Statements for Internet and XML Access

This chapter gives a functional overview of the Natural statements for internet and XML access, specifies the general prerequisites for using these statements in a mainframe environment, informs about restrictions that apply and contains a list of further references. To take full advantage of these statements, a thorough knowledge of the underlying communication standards is required.

The following topics are covered:

- Statements Available
- General Prerequisites
- HTTPS Support for the REQUEST DOCUMENT Statement under z/OS
- Restriction Concerning IMS TM
- Preconditions for the Support of XML-Related Statements under openUTM
- Sample Program
- Frequently Asked Questions
- References

Statements Available

The following Natural statements are available for access to the internet and to XML documents:

- REQUEST DOCUMENT
- PARSE XML

REQUEST DOCUMENT

Functionality

This statement enables you to use the Hypertext Transfer Protocol (HTTP) and - under z/OS only - the Hypertext Transfer Protocol Secure (HTTPS) in order to access documents on the web with a given Uniform Resource Identifier (URI) or Uniform Resource Locator (URL), that is, the internet or intranet address of a web site.

`REQUEST DOCUMENT` implements an HTTP client at Natural statement level, which allows applications to access any HTTP server on either the intranet or the internet. The statement has a set of operands, which allows it to formulate HTTP requests according to the needs of the user application. For example, using outbound operands it is possible to send user-defined HTTP headers, form data, or entire documents to a HTTP server. The inbound operands can be used to retrieve a document from the server, to view the entire HTTP header block returned from the server, or to return the values of dedicated headers, etc. Via binary format operands, binary objects such as gif files can be exchanged with the http server as well. For basic authorization purposes, user ID and password operands can be specified. The content of this operand is sent with base64 encoding over the line, according to HTTP standards.

Natural supports the following request methods:

- `GET` - retrieve a document and HTTP headers,
- `HEAD` - retrieve HTTP headers only,
- `POST` - transfer form data to an HTTP server,
- `PUT` - upload a file to an HTTP server.

The request method is normally evaluated automatically, based on the operands coded for the executed `REQUEST DOCUMENT` statement. However, the predetermined request method can be overwritten by an explicit user specification of a request method header.

Data transfer with the `REQUEST DOCUMENT` statement normally does not involve any code page conversion. If you want to have the outgoing and/or incoming data encoded in a specific code page, you can use the `DATA ALL` clause and/or the `RETURN PAGE` clause of the `REQUEST DOCUMENT` statement to specify this.

In order to simplify data exchange from EBCDIC-based mainframes with HTTP servers, which in most cases work with UTF-8 or ISO-8859-1 encoded data, the statement provides `ENCODED` clauses to allow implicit or automatic conversion of outbound and inbound document data.

Technical Implementation

The implementation of the `REQUEST DOCUMENT` statement mainly consists of two layers:

- an independent runtime layer, where the entire HTTP processing, URL analysis, data conversion, etc., is executed; and
- a layer where an environment-dependent routine processes the TCP/IP communication between Natural and the HTTP server. This layer is implemented based on LE (Language environment) sockets for z/OS, VSE, and VM/CMS; SMARTS sockets for Com-plete and Natural Development Server; and CRTE sockets for BS2000/OSD. For CICS, the appropriate socket library is included into the build process.

Natural for Mainframes supports the HTTP protocol version 1.0 only, meaning that no persistent connection to the server is maintained. Since virtually every corporate network processes access to the internet via a proxy server from the client, Natural can be configured with the adequate settings for the proxy server and the port on which the proxy server runs. Moreover, it is possible to specify local domain name suffixes (intranet sites), which shall be accessed directly instead via the proxy server. See also *Overview of Applicable Natural Parameters*.

The proxy server, which is located between client (user) and internet, serves the following purposes: It receives the request from the clients, forwards it to the target server, caches the returned document and forwards it to the client. A proxy server is advantageous because of its improved performance, which is due to the caching, and because it helps to avoid security issues (most proxy servers are working as firewall as well).

The following is an example of how the `REQUEST DOCUMENT` statement can be used to access an externally-located document:

```
REQUEST DOCUMENT FROM
"http://bolsap1:5555/invoke/sap.demo/handle_RFC_XML_POST"
WITH
USER #User PASSWORD #Password
DATA
NAME 'XMLData' VALUE #Queryxml
NAME 'repServerName' VALUE 'NT2'
RETURN
PAGE #Resultxml
RESPONSE #rc
```

Syntax

The syntax of the REQUEST DOCUMENT statement and detailed application hints are to be found in the *Statements* documentation.

Platform Support for REQUEST DOCUMENT

The REQUEST DOCUMENT statement is supported on the following mainframe platforms:

- **z/OS:** Batch, TSO, CICS, Com-plete, IMS TM
- **z/VSE:** Batch, Com-plete, CICS
- **VM/CMS**
- **BS2000/OSD:** Batch, TIAM, *open*UTM *

* See also *Preconditions for the Support of XML-Related Statements under openUTM* below.

Moreover this statement is available on all OpenSystems platforms that are supported by Natural.

PARSE XML

Functionality

The PARSE XML statement allows you to parse XML documents from within a Natural program.

The PARSE XML statement integrates a full XML parser into Natural, thus allowing Natural applications to parse XML documents in order to easily process their content. The PARSE XML statement opens a processing loop and returns, whenever one of a list of events occurs during the parse process, the respective path through the document, name and value of parsed elements together with some parser status system variables.

Technical Implementation

For parsing XML documents the following parsing strategies or models are most common:

- DOM (Document Object Model), an object oriented approach
- SAX (Simple Access to XML), a stream-oriented parsing method

The implementation of the PARSE XML statement in Natural for Mainframes is based on the SAX method, using a mainframe port of version 2.0.0 of the (open source) SAX parser EXPAT.

Parsing is processed internally on a UTF-16 encoded image of the document to parse, that is, if the document is not delivered in this encoding, an internal conversion to UTF-16 is performed before the parsing starts. This has to be considered at Natural installation time, for example, when the thread size for the TP environment is evaluated.

The encoding of the document to parse is checked automatically.

1. A check for a BOM (Byte Order Mark), which marks the document's encoding, is done.
2. If no BOM is found, a check for ASCII, EBCDIC, or UTF-16 (BE or LE: big endian or little endian) is done.
3. If an EBCDIC or ASCII encoding is identified, a search for an encoding processing instruction is performed.

   If no encoding can be identified, an adequate error message is issued and the parse process is terminated. Internally, the parser works with UTF-16BE, so the document to parse is always converted to this encoding before it is passed to the EXPAT parser.
4. If an encoding PI (Processing Instruction) is found, the following defaults apply:
   - for ASCII, UTF-8 is assumed as encoding
   - for EBCDIC, the Natural default code page (see system variable `*CODEPAGE`) is assumed as encoding

The parse process itself consists of two phases.

- In the first phase, the parser is called repeatedly to announce a well-defined set of callback entries. Those entries are entered by the parser whenever a corresponding element is encountered in the current parsed document. The occurrence of a start tag is, for instance, such an event which triggers a callback to the corresponding entry. The callback entries expose the Natural runtime logic for the execution of the parse process.
- The second phase is the actual parse process. The parser is called with the document to parse as input operand. Now, each element is parsed, and for each element type its corresponding callback routine is called. The Natural runtime then processes the returned element, updates the return operands, and enters the parse loop for processing those operands. Then the parser is restarted to continue the parse process. The parse process is finished either if the document is completely parsed or if an XML syntax error occurs in the current document, meaning the document is not well formed.

Note:
For technical reasons, nested parse loops are not supported in Natural for Mainframes.

Processing of XML Whitespace Characters and Predefined Entities

As of Natural Version 4.2.5, parsing of character data does not cause a break or a loop path if the parsed string contains whitespace characters or predefined XML entities. This problem with Natural versions prior to Version 4.2.5 has been solved. With Natural Version 4.2.5, parsing of character data is compatible with Natural for Windows, UNIX and Linux.

The outputs from the following sample program show the difference between Version 4.2.5 and its predecessors.

```
DEFINE DATA
LOCAL
1 PAGE    (A)   DYNAMIC
1 #PATH   (A200)
1 #NAME   (A) DYNAMIC
1 #VALUE  (A40)
1 #CMX    (A) DYNAMIC
1 #CMP    (A) DYNAMIC
END-DEFINE
FORMAT PS=60 LS=80
COMPRESS ' A&lt;B ' H'0D0D' ' B&lt;C' INTO #CMX LEAVING NO
MOVE ALL  #CMX TO #CMP UNTIL 16
COMPRESS
'<?xml version="1.0" ?>'
'<character-data-sample>'
'<string_with_whitespace_and_predefined_entity>' #CMX
'</string_with_whitespace_and_predefined_entity>'
'</character-data-sample>'
 INTO PAGE LEAVING NO
PARSE XML PAGE INTO PATH #PATH NAME #NAME VALUE #VALUE
PRINT #PATH /   'NA='  #NAME / 'VA=' #VALUE
LOOP
END
```

Output if the program is executed in Natural versions below Version 4.2.5:

```
Page      1                                                  08-11-04  14:39:51

character-data-sample
NA= character-data-sample
VA=
character-data-sample/string_with_whitespace_and_predefined_entity
NA= string_with_whitespace_and_predefined_entity
VA=
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA=  A
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA= <
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA= B
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA= ?
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA= ?
NA=
VA= B
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA= ?
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA= ?
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
```

```
VA=  B
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA= <
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA= C
character-data-sample/string_with_whitespace_and_predefined_entity//
NA= string_with_whitespace_and_predefined_entity
VA=
character-data-sample//
NA= character-data-sample
VA=
MORE
```

Output if the program is executed in Natural Version 4.2.5 (or higher):

```
Page      1                                                08-11-04  13:41:34

character-data-sample
NA= character-data-sample
VA=
character-data-sample/string_with_whitespace_and_predefined_entity
NA= string_with_whitespace_and_predefined_entity
VA=
character-data-sample/string_with_whitespace_and_predefined_entity/$
NA=
VA=  A<B?? B<C
character-data-sample/string_with_whitespace_and_predefined_entity//
NA= string_with_whitespace_and_predefined_entity
VA=
character-data-sample//
NA= character-data-sample
VA=


MORE
```

Syntax

The syntax of the PARSE XML statement and detailed application hints are to be found in the *Statements* documentation.

Platform Support for PARSE XML

The PARSE XML statement is supported on the following mainframe platforms:

- **z/OS:** Batch, TSO, CICS, Com-plete, IMS TM *
- **z/VSE:** Batch, Com-plete, CICS
- **VM/CMS**
- **BS2000/OSD:** Batch, TIAM, *open*UTM **

* See *Restriction Concerning IMS TM* below.

* * See also *Preconditions for the Support of XML-Related Statements under openUTM* below.

Moreover this statement is available on all Open Systems platforms that are supported by Natural.

General Prerequisites

This section describes the general prerequisites that apply if you wish to use the Natural statements `REQUEST DOCUMENT` and `PARSE XML`.

- Installation Prerequisites
- Profile Settings
- Activation/Deactivation
- Unicode Support

Installation Prerequisites

To enable the use of the Natural statements `REQUEST DOCUMENT` and `PARSE XML`, the installation steps described in the Installation documentation must be performed; *Installation Steps for REQUEST DOCUMENT and PARSE XML*.

Since `REQUEST DOCUMENT` as well as `PARSE XML`, at least internally, always have to convert data from one encoding to another, Natural has to be driven with active ICU support. Therefore, the ICU library must be installed.

If `REQUEST DOCUMENT` or `PARSE XML` is to be executed, the following prerequisites must be fulfilled:

- a TCP/IP stack must be available and enabled for the execution environment,
- a DNS (Domain Name System) server or DNS services must be available in the execution environment to resolve internet address resolution requests (`gehthostbyname` function),
- a Natural driver must be installed LE enabled (in IBM environments) or CRTE enabled (in BS2000/OSD environments),
- support of HTTPS under Com-plete requires APS Version 2.7.2 Patch Level 16.

Profile Settings

Overview of Applicable Natural Parameters

The following is an overview of Natural profile and/or session parameters that enable/disable or influence the support of the statements `REQUEST DOCUMENT` and/or `PARSE XML`:

Parameter	Purpose
XML	This Natural profile parameter and/or the corresponding parameter macro NTXML in conjunction with their keyword subparameters are used to activate/deactivate the statements REQUEST DOCUMENT and PARSE XML. In addition, there are various options that can be set with the keyword subparameters of NTXML and XML, such as separate enabling/disabling of support of the REQUEST DOCUMENT and PARSE XML statements, name of the default code page, URL of the (intranet) proxy server, port number of the proxy, URL and port number of the (intranet) SSL proxy server, and name(s) local domain(s) which are to be addressed directly. As a prerequisite for using the XML profile parameter, the profile parameter CFICU must be set to ON.
CFICU	This Natural profile parameter and/or the corresponding parameter macro NTCFICU in conjunction with their keyword subparameters are used to enable Unicode and code page support.
CP	This Natural profile parameter defines the default code page for Natural data and Natural sources.
CPCVERR	This Natural profile and session parameter specifies whether a conversion error that occurs when converting results in a Natural error or not.

Activation/Deactivation

To activate the support of the statements REQUEST DOCUMENT and PARSE XML for the *current* session

1. To activate both statements *together*, set the Natural profile parameter XML (or the corresponding parameter macro NTXML) and also its keyword subparameters RDOC and PARSE to ON.

   Or:
   To activate the support *individually*, set only the corresponding XML/NTXML keyword subparameter to ON:

   RDOC to enable support of REQUEST DOCUMENT

   PARSE to enable support of PARSE XML

2. If the installation platform operates behind an internet firewall or if the internet traffic is routed via a proxy server, the XML/NTXML keyword subparameters for proxy and proxyport have to be specified accordingly.

To activate the support of the statements REQUEST DOCUMENT and PARSE XML for *all* sessions

- Ask your system administrator to add the parameters and/or macros listed in the *Overview of Applicable Natural Parameters* to the Natural parameter module NATPARM and to set the values correspondingly.

To deactivate the support of the statements REQUEST DOCUMENT and PARSE XML

- To activate both statements *together*, set the Natural profile parameter `XML` or the parameter macro `NTXML` to `OFF`.

  Or:
  To deactivate the support *individually*, set only the corresponding `XML`/`NTXML` keyword subparameter to `OFF`:

  `RDOC` to disable support of `REQUEST DOCUMENT`

  `PARSE` to disable support of `PARSE XML`

For information, see *XML - Activate PARSE XML and REQUEST DOCUMENT Statements* in the *Parameter Reference* documentation

Unicode Support

To enable Unicode support

- Set the profile parameter `CFICU` must be set to `ON`.

For information on the various options that can be set with the keyword subparameters of profile parameter `CFICU`, see *CFICU - Unicode Support* in the *Parameter Reference* documentation.

See also the paragraphs relating to the statements `PARSE XML` and `REQUEST DOCUMENT` in the section *Statements*, which is part of the section *Unicode and Code Page Support in the Natural Programming Language* in the *Unicode and Code Page Support* documentation.

HTTPS Support for the REQUEST DOCUMENT Statement under z/OS

- Short Introduction to HTTPS
- HTTPS over AT-TLS
- Maintenance of Certificates under z/OS
- Using RACF Key Rings
- Using Key Databases

Short Introduction to HTTPS

HTTPS, short for Hypertext Transfer Protocol Secure, is an additional security layer between HTTP and the TCP/IP protocol stack:

Layer	Protocol
Application layer	HTTP(S)
Security layer	TLS/SSL
Transport layer	TCP
Network layer	IP

It was introduced to enable encryption and communication partner authentication for a secure data communication over the internet.

The HTTPS URI scheme is used to indicate, that the HTTP communication is secured. For the encryption of the data the SSL (Secure Socket Layer) protocol or its successor TLS (Transport Layer Security) is used. Authentication is hereby provided by the exchange of certificates, which guarantee the identity of the communication partners.

In most cases of HTTPS communication, however, only the server identifies itself with a certificate against the client. Client authentication with a client certificate occurs quite seldom.

SSL communication is established in several steps:

- It starts with identification and authentication of the communication partners over the so called SSL handshake protocol (Client Hello, Server Hello).
- The handshake is followed by the exchange of a symmetric session key via asymmetric encryption (private – public key proceeding). The public key, which is hereby used by the client, is an essential part of the server certificate.
- After the handshake and key exchange have been performed, the encrypted payload request messages are communicated. The symmetric session key, which was negotiated in the preceding steps, is used for the encryption/decryption of those messages.

The HTTPS protocol uses port numbers which differ from the standard HTTP port numbers. While HTTP normally uses port 80, the default port number for HTTPS is 443.

HTTP access to the internet from a client, which is connected to a LAN (Local Area Network), is normally processed via special HTTP servers, so called proxies. Proxies are gateways to the internet from the LAN, which perform security policies, provide caches and validation routines or filter functions and act as firewalls. HTTPS secured internet access is most often performed over a proxy server of its own, which maintains the connections to the remote servers. This proxy is also known as "SSL proxy".

Certificates are binary documents, which contain, among other information items, information about the owner and the issuer of the certificate, the public key for the encryption of the session key data, an expiration date and a digital signature. The certificates, which are presented by HTTPS servers, are normally the lowest link of an entire chain of certificates. Such a certificate chain is called a Public Key Infrastructure (PKI). The certificate on top of such a chain is called a root certificate. Root certificates are generally issued by special organizations, named Certificate Authorities (CA). Root certificates, which are issued and signed by a CA are also called CA (root) certificates. For further information, see *HTTP Developers Manual* and other sources in the internet.

HTTPS over AT-TLS

HTTPS support for the Natural `REQUEST DOCUMENT` statement is based on the z/OS Communication Server component AT-TLS (Application Transparent-Transport Layer Security).

AT-TLS provides TLS/SSL encryption as a configurable service for sockets applications. It is realized as an additional layer on top of the TCP/IP protocol stack, which exploits the SSL functionality in nearly or even fully transparent mode to sockets applications. AT-TLS offers three modes of operation. See *z/OS Communications Server, IP Programmer's Guide and Reference. Version 1, Release 9*, Chapter 15, IBM manual SC31-8787-09).

These modes are:

- **Basic**

  The sockets application runs without modification in transparent mode, unaware of performing encrypted communication via AT-TLS. Thus legacy applications can run in secured mode without source code modification.

- **Aware**

  The application is aware of running in secured mode and is able to query TLS status information.

- **Controlling**

  The sockets application is aware of AT-TLS and controls the use of AT-TLS encryption services itself. This means, the application is able to switch between secured and non secured communication.

Natural for Mainframes uses *Controlling* mode to switch on secured mode for HTTPS requests only, while HTTP requests remain unencrypted.

Maintenance of Certificates under z/OS

Certificates, which are to be used with AT-TLS, can be maintained in two ways under z/OS. They are stored in RACF key rings or in key databases, which are located in the z/OS UNIX file system. Which of these proceedings actually applies is defined in the AT-TLS Policy Agent Configuration file for the z/OS TCP/IP stack, which is used by the Natural HTTPS client.

IBM delivers a set of commonly used CA root certificates with each z/OS system delivery. If key rings are going to be used to hold server certificates, those root certificates must be manually imported into the key rings by the system administrator. If IBM delivers newer replacements for expired root certificates, all affected key rings have to be updated accordingly.

Unlike key rings, key databases contain the current set of root certificates automatically after they have been newly created. However, the need for maintaining always the latest set of root certificates applies to the key database alternative as well.

Certificates to be used by the Natural HTTPS client, must be flagged as trusted. If they are part of a Public Key Infrastructure, the corresponding CA root certificate has to be flagged as trusted.

Using RACF Key Rings

In RACF, digital certificates are stored in so called key rings. The RACF command RACDCERT is used to create and maintain key rings and certificates, which are contained in those key rings.

See *z/OS Security Server RACF Security Administrator's Guide*, IBM manual SA22-7683-11, and *z/OS Security Server RACF Command Language Reference*, IBM manual SA22-7687-11.

Using Key Databases

Alternatively to RACF, certificates can be kept in key databases, which reside in the z/OS UNIX services file system. For the creation and maintenance of key databases, the GSKKYMAN utility has to be used.

See *z/OS Cryptographic Services PKI Services Guide and Reference*, IBM manual SA22-7693-10.

Restriction Concerning IMS TM

The following restriction applies if you wish to use the Natural statements REQUEST DOCUMENT and PARSE XML in an IMS TM environment:

- The PARSE XML statement can be executed under the TP monitor IMS TM with the restriction that no I/O statement is allowed within an active PARSE loop. If an I/O occurs within a PARSE loop, error NAT0967 is issued.

For further restrictions, see the corresponding notes in the statement descriptions.

Preconditions for the Support of XML-Related Statements under *open*UTM

During an active parse loop with I/Os, the UTM function call PGWT must be used. This means:

1. The UTM application must be started with not less than 2 tasks, otherwise a UTM error K319 with subsequent dump will occur.
2. PGWT conditions must be defined for the KDCDEF.
    1. Define the maximum wait time (in seconds) for input messages during a PGWT call.

       Example:

       ```
       MAX PGWTTIME=60
       ```

    2. Define the maximum number of UTM tasks for PGWT calls.

       Example:

       ```
       MAX TASKS-IN-PGWT=1
       ```

    3. PGWT can be controlled using either the TAC-PRIORITIES instruction or the TACCLASS concept:

- Control of PGWT using the TAC-PRIORITIES instruction:

  Example:

```
DEFAULT TAC TYPE=D,PROGRAM=NATUTM,. . . . . .
TAC NAT,ADMIN=NO,TIME=(0,0),PGWT=YES,TACCLASS=1
TAC-PRIORITIES DIAL-PRIO=EQ
```

- Control of PGWT using the TACCLASS concept:

  Example:

```
DEFAULT TAC TYPE=D,PROGRAM=NATUTM,. . . . . .
TAC NAT,ADMIN=NO,TIME=(0,0),TACCLASS=1
TAC NAT1,ADMIN=NO,TIME=(0,0),TACCLASS=2
TACCLASS 1,TASKS=2
TACCLASS 2,TASKS=1,PGWT=YES
```

3. The keyword subparameter ILCS of parameter macro NURENT must be set to ILCS=CRTE.

Sample Program

The following sample program shows the usage of the REQUEST DOCUMENT and the PARSE XML statement.

Further sample programs are provided at the end of the description of each statement and in the Natural library SYSEXV.

```
DEFINE DATA
LOCAL
1 #FROM   (A) DYNAMIC
1 #HEADER (A) DYNAMIC
1 #PAGE   (A) DYNAMIC
1 #RC     (I4)
1 #COL    (N8)
1 #COL1   (I4)
1 #COL2   (I4)
1 #COL3   (I4)
1 #LOC    (A30)
1 #CP     (A) DYNAMIC
1 #PATH   (A) DYNAMIC
1 #NAME   (A) DYNAMIC
1 #VALUE  (A) DYNAMIC
1 #RTERR  (I4)
END-DEFINE
*
ASSIGN #FROM = 'HTTP://SI15.HQ.SAG/autos6.xml'
**
REQUEST DOCUMENT FROM #FROM
RETURN
HEADER ALL #HEADER
PAGE #PAGE    ENCODED    FOR TYPES 'TEXT/XML'
 CODEPAGE ' '
RESPONSE #RC
GIVING #RTERR
**
IF #RC NE 200 /* TEST FOR HTTP RESPONSE 200 = 'OK'
WRITE 'HTTP RESPONSE' #RC 'RECEIVED'
ESCAPE ROUTINE
```

```
END-IF
EJECT
PRINT #HEADER
/ '_'(79)
PRINT #PAGE
/ '_'(79)
/ '_'(79)
ASSIGN #CP = *CODEPAGE
EXAMINE #PAGE FOR 'encoding' GIVING POSITION  #COL1
 IF #COL1 GT 0
   EXAMINE #PAGE FOR '?>' GIVING POSITION #COL3
   IF #COL3 GT #COL1
     EXAMINE #PAGE FOR 'ISO-8859-1' GIVING POSITION #COL2
   END-IF
   IF #COL2 GT #COL1 AND #COL2 LT  #COL3
     EXAMINE #PAGE FOR 'ISO-8859-1' REPLACE #CP
   END-IF
 END-IF
PRINT #PAGE
/ '_'(79)
EJECT
PARSE XML #PAGE INTO PATH #PATH NAME #NAME VALUE #VALUE
 PRINT #PATH / 'NAME=' #NAME / 'VALUE=' #VALUE / '_'(79)
END-PARSE
END
```

Note:
The URL accessed in the above program addresses an intranet site and cannot be accessed from the internet.

Output of the sample program:

```
HTTP/1.1 200 OK?Date: Thu, 10 Aug 2006 16:26:22 GMT?Server: Apache/1.3.19 (
BS2000)?Last-Modified: Thu, 27 Jul 2006 16:44:42 GMT?ETag: "2602c-111-44c8ed7a"
?Accept-Ranges: bytes?Content-Length: 273?Connection: close?Content-Type: text/
xml??
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?><autos>?<make></make>?<make>Ford</
make>?<model>Thunderbird</model>?<make>Merceds-Benz</make><model>S400</model><
make>BWM</make><model version="latest">330I</model>?<make><label><company>
Mercedes</company></label></make>?</autos>?
```

```
<?xml version="1.0" encoding="IBM01140" ?><autos>?<make></make>?<make>Ford</
make>?<model>Thunderbird</model>?<make>Merceds-Benz</make><model>S400</model><
make>BWM</make><model version="latest">330I</model>?<make><label><company>
Mercedes</company></label></make>?</autos>?
```

```
MORE

autos
Name= autos
Value=
```

```
autos/$
Name=
Value= ?
```

```
autos/make
Name= make
Value=
```

```
autos/make//
Name= make
Value=
```

```
autos/$
Name=
Value= ?
```

```
autos/make
Name= make
Value=
VVVV
Name= autos
Value=
```

```
autos/$
Name=
Value= ?
```

```
autos/make
Name= make
Value=
```

Frequently Asked Questions

- Why needs code page support to be enabled?
- How to use the XML keyword subparameters (e.g. RDP and RDNOP)
- How to determine proxy server, port number and HTTP server at a site?
- How to decide if a problem is a TCP/IP or HTTP issue or if it is a Natural issue?
- How can I check if I can reach a website from my mainframe without using Natural?
- Is NAT2TCP correctly loaded?
- I get a message "unsupported coding"
- How to avoid Natural error NAT3411 with REQUEST DOCUMENT?
- Can I use self-signed certificates?
- Which is the preferable method for maintaining certificates?
- How to configure TCP/IP for AT-TLS?
- How to verify AT-TLS configuration?
- Is there more information about problem determination?
- How to switch on P-agent trace?

- Error at connection establishment

Why needs code page support to be enabled?

Documentation for Natural on mainframe states that "The Natural ICU handler must be linked to the Natural nucleus".

PARSE XML statement

The codepage support is needed, as on mainframe platforms, the document to be parsed is always internally converted to UTF-16 (if the document is not already encoded in UTF-16). In most cases the document is not in UTF-16 and a conversion will take place. For more detailed information, see the `PARSE XML` statement documentation and `PARSE XML` in the Unicode and Code Page Support documentation.

REQUEST DOCUMENT statement

The ICU library is needed to interpret incoming HTTP headers and convert outgoing HTTP headers. Typically the incoming headers are ISO 8859-1 encoded and on mainframe always have to be converted to the Natural default codepage (see also system variable `*CODEPAGE`) - on PC a conversion is not always necessary.

How to use the XML keyword subparameters (e.g. RDP and RDNOP)

On the PC, the `REQUEST DOCUMENT` statement executes the Internet Explorer and uses the settings as defined there.

On the mainframe, the URL of the (intranet) proxy server through which all requests have to be routed has to be specified with the `NTXML/XML` keyword subparameter `RDP`. With the keyword subparameter `RDNOP`, local domain(s) which are to be addressed directly, not via the proxy server can be defined.

How to determine proxy server, port number and HTTP server at a site?

Information about proxy server, port number and HTTP server at a site has to be provided by the network administrator.

You can also look into your browser which proxy server is defined for your site.

For example, in the Internet Explorer under: Tools > Internet Options > Lan Settings > Advanced

You can also search the web for tools which provide such information. For example (untested):
http://www.sharewareconnection.com/titles/proxy-settings.htm

How to decide if a problem is a TCP/IP or HTTP issue or if it is a Natural issue?

HTTP response codes

HTTP response is returned via the `RESPONSE` clause in *operand16*. An *Overview of Response Numbers for HTTP/HTTPs Requests* is available in the *Statements* documentation.

TCP/IP errors

The range for these errors is 8300 ff.

Especially error NAT8304 gives more detailed information about a failing HTTP request.

As the TCP/IP error number may be different depending on the installed environment, the text returned by NAT8304 is the best reference.

Additional information:

- See buffer RDOCWA at Offset 480
- Quite often these errors are ICU errors: Recommendation is to set profile or session parameter CPCVERR to OFF.

How can I check if I can reach a website from my mainframe without using Natural?

To determine if a problem is related to the Natural installation or if there is a more general problem, you can do a PING from within TSO.

For example, in the TSO command shell enter:

```
TSO PING www.google.com
```

This returns:

```
CS V1R9: Pinging host WWW.GOOGLE.COM (66.249.91.99)
Ping #1 response took 0.018 seconds.
```

From within the Natural session you then can test the access to this website with the following small program.

For example, start Natural with:

```
NATvr CFICU=ON
XML=(ON,RDOC=ON,PARSE=ON,RDP='HTTPPROX.HQ.SAG',RDPPORT=8080,RDNOP='*.EUR.AD.SAG;
*.HQ.SAG;*.SOFTWAREAG.COM')
```

where *vr* stands for the Natural release and version number.

These values from an internal environment and a profile were used to store it. You have to get your settings for the keyword subparameters RDP, RDPPORT and RDNOP from your network administrator, or try the values as defined in your browser (Internet Explorer).

Execute:

```
DEFINE DATA LOCAL
1 #RESULTXML (A) DYNAMIC
1 #RC (I4)
END-DEFINE
REQUEST DOCUMENT FROM "HTTP://WWW.GOOGLE.DE"
RETURN HEADER ALL #HEADER RESPONSE #RC
WRITE #RC
WRITE #HEADER (AL=79)
END
```

Is NAT2TCP correctly loaded?

You can check this with the `SYSPROD` utility.

In `SYSPROD`, enter the command `SC` (`Display subcomponents`) for product Natural. When you scroll through the installed subcomponents, you should find an entry for `Nat Request Document` (Product ID .... TCP).

I get a message "unsupported coding"

This is a frequent user error: An XML document is converted implicitly or explicitly from one code page to another, for example, from ISO-8859-1 to the code page found in system variable `*CODEPAGE`. The document's encoding `PI encoding="ISO-8859-1"`, however, has not been not adapted to the changed encoding. In this case, the parser terminates with an error already on the first character of the document to parse.

How to avoid Natural error NAT3411 with REQUEST DOCUMENT?

Set session parameter `CPCVERR` to `OFF`.

Can I use self-signed certificates?

Self-signed certificates can be used on an intranet server for test purposes, using the open ssl sdk. After these certificates have been imported into a key database or a RACF key ring, they must be flagged as trusted.

Which is the preferable method for maintaining certificates?

The necessary effort for the RACF key ring approach seems to be much higher than for using key databases. Key rings must be created for every user who wants to access a HTTPS server, whereas key databases can be shared by multiple users.

How to configure TCP/IP for AT-TLS?

Proceed as follows:

1. In the TCP/IP configuration file, set the option `TTLS` in the `TCPCONFIG` statement.
2. Configure and start the AT-TLS Policy Agent. This agent is called by TCP/IP on each new TCP connection to check if the connection is SSL.

3. Create the Policy Agent file containing the AT-TLS rules. The Policy Agent file contains the rules to stipulate which connection is SSL.

See also *z/OS Communications Server: IP Configuration Guide*, Chapter 18 *Application Transparent Transport Layer Security (AT-TLS) data protection*.

The Sample Policy Agent file defines all outgoing connections as application controlled TLS. This should not affect any other TCP/IP application except the Natural `REQUEST DOCUMENT` support, because the rule is defined as application controlled. That means the application is allowed to set the connection status as SSL. As long as the application does not set this status, it is not affected. However, the Policy Agent file allows also to restrict the application controlled SSL connections to particular ports, users or address spaces. The sample expects the certificate database on the HFS file `/ u/admin/CERT.kdb`.

```
TTLSRule                          ConnRule01~1
{
  LocalAddrSetRef                 addr1
  RemoteAddrSetRef                addr1
  LocalPortRangeRef               portR1
  Direction                       Outbound
  Priority                        255
  TTLSGroupActionRef              gAct1~AllUsersAsClient
  TTLSEnvironmentActionRef        eAct1~AllUsersAsClient
  TTLSConnectionActionRef         cAct1~AllUsersAsClient
}
TTLSGroupAction                   gAct1~AllUsersAsClient
{
  TTLSEnabled                     On
  Trace                           6
}
TTLSEnvironmentAction             eAct1~AllUsersAsClient
{
  HandshakeRole                   Client
  EnvironmentUserInstance         0
  TTLSKeyringParmsRef             keyR1
}
TTLSConnectionAction              cAct1~AllUsersAsClient
{
  HandshakeRole                   Client
  TTLSCipherParmsRef              cipher1~AT-TLS__Silver
  TTLSConnectionAdvancedParmsRef  cAdv1~AllUsersAsClient
  Trace                           0
}
TTLSConnectionAdvancedParms       cAdv1~AllUsersAsClient
{
  ApplicationControlled           On
}
TTLSKeyringParms                  keyR1
{
  Keyring                         /u/admin/CERT.kdb
  KeyringStashFile                /u/admin/CERT.sth
}
TTLSCipherParms                   cipher1~AT-TLS__Silver
{
  V3CipherSuites                  TLS_RSA_WITH_DES_CBC_SHA
  V3CipherSuites                  TLS_RSA_WITH_3DES_EDE_CBC_SHA
  V3CipherSuites                  TLS_RSA_WITH_AES_128_CBC_SHA
}
IpAddrSet                         addr1
{
  Prefix                          0.0.0.0/0
```

```
}
PortRange                          portR1
{
  Port                             1024-65535
}
```

How to verify AT-TLS configuration?

Check Policy-Agent job output JESMSGLG for:

```
EZZ8771I PAGENT CONFIG POLICY PROCESSING COMPLETE FOR <your TCP/IP address space>: TTLS
```

This message indicates a successful initialization.

Check Policy-Agent job output JESMSGLG for:

```
EZZ8438I PAGENT POLICY DEFINITIONS CONTAIN ERRORS FOR <your TCP/IP address space>: TTLS
```

This message indicates errors in the configuration file. Check the syslog.log file for further information.

Does the configuration rule cover the client?

Check syslog.log for:

```
EZD1281I TTLS Map   CONNID: 00002909 LOCAL: 10.20.91.61..1751 REMOTE: 10.20.91.117..443
JOBNAME: KSP USERID: KSP TYPE: OutBound STATUS: Appl Control RULE: ConnRule01
ACTIONS: gAct1 eAct1 AllUsersAsClient
```

The above entry indicates that the connection to Port 443 by user KSP is application controlled.

Is there more information about problem determination?

See also *z/OS V1R8.0 Comm Svr: IP Diagnosis Guide: 3.23*, Chapter 29 *Diagnosing Application Transparent Transport Layer Security (AT-TLS)*

How to switch on P-agent trace?

See *Comm Svr: IP Configuration Reference*, Chapter 20 *Syslog deamon and Comm Svr: IP Configuration Guide*, Chapter 1.5.1 *Configuring the syslog daemon (syslogd)*

Error at connection establishment

Find return code RC and corresponding GSK_ function name in P-agent trace.

See *System SSL Programming* and locate the RC in Chapter 12.1 *SSL Function Return Codes*.

Sample trace with trace=255:

```
EZD1281I TTLS Map   CONNID: 00002909 LOCAL: 10.20.91.61..1751 REMOTE: 10.20.91.117..443 JOBNAME: KSP USERID: KSP TYPE: OutBound STATUS: A
EZD1283I TTLS Event GRPID: 00000003 ENVID: 00000000 CONNID: 00002909  RC:    0 Connection Init
EZD1282I TTLS Start GRPID: 00000003 ENVID: 00000002 CONNID: 00002909 Initial Handshake ACTIONS: gAct1 eAct1 AllUsersAsClient HS-Client
EZD1284I TTLS Flow  GRPID: 00000003 ENVID: 00000002 CONNID: 00002909  RC:    0 Call GSK_SECURE_SOCKET_OPEN - 7EE4F718
EZD1284I TTLS Flow  GRPID: 00000003 ENVID: 00000002 CONNID: 00002909  RC:    0 Set GSK_SESSION_TYPE -  CLIENT
EZD1284I TTLS Flow  GRPID: 00000003 ENVID: 00000002 CONNID: 00002909  RC:    0 Set GSK_V3_CIPHER_SPECS -  090A2F
EZD1284I TTLS Flow  GRPID: 00000003 ENVID: 00000002 CONNID: 00002909  RC:    0 Set GSK_FD - 00002909
EZD1284I TTLS Flow  GRPID: 00000003 ENVID: 00000002 CONNID: 00002909  RC:    0 Set GSK_USER_DATA - 7EEE9B50
EZD1284I TTLS Flow  GRPID: 00000003 ENVID: 00000002 CONNID: 00002909  RC:  435 Call GSK_SECURE_SOCKET_INIT - 7EE4F718
EZD1283I TTLS Event GRPID: 00000003 ENVID: 00000002 CONNID: 00002909  RC:  435 Initial Handshake 00000000 7EEE8118
EZD1286I TTLS Error GRPID: 00000003 ENVID: 00000002 CONNID: 00002909 JOBNAME: KSP USERID: KSP RULE: ConnRule01  RC:  435 Initial Handshake
EZD1283I TTLS Event GRPID: 00000003 ENVID: 00000002 CONNID: 00002909  RC:    0 Connection Close 00000000 7EEE8118
```

References

Below is a list of resources that you may find useful.

- Training Courses
- Useful Links

Training Courses

Software AG's Corporate University offers special training courses on this subject. See the Corporate University offerings on ServLine24 at *http://servline24.softwareag.com/public/*.

Or, ask your local Software AG representative for the availability of special on-site training courses at your location.

Useful Links

Below is a collection of links that may be of interest.

- World Wide Web Consortium (W3C): *http://www.w3.org/*
- Extensible Markup Language (XML): *http://www.w3.org/XML/*
- HyperText Markup Language (HTML) Home Page: *http://www.w3.org/MarkUp/*
- W3 Schools: *http://www.w3schools.com/*