

For example, in Natural:

```
#PARAM1 (A1) /* oapi.n:nameParm IN
```

This creates an API parameter named `nameParm`, which is mapped to the `#PARAM1` Natural parameter.

Restricted Statements

- **Must not be used**
- **Restricted use**
- **Must not block**

Must not be used

Statement	Reason
TERMINATE	Ends the session the server owns. The request is abandoned and output parameters are not written back.
STOP	Leaves the <code>CALLNAT</code> without writing the output parameters back. Use <code>ESCAPE ROUTINE</code> .
FETCH	Starts a program inside the request. The rest of the subprogram does not run and output parameters are not written back.
RUN	Requires source and a compiler in a session that serves requests.
LOGON	The library environment is established by the server per request.
REINPUT	Requires a terminal.
Maps (INPUT USING MAP, WRITE USING MAP), PROCESS PAGE	Requires a terminal.
SET KEY	Requires a terminal. PF key settings are reset per request.
SET CONTROL	Requires a terminal.
OPEN CONVERSATION, CLOSE CONVERSATION	Conversational RPC is not supported.
DEFINE DATA CONTEXT	Context variables are the data of a conversation, and there is none. The area belongs to the single <code>CALLNAT</code> that creates it and is released with it, so it carries nothing from one call to the next. Use the <code>PARAMETER</code> block.

Restricted use

Statement	What is Possible
STACK	Data can be placed on the stack and read back within the same request. Commands are not executed.
INPUT	Only without terminal I/O, that is, reading data from the stack. Any INPUT that would address a screen fails.

Must not block

A blocked subprogram blocks the session it runs in, and with it every request that would have been served by it.

- Do not wait for user interaction, external events, or locks held elsewhere.
- Keep CALLs to 3GL modules short. The server cannot interrupt a running 3GL call, so the session stays occupied for as long as the call lasts.
- Keep units of work short. Database locks are held until `END TRANSACTION` or `BACKOUT TRANSACTION`, so a long unit of work keeps other requests away from the same records.

The same applies to the subprogram as a whole: its runtime is the time the session is unavailable to everyone else.

Statelessness

Every call must stand on its own: it must not carry data forward, and it must not expect data left behind by an earlier call.

The following are put back to a defined state at every request boundary:

- Global data areas, including the COMMON GDA.
- Application independent variables (`DEFINE DATA INDEPENDENT`).
- Session parameters, including those changed with `SET GLOBALS` and `*LANGUAGE`.
- The limit counters `MADIO`, `MAXCL`, `MAXERR`, and `LT`.
- Steplib chains.
- Retained sets and command IDs.
- PF key settings.

A GDA is empty at the start of every request unless the subprogram fills it itself, and a set retained by `RETAIN` in one request does not exist in the next.

Context variables (`DEFINE DATA CONTEXT`) are no way around this either. Without a conversation, they are not a data area that spans calls: the context exists for the one `CALLNAT` and is gone when

it returns, so it can neither carry state forward nor be read by the next request. For more information, see *Must not be used*.

*ETID and the ET data stored under it belong to the session, not the end user. They are shared by all requests the session serves and must not be used to carry per-user state.

Transactions and Work Files

End your own units of work. Complete every one with `END TRANSACTION` or `BACKOUT TRANSACTION` before returning. The server does not commit.

At the request boundary, the server normally backs out the transaction (`BACKOUT TRANSACTION`) and closes open work files. On Linux, you can disable either behavior with `backoutTransactionPerRequest` and `closeWorkFilesPerRequest`; otherwise both behaviors always occur.



Notes:

1. A unit of work that is not committed within the request is lost.
2. Records must not be held across calls.
3. A work file must be opened, used and closed within one request. `CLOSE MODE=USER` does not keep it open across the request boundary.

Error Handling

Errors that occur while a request is being processed are intercepted by the server and reported to the caller with error number, line, status, and object name.

It is recommended to use `ON ERROR` to return errors as data the caller can act on — an error code and message in the output parameters — and reserve unhandled errors for genuinely unexpected conditions. Clean up on the error path exactly as on the success path.

System Variables

Variable	Value
*SERVER-TYPE	CALLNAT
*USER	The user of the current request in uppercase, up to 8 characters.
*NET-USER	The user of the current request, like *USER. On z/OS, it holds the user of the session the server started instead — the same value as *INIT-USER — and therefore does not identify the requesting user.
*LIBRARY-ID	The library named by the current request.
*INIT-USER, *INIT-ID, *ETID	The session the server started, not the requesting user. On z/OS, *INIT-ID holds a session number that the server increments for each new session.

Variable	Value
*DEVICE	A batch device.

Checklist

- [] No TERMINATE, STOP, FETCH, RUN, and LOGON.
- [] No REINPUT, no maps, no SET KEY, no SET CONTROL; INPUT only without terminal I/O.
- [] No OPEN CONVERSATION / CLOSE CONVERSATION, no DEFINE DATA CONTEXT.
- [] The subprogram read only values that it had set itself; it did not read values from a GDA, independent variables, or values retained from an earlier call.
- [] Every unit of work ended with END TRANSACTION or BACKOUT TRANSACTION, on every path.
- [] Every work file opened is closed, on every path.
- [] Nothing blocks — neither the subprogram nor a 3GL module it calls.
- [] The end user is identified through *USER, not through *NET-USER, *INIT-USER, or *ETID.