

SAP Content Server HTTP Interface

HELP.BCSRVARLHTTP

Release 6.40

Copyright

© Copyright 2001 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft[®], WINDOWS[®], NT[®], EXCEL[®], Word[®], PowerPoint[®] and SQL Server[®] are registered trademarks of Microsoft Corporation.

IBM[®], DB2[®], OS/2[®], DB2/6000[®], Parallel Sysplex[®], MVS/ESA[®], RS/6000[®], AIX[®], S/390[®], AS/400[®], OS/390[®], and OS/400[®] are registered trademarks of IBM Corporation.

ORACLE[®] is a registered trademark of ORACLE Corporation.

INFORMIX[®]-OnLine for SAP and Informix[®] Dynamic Server[™] are registered trademarks of Informix Software Incorporated.

UNIX[®], X/Open[®], OSF/1[®], and Motif[®] are registered trademarks of the Open Group.

HTML, DHTML, XML, XHTML are trademarks or registered trademarks of W3C[®], World Wide Web Consortium, Massachusetts Institute of Technology.

JAVA[®] is a registered trademark of Sun Microsystems, Inc.

JAVASCRIPT[®] is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

SAP, SAP Logo, R/2, RIVA, R/3, ABAP, SAP ArchiveLink, SAP Business Workflow, WebFlow, SAP EarlyWatch, BAPI, SAPPHIRE, Management Cockpit, mySAP.com Logo and mySAP.com are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other products mentioned are trademarks or registered trademarks of their respective companies.

Icons

Icon Meaning



Caution



Example



Note



Recommendation



Syntax



Tip

Table of Contents

SAP Content Server HTTP Interface.....	5
Introduction.....	5
Definition of Terms.....	5
Implementation.....	7
Security.....	7
secKey.....	7
Degree of Protection / Right of Access.....	9
Syntax.....	10
General.....	10
URL Encoding.....	11
Coding in the Response Body.....	12
Functions.....	12
Access Functions.....	13
info.....	13
get.....	17
docGet.....	18
create.....	21
HTTP PUT.....	23
HTTP-POST multipart/form-data.....	23
mCreate.....	24
append.....	26
update.....	28
HTTP PUT.....	29
HTTP-POST multipart/form-data.....	29
delete.....	29
search.....	30
attrSearch.....	32
Administration Functions.....	37
putCert.....	37
serverInfo.....	38
Error Codes.....	40
SAP Cache Server.....	41
Appendix.....	41
Parameters and Keywords.....	41
Migrating Existing Archives.....	45

SAP Content Server HTTP Interface

This document describes the new **SAP Content Server HTTP Interface**. The current interface version is 4.7

The aim of this interface is that only general industry standards, such as HTTP and BAPIs, should be used in communication with external storage systems (content servers).

The HTTP Content Server Interface can be certified. As per the interface version 4.6 or 0046 there are developments to the SAP Cache Server (see [SAP Cache Server \[page 41\]](#))

To allow you to use scanning feature in the SAP Content Server, the Content Server component of the SAP Content Server HTTP Interface has been extended to 4.7 or 0047 from the previous version (version 4.5 or 4.6).

This version 4.7 has a new parameter added in the URL called **scanPerformed** which indicates whether the content needs to be scanned by SAP Content Server.

The Functions which are changed as per this new interface 4.7 are Create, Mcreate, Append and Update (see [HTTP_PUT \[page 23\]](#) for an example on new format of URL with version 0047).

In the absence of this parameter **scanPerformed** in the URL, the interface version 4.5 or 4.6 would be retained.

This version 4.7 also has a change in the function "attrsearch" where the separator for differentiating between the search patterns would be "_" instead of a "#". (see attrsearch example [page 36]).

Introduction

Definition of Terms

For the purposes of this section, a **document** comprises **administrative data** and **content**.

Administrative data identifies and describes a document.

The **content** of a document refers to closed datasets. The administrative data identifies and describes the content. A single closed dataset is known as a **content unit**.

In SAP terminology, a **content server** is any server that manages content. A content server may be a database, a fileserver, an SAP System or an external archive.

The data administration terms **content repository**, **document header**, and **component** are of particular importance in identifying documents.

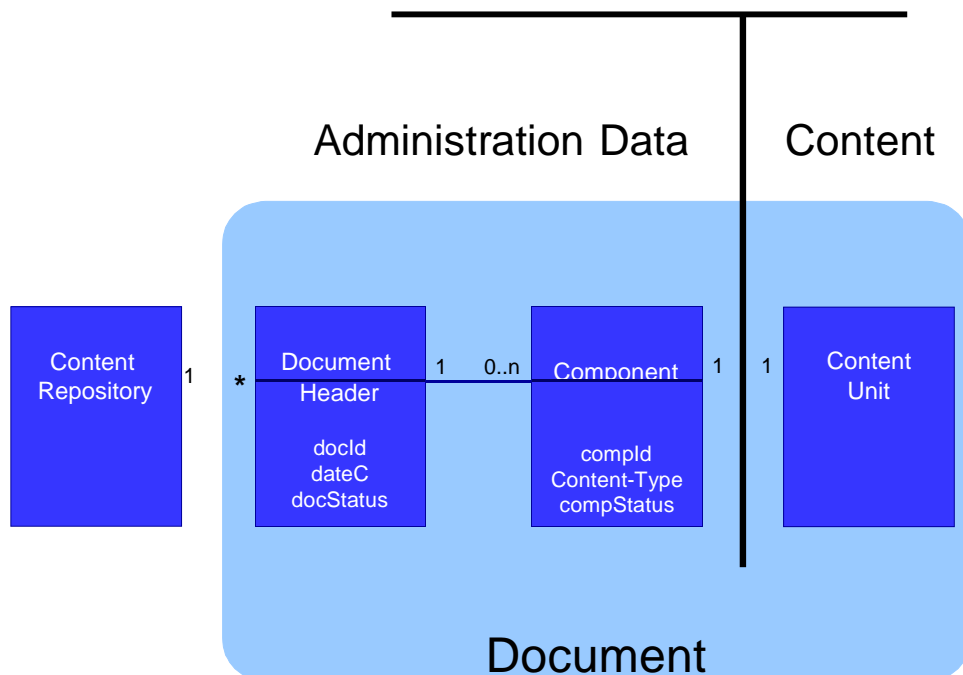
A **content repository** is the administrative entity that accesses the logical storage space for documents on a content server. Several content repositories can exist on one content server. A content repository is identified by the parameter **contRep**

The **document header** is an administrative entity comprising several components. It is identified by the parameter **docId**. A document header is assigned to one particular content repository.

A **component** represents one particular content unit. A component is assigned to one particular document header, and is identified by the parameter **compId**

The **scanPerformed** represents whether the content has to be scanned by SAP Content Server before storing.

The relationship between content repository, document header, component, and content is illustrated in the diagram below:



It can happen that a document with one component is created, and that this component is later deleted. This results in an 'empty' document, that is, a document with no components. In order to avoid possible contradictions in this documentation, we will assume that empty documents can occur. 'Empty' components can also occur, for example, if a file has 0 bytes.

The combination `contRep/docId` is the **unique address** of a document header.

The combination `contRep/docId/compId` is the **unique address** of a component.

In certain circumstances, documents require a **degree of protection**. This means that functions executed on the document must be authenticated. For each document header, you can define whether authorization is necessary for particular functions. However, this information is not defined in the document header for each function of the HTTP Content Server Interface, but via **access modes**. Access modes are discrete groups of SAP Content Server HTTP interface functions.

The new interface is designed in such a way that the client R/3 System always initiates communication. The content server being addressed by the R/3 System is always only a server and never a client, which means that it never instigates communication with the R/3 System.

HyperText Transfer Protocol (HTTP) is a communication process typically used to access objects on the **World Wide Web (WWW)**. The W3C (WWW Consortium, <http://www.w3c.org>) is currently further developing this protocol. There are two versions of the HTTP protocol, HTTP/1.1 and HTTP/1.0, and either can be used for the communication process. Request for Comment (RFC) 2068 specifies the protocol HTTP/1.1. HTTP/1.1 contains more specific regulations than HTTP/1.0 (RFC 1945), with the intention of making implementations of the protocol more reliable.

HyperText-Markup-Language (HTML) is a standard format and description language for WWW pages.

Uniform Resource Locators (URLs), see RFC 2396) are a standardized mechanism used to address uniquely defined objects on the WWW. As well as serving as addresses, URLs can also contain functions and parameters that can be interpreted by the object being addressed.

UTC (**Universal Time Coordinated**) is used for all expressions of time in this specification.



The following rules apply to the spelling of functions, parameters and key words in this section: All terms defined in the Hypertext Transfer Protocol HTTP/1.1 (RFC 2068) are used here (for example, **Content-Type**). The terminology of the HTTP/1.0 (RFC 1945) protocol is also used.

Terms specific to this interface are not capitalized if they consist of one syntactical term (for example, **info**). A combination of lowercase and uppercase is used if a term consists of more than one syntactical term (for example, **contRep**).

Implementation

The HTTP protocol is used for communication with content servers. Servers and documents are addressed using URLs, and data is transferred in the request body or in the response body.

The URL specifies which function is executed with a document, that is, whether the document is to be transferred from the server to the client (**get**), whether information about the document is to be included (**info**), and whether a new document is to be created (**create**). The necessary parameters for these functions are also part of the URL.

This documentation describes the URL syntax and the semantics for the various functions.

Security

General security and secure data transfer are central aspects of the Content Server Interface. It is important to note the following points in relation to the Content Server Interface:

It is assumed that all required authorization checks in the SAP System are carried out.

To ensure that users cannot circumvent these authorization checks when accessing the Content Server, a public/private key procedure (see also [Public Key Technology \[external\]](#)) is used.

Public and private keys are R/3-specific, not user-specific.

The security concept of the Content Server Interface is based around the fact that the SAP System public key is stored on the Content Server. **putCert** is the command used to put the key onto the server. The Content Server uses the public key to check URIs and signatures (see also [putCert \[page 37\]](#)).



For further information, see [Secure Store & Forward / Digital Signatures \[external\]](#).

secKey

secKey ensures that a URL cannot be changed after it has been generated by the R/3 System. This ensures that access to the document is protected and that access protection is managed in the R/3 System. The **secKey** does not protect the document content. The following parameters are always signed (that is, authenticated by means of a digital signature) in **secKey**:

contRep	Content Repository
accessMode	Access mode
authId	Client ID
expiration	Expiry time (UTC)

authId must uniquely identify the client (for example, the R/3 System).

The UTC expiry time is written in the format `yyyymmddhhmmss`. If the expiry time is exceeded, the content server must report HTTP status code 401 to the client.

If a **secKey** is transferred with the URL, the parameters **accessMode**, **authId** and **expiration** must also be transferred. These parameters need not be transferred if **secKey** is not transferred.

Also, other parameters have to be signed. These additional parameters depend on the particular function and are specified in the function description. The name of the function itself is not signed. The parameters to be signed can appear in the URL in any order. However, the order in which the parameters are transferred to the signing module must be the same as the order in the URL, so that the signature can be authenticated properly.

The **secKey** for the chosen procedure is about 500 bytes long.

The parameters to be signed for a particular function are specified in the function definition. They are specified in the last column of the parameter table. Obviously, optional parameters can only be signed if they are used. ^{s-mandatory} parameters must appear in the URL if a signature is used, and are always signed. If no signature is used, these parameters are not evaluated.

The URL parameters to be signed are referred to in this section as the ^{message}. The message is used to determine a hash value. The parameters must be kept in the same order so that the hash value can be calculated. The hash or message digest is a one-way function, that is, it cannot be reversed. Using the sender's private key, the SAP Secure Store & Forward (SSF) module uses the Digital Signature Standard (DSS) to digitally sign the hash value according to PKCS#. The digital signature is transferred in the URL in the parameter **secKey**, as described above.

Once the digital signature has been created, the URL parameters are protected and cannot be tampered with. However, they are not encoded. All recipients can check the URL parameters using the sender's public key. Any changes would therefore be detected. This ensures that an activity on the content server can only be started if the URL transferred has not been tampered with.

Using the sender's public key, the content server then regenerates the message digest from the transferred URL. Likewise, it then forms a hash from the message (the order of the parameters in the URL is important here) and compares the two hashes (the message hash and the hash generated by the sender). If the two hashes match, the URL has been transferred intact between the SAP System and the content server.



The library for checking signatures can be obtained from SAP AG. Because the standard format PKCS#7 was used for the signature, other products can also be used for decoding.

Summary of Technical Information

Format of digital signature:	PKCS#7 "signed data"
Public key procedure:	DSS
Key length:	512 – 1024 bits
Public exponent:	$2^{16} + 1$
Public key format:	X.509 v3 certificate
MD (message digest) algorithm:	MD5 or RIPEMD-160

Degree of Protection / Right of Access

The degree of protection of a document is specified when the document is created and stored. When a document is accessed, the server establishes what functions a user may execute on this document. Similar functions are grouped together. The groups are called access modes. They are listed in the following table:

Access Mode	Abbreviation
Read	r
Create	c
Change	u
Delete	d



The degree of protection applies to all components of the document. If the access mode is "change", any component of a document may be deleted.

The access mode must be specified in the HTTP request as a parameter (`accessMode`). A combination of access modes can be specified, for example, `ud`. A `secKey` confirms the right of access. In the descriptions of individual functions, the corresponding access mode is specified. When a document is accessed, the content server checks whether the `secKey` should be checked, that is, whether a function of the document is protected, and if so, what degree of protection it has. It therefore makes sense that any user may read documents, while only certain users may change them. In this case, read protection is deactivated (no `secKey` is required). For write and delete access, however, a `secKey` must be transferred. The fact that the `secKey` can only be generated by the R/3 System ensures that an access protection check based on the R/3 authorization concept is carried out.

The degree of protection of a document is defined when the document is created. To do this, use the parameter `docProt`.



Degree of Protection	Description
<code>docProt=</code>	No access restrictions
<code>docProt=du</code>	Only signed (that is, authenticated) URLs may delete or update documents. The <code>accessMode</code> must have at least a <code>d</code> for delete operations, and at least a <code>u</code> for update Operations. Read operations do not require any signature.



You may transfer a number of access types, for example, `accessMode=rd` in a read operation. This can be useful in certain situations. For example, if a `get` URL with `accessMode=rd` and a corresponding signature is transferred to a client program, the client has not only read permission but also delete permission for the document. To use the URL for deleting, simply replace the `get` command with the `delete` command, and do not transfer the `compId`. The same parameters are signed for both `get` and `delete`, so the signature remains valid. Because the `accessMode` contains a `d`, then, in this example it is possible to read and delete the document using the same signature.

Based on the access type of an operation and the degree of protection a document has, the Content Server decides whether it has to check the `secKey`. If the Content Server decides that no check is necessary, all s-mandatory parameters become obsolete, in other words, they do not need to be checked.



However, these parameters can be checked, if they are transferred accidentally, for example. However, this does not provide any extra security and is therefore superfluous, especially in the case of operations with no degree of protection, as the absence of an authorization check enhances system performance.

The parameter `docProt` is optional, but is usually transferred if the URL is not signed. If neither the Content Server nor the R/3 System uses a signature, this has no effect on the degree of protection, which is set for documents when they are created.

If the parameter `docProt` is not transferred, the default setting on the server is used. The content server default is set when the system is being implemented, and its value is entirely at the discretion of the system administrator.

If the R/3 System does transfer the `docProt` parameter, the R/3 System assumes that the maximum degree of protection applies for all access attempts on the relevant documents, and uses corresponding signed URLs.



The signature can be deactivated in the R/3 System only if it is also deactivated on the Content Server. In live systems, however, you should use signatures.

For all access modes, the Content Server must allow the system administrator to set as default whether a `secKey` must be specified or not. This server default can, however, be overwritten in the URL for the functions `create` and `mCreate`. If no degree of protection is specified, the server default is used.

Old data and documents that were stored in the Content Server without the use of the HTTP interface have the highest degree of protection, that is, all access attempts must be authenticated.

Syntax

General

The URL syntax is:

`http://servername:port/script?command¶meters`

The `servername` is the name of the server that is accessed, and `port` (optional) is a TCP/IP port that can be used to address the server. `script` is the name of the program used to access the content server. This could be a DLL, a CGI script or an Active Server Page (ASP). The object is created by the content server provider. A `command` must exist, followed by one or more `parameters`.



There must not be any blank spaces in the URL.



`http://pswdf009:1080/ContentServer/ContentServer.dll?get&pVersion=0046&contRep=K1&docId=361A524A3ECB5459E0000800099245EC`

URL Encoding

The structure of URLs is described in RFC 2396. RFC 1738 specifies which character set may be used for a URL and how characters not in this set should be encoded.

Only characters from an ASCII character set may be used in a URL (0x00 - 0x7F). The characters 0x00 – 0x1F and 0x7F must be encoded. If they must appear in the URL, a '%' (percentage sign) followed by the hexadecimal representation of the character should be used.



A line feed (0x0A) is represented as %0A in a URL.

Unsafe Characters

'Unsafe' characters must also be encoded in the way described above. These characters are: space, <, >, ", #, %, {, }, |, \, ^, ~, [,], ` . These characters are considered unsafe either because they execute special functions in the URL, or because they could be interpreted as special characters during transfer.

Reserved Characters

There are also 'reserved' characters: :, /, ?, :, @, =, &.

Reserved characters must also be encoded.

Transferring Binary Data

Problems can occur if binary data is to be transferred in a URL. This is the case when using the Content Server Interface, since the **secKey** consists of binary data. The ASCII character set must first be encoded. Base64 encoding (RFC 1521) must be used for the Content Server Interface.

Example

1. Compiling the URL

```
http://pswdf009:1080/ContentServer/ContentServer.dll?get&pVersion=0046&contRep=K1&docId=361A524A3ECB5459E0000800099245EC&accessMode=r&authId=pawdf054_BCE_26&expiration=19981104091537
```

2. Generating the secKey

The **secKey** is made up of the encoded parameters. The parameters to be signed are specified in the function definition.

In this example (**get** function) the parameters are:

```
ContRep = K1
DocId = 361A524A3ECB5459E0000800099245EC
AccessMode = r
AuthId = pawdf054_BCE_26
Expiration = 19981104091537
```

In the next step, the parameter values are summarized to form a message (without separators), in accordance with the sequence in the URL:

```
K1361A524A3ECB5459E0000800099245ECrpawdf054_BCE_2619981104091537
```

The message is used to form the hash from which the **SecKey** is calculated. In this example, arbitrary values are chosen for the **secKey**, for the sake of clarity.

The **secKey** has the following values: 0x83, 0x70, 0x21, 0x42.

3. Encoding the secKey in the ASCII character set

Base64 must always be used to encode the **secKey**.

```
0x83, 0x70, 0x21, 0x42 -> g3AhQg==
```

4. Encoding the URL in accordance with URL character set limitations

Characters may need to be encoded, as in this example:

g3AhQg== -> g3AhQg%3D%3D

The following URL is generated:

```
http://pswdf009:1080/ContentServer/ContentServer.dll?get&pVersion=0046&contRep=K1&docId=361A524A3ECB5459E0000800099245EC&accessMode=r&authId=pawdf054_BCE_26&expiration=19981104091537&secKey=g3AhQg%3D%3D
```

Coding in the Response Body

Many of the functions described in this section return information within the response body. If the information is returned in ASCII format, the lines always consist of key/value pairs separated by a semicolon, as follows:

```
key1="value1";key2="value2";...keyn="value2";CRLF
```

Only printable ASCII characters may be used. If a value contains an inverted comma, this must be entered in addition to the inverted commas already inserted around the value.

Functions

This section describes the available functions and their parameters. The effect of each function, its parameters, and an example are given.

Effect

'Effect' describes the executed function. The individual parameters are also explained.

Default

'Default' describes the effect of transferring only the mandatory parameters of a multi-parameter function.

Access Mode

'Access mode' specifies the access mode for the function.

Client Server

'Client Server' lists and specifies as optional or mandatory the parameters that are transferred from the client to the server. 's-mandatory' means that the relevant parameter must only be specified if a **secKey** is transferred. Client Server also defines the HTTP request type, and the way in which the parameters are to be coded in the URL or the body.

Example

'Example' gives an example of the executed function, using example parameters. Line breaks are included in the examples purely as an aid to legibility. The actual URLs do not contain any line breaks.

Server Client

'Server Client' defines the structure of the HTML response. HTML responses are generated by the server and are then sent to the client.

The HTTP status codes specific to the content server are also listed in this section. If no security key is entered, this may cause errors, for example, error 401 ('unauthorized'). A wrongly addressed document can cause error 404 ('not found'). If an error occurs, the content server must also deliver an ASCII string describing the error. The error must be entered in the header field **x-**

ErrorDescription.

Function Overview

Command	Effect	Access mode
info	Retrieve information about the document	r
get	Fetch (within a range) a content unit of a component	r
docGet	Fetch the entire content of a document	r
create	Create a new document	c
mCreate	Creates a number of new documents	c
append	Append data to a content unit	u
update	Modify an existing document	u
delete	Delete a document or a component	d
search	Search for a text pattern within a content unit	r
attrSearch	Search for one or more attributes within a document (search within a print list)	r
putCert	Transfer client (for example, the SAP System) certificate	-
serverInfo	Retrieve information about the content server and the corresponding content repositories	-

Access Functions

info

Effect

The function **info** retrieves document information. In response to **info**, the server sends the document header information and information on all components. If you require information on only one component, specify a **compId**. If you do so, the command **info** has the same effect as the command **docGet**, except that no component data is transferred with **info**.

Using **resultAs**, you can specify the format in which the information is to be provided. Return values can be provided in ASCII format, which means they can be easily parsed, or in an HTML file. The use of **resultAs** is optional. If you do not use it, the standard format (**ascii**) is used.

If and the function is executed successfully, the data is transferred as an entity body in **multipart/form-data** format (see RFC 1867) as a response to an **HTTP-GET-Request**.

Default

Standard information about the document header and the components of the addressed document is returned in ASCII format.

Access Mode

Read (r)

Client Server

The client sends an **HTTP-GET-Request**. The URL contains the following parameters:

Parameter	Optional/Mandatory	Default	Sign
contRep	mandatory		X
docId	mandatory		X
compId	optional		
pVersion	mandatory		
resultAs	optional	ascii	
accessMode	s-mandatory		X
authId	s-mandatory		X
expiration	s-mandatory		X
secKey	optional		



s-mandatory means that this parameter must only be specified if the URL is signed.

Example

```
http://pswdf009:1080/ContentServer/ContentServer.dll?info&pVersion=0046&contRep=K1&docId=361A524A3ECB5459E0000800099245EC
```

This example is a request for information about the document header and all the document components.

Server Client

The server answers the request with a response. The response status code indicates the outcome of the call.

HTTP Status Code	Meaning
200 (OK)	OK, information sent
400 (bad request)	Unknown function or unknown parameter
401 (unauthorized)	Breach of security
404 (not found)	Document or component not found
409 (conflict)	Administrative data inaccessible
500 (internal server error)	Internal error in content server

The response header contains the following information about the document:

Keyword	Format	Meaning
Content-Type	String	Content type (if known)
boundary	String	Separator between individual components
Content-Length	Integer string	Entire length of the body actually transferred
X-dateC	YYYY-MM-DD	Creation date (UTC)
X-timeC	HH:MM:SS	Creation time

X-dateM	YYYY-MM-DD	Last changed on [date] (UTC)
X-timeM	HH:MM:SS	Last changed at [time] (UTC)
X-numberComps	Integer string	Number of components
X-contentRep	String	Content repository
X-docId	String	Document ID
X-docStatus	String	Status
X-pVersion	String	Version

Each time the function is called, all document header information is provided. If no particular component is addressed, information on all components is made provided. If you require information on only one component, you can restrict the output to this component by specifying the `compId`. The following combinations are possible:

docId=ID, compId=ID :

Provides information about the document header and one component.

docId=ID :

Provides information about the document header and all components of the document.

The component header contains the following information about the component:

Keyword	Format	Meaning
Content-Type	String	Content-Type (if known)
charset	String	Character set (if known)
version	String	Application version used to create the content of the component
Content-Length	Integer string	Actual body size in the response, always 0
X-Content-Length	Integer string	Size of the component in bytes
X-compId	String	Component ID
X-compDateC	YYYY-MM-DD	Creation date (UTC)
X-compTimeC	HH:MM:SS	Creation time
X-compDateM	YYYY-MM-DD	Last changed on [date] (UTC)
X-compTimeM	HH:MM:SS	Last changed at [time] (UTC)
X-compStatus	String	Component status
X-pVersion	String	Interface version

The parameter `resultAs` controls coding. There are two methods of coding the results in the response body. These methods are explained below.

1. `resultAs=ascii` (default)

The server sends a response in *multipart/form-data* format (see RFC 1867). The total length of the body is specified by the parameter `Content-Length` in the response header. The individual parts of the response body are separated by a boundary specified in the response header. Each part represents one component. Each component has a component header and a component body. These have the length 0 because no component data is transferred (unlike the `docGet`

command). Therefore, the component parameter `Content-Length` is always set to 0. If you want to set the component length, you can do so using the parameter `X-Content-Length`. If the `charset` of a component is known, it must be transferred as a `Content-Type` parameter. Likewise, the parameter `version` (that is, the version number of the application used to create the component content (see [Parameters and Key Words \[page 41\]](#))) for a component, if known, must be transferred as a `Content-Type` parameter.



```

HTTP/1.1 200 (OK)
Server: Microsoft-IIS/4.0
Date: Wed, 04 Nov 1998 07:41:03 GMT
Content-Type: multipart/form-data; boundary=A495ukjfasdfddrg4hztzu...
Content-Length: 32413
X-dateC: 1998-10-07
X-timeC: 07:55:57
X-dateM: 1998-10-07
X-timeM: 07:55:57
X-contentRep: K1
X-numberComps: 2
X-docId: ID
X-docStatus: online
X-pVersion: 0046

--A495ukjfasdfddrg4hztzu898aA0jklmAxcvla12319981147528895
  Content-Type: application/x-alf; charset=
  Content-Length: 0
  X-compId: descr
  X-Content-Length: 2591
  X-compDateC: 1998-10-07
  X-compTimeC: 07:55:57
  X-compDateM: 1998-10-07
  X-compTimeM: 07:55:57
  X-compStatus: online
  X-pVersion: 0046

--A495ukjfasdfddrg4hztzu898aA0jklmAxcvla12319981147528895
  Content-Type: application/x-alf; charset=
  Content-Length: 0
  X-compId: data
  X-Content-Length: 29213
  X-compDateC: 1998-10-07
  X-compTimeC: 07:55:57
  X-compDateM: 1998-10-07
  X-compTimeM: 07:55:57
  X-compStatus: online
  X-pVersion: 0046
--A495ukjfasdfddrg4hztzu898aA0jklmAxcvla12319981147528895--

```

If the `info` command is executed on an empty document, the response body will contain something like the following:

```

--A495ukjfasdfddrg4hztzu898aA0jklmAxcvla1231999102562159269
--A495ukjfasdfddrg4hztzu898aA0jklmAxcvla1231999102562159269--

```

2. `resultAs=html`

If `resultAs=html` is set, the server sends an HTML page. The structure of the HTML page is not specified, which means that graphical elements can be used as much as required.

get

Effect

A content unit of a component or a range within a content unit is retrieved from the content repository. The parameters **ContRep**, **docId** and **compId** describe the component. **fromOffset** and **toOffset** describe the range of the content unit.

If the function is executed successfully, the content unit is transferred from the server to the client as an entity body in the response to an HTTP GET request.

Default

If no **compId** is specified, the following conditions are tested in the appropriate order:

1. If the component "data" exists, this component is returned.
2. If the component "data1" exists, this component is returned.

If an incorrect **compId**, or no **compId**, was specified, and if neither of the conditions above is fulfilled, the function returns Error 404 (not found).

Access Mode

Read (r)

Client Server

The client sends an **HTTP-GET-Request**. The URL contains the following parameters:

Parameter	Optional/Mandatory	Default	Sign
contRep	mandatory		X
docId	mandatory		X
compId	optional	see above	
pVersion	mandatory		
fromOffset	optional	0	
toOffset	optional	-1	
accessMode	s-mandatory		X
authId	s-mandatory		X
expiration	s-mandatory		X
secKey	optional		



s-mandatory means that this parameter must only be specified if the URL is signed.

Example

```
http://pswdf009:1080/ContentServer/ContentServer.dll?get&pVersion=0046&contRep=K1&docId=361A524A3ECB5459E0000800099245EC&compId=data
```

This requests the document component "data".

Server Client

The server answers the request with a response. The status code of the response indicates the outcome of the call, as shown in the following table:

HTTP Status Code	Meaning
200 (OK)	OK, content unit of component is transferred
400 (bad request)	Unknown function or unknown parameter
401 (unauthorized)	Security breach
404 (not found)	Document or component not found
409 (conflict)	Document or component inaccessible
500 (Internal Server Error)	Internal error on content server

The response header contains the following standard information about the document:

Keyword	Meaning
Content type	Content type
charset	The character set of the component (as a <code>contenttype</code> parameter)
version	The version of the component (as a <code>contenttype</code> parameter).
Content length	Length of document

The response 'content type' depends on the content type of the component requested. If the `charset` of a component is known, it must be transferred as a `Content-Type` parameter.

Likewise, the parameter `version` (that is, the version number of the application used to create the component content (see [Parameters and Key Words \(page 4\)](#))) for a component, if known, must be transferred as a `Content-Type` parameter.

The content unit (or range within the content unit) of the component is transferred in the response body.

docGet

Effect

The entire content of a document is retrieved from the content repository.

If an incorrect `docId` was specified, error 404 (not found) occurs. If the function is executed successfully, the data is transferred as an entity body in *multipart/form-data* format (see RFC 1867) as a response to an `HTTPGET-Request`.

Default

Access Mode

Read (r)

Client Server

The client sends an `HTTP-GET-Request`.

The URL contains the following parameters:

Parameter	Optional/Mandatory	Default	Sign
contRep	mandatory		X
docId	mandatory		X
pVersion	mandatory		
accessMode	s-mandatory		X
authId	s-mandatory		X
expiration	s-mandatory		X
secKey	optional		



s-mandatory means that this parameter must only be specified if the URL is signed.

Example

```
http://pswdf009:1080/ContentServer/ContentServer.dll?docGet&pVersion=0046&contRep=K1&docId=361A524A3ECB5459E0000800099245EC
```

This transfers the entire content of a document to the client.

Server Client

The server answers the request with a response. The response status code indicates the outcome of the call.

HTTP Status Code	Meaning
200 (OK)	OK, document is transferred
400 (bad request)	Unknown function or unknown parameter
401 (unauthorized)	Security breach
404 (not found)	Document or component not found
409 (conflict)	Document or component inaccessible
500 (Internal Server Error)	Internal error on content server

The server sends a response in **multipart/form-data** format (see RFC 1867). The individual parts of the response body are separated by a boundary specified in the response header. In contrast to the **info** command, when **docGet** is used, components are actually transferred and the Length of the transferred components is specified in the field **Content-Length** of the relevant component, that is, **Content-Length** and **X-Content-Length** have identical values.

The response header contains the following information about the document:

Keyword	Format	Meaning
Content-Type	String	Content type, always multipart/form-data
boundary	String	Separator between individual components
Content-Length	Integer string	Entire length of the body actually transferred
X-dateC	YYYY-MM-DD	Creation date (UTC)
X-timeC	HH:MM:SS	Creation time
X-dateM	YYYY-MM-DD	Last changed on [date] (UTC)
X-timeM	HH:MM:SS	Last changed at [time] (UTC)
X-numComps	Integer string	Number of components
X-contRep	String	Content repository
X-docId	String	Document ID
X-docStatus	String	Status
X-pVersion	String	Version

The component header contains the following information about the component:

Keyword	Format	Meaning
Content-Type	String	Content type (if known)
charset	String	Character set (if known)
version	String	
Content-Length	Integer string	Size of the component in bytes
X-Content-Length	Integer string	Size of the component in bytes
X-compId	String	Component ID
X-compdateC	YYYY-MM-DD	Creation date (UTC)
X-compTimeC	HH:MM:SS	Creation time
X-compDateM	YYYY-MM-DD	Last changed on [date] (UTC)
X-compTimeM	HH:MM:SS	Last changed at [time] (UTC)
X-compStatus	String	Component status
X-pVersion	String	Interface version

If the **charset** of a component is known, it must be transferred as a **Content-Type** parameter. Likewise, the parameter **version** (that is, the version number of the application used to create the component content (see [Parameters and Key Words \[page 41\]](#))) for a component, if known, must be transferred as a **Content-Type** parameter.

Example:

HTTP/1.1 200 (OK)

Server: Microsoft-IIS/4.0

Date: Wed, 04 Nov 1998 07:41:03 GMT

Content-Type: multipart/form-data; boundary=A495ukjfasdfddrg4hztzu...

```

...somemoreheaderinformation...
  Content-Length: 32413
X-dateC: 1998-10-07
X-timeC: 07:55:57
X-dateM: 1998-10-07
X-timeM: 07:55:57
X-contRep: K1
X-numComps: 2
X-docId: ID
X-docStatus: online
X-pVersion: 0046

--A495ukjfasdfddrg4hztzu898aA0jklmAxcv1a12319981147528895
  Content-Type: application/x-alf; charset=
  Content-Length: 2591
  X-compId: descr
  X-Content-Length: 2591
  X-compDateC: 1998-10-07
  X-compTimeC: 07:55:57
  X-compDateM: 1998-10-07
  X-compTimeM: 07:55:57
  X-compStatus: online
  X-pVersion: 0046

...componentdata...
--A495ukjfasdfddrg4hztzu898aA0jklmAxcv1a12319981147528895

Content-Type: application/x-alf; charset=
Content-Length: 29313
X-compId: data
X-Content-Length: 29213
X-compDateC: 1998-10-07
X-compTimeC: 07:55:57
X-compDateM: 1998-10-07
X-compTimeM: 07:55:57
X-compStatus: online
X-compStatus: online
X-pVersion: 0046

```

```

...componentdata...
--A495ukjfasdfddrg4hztzu898aA0jklmAxcv1a12319981147528895--

```

If the `docGet` command is executed on an empty document, the response body could contain the following, for example:

```

--A495ukjfasdfddrg4hztzu898aA0jklmAxcv1a1231999102562159269
--A495ukjfasdfddrg4hztzu898aA0jklmAxcv1a1231999102562159269--

```

create

Effect

This function creates new documents in the content repository with one or more components. The parameters `contRep`, `docId` and `compId` describe the component (see [Definition of Terms \[page 5\]](#)). If you try to create a document that already exists in the content repository, the function returns an error. (If you want to modify existing documents, use the functions `update` and `append`.)

The function can be called once with an HTTP PUT or POST (for further information, see [HTTP PUT \[page 23\]](#) and [HTTP POST multipart/form-data \[page 23\]](#)).

Default

A new document with the specified `docId` is created. One or more components are stored in the content repository. What degree of protection, if any, the document gets depends on the standard on the content server.

Access Mode

create (c)

Client Server

The following parameters exist:

Parameter	Optional/Mandatory	Default	Position (POST/PUT)	Sign (POST/PUT)
contRep	mandatory		URL/URL	X/X
compId	mandatory		body/URL	-/X
docId	mandatory		URL/URL	X/X
pVersion	mandatory		URL/URL	
Content-Type	optional		body/body	
charset	optional		body/body	
version	optional		body/body	
Content-Length	mandatory		Header-body/ Header-body	
docProt	optional	server setting	URL/URL	X/X
accessMode	s-mandatory		URL/URL	X/X
authId	s-mandatory		URL/URL	X/X
expiration	s-mandatory		URL/URL	X/X
secKey	optional		URL/URL	
scanPerformed	optional		URL/URL	X/X

For an HTTP POST, the `Content-Length` in the request header is the total length of the body and the `Content-Length` in each part header is the length of the individual content units. For an HTTP PUT, the `Content-Length` is always the total length of the body.

Note the difference between when the parameter `docProt` is not transferred at all, and when it is transferred, but with no value (`docProt=`). In the first case, the content server default for degree of protection is used. The second case specifies explicitly that no degree of protection is required.

s-mandatory means that this parameter must only be specified if the URL is signed.

The function can be executed in two ways. Either a single component is transferred using [an HTTP PUT \[page 23\]](#), or an an HTTP POST in `multipart/form-data` format (see [HTTP POST multipart/form-data \[page 23\]](#)) is used. In the case of the former, only a single component can be loaded onto the server. In the case of the latter, between 0 and n components can be transferred.

HTTP PUT

With HTTP PUT, all parameters are entered in accordance with the table in the section [create \[page 21\]](#).

- `http://pswdf009:1080/ContentServer/ContentServer.dll?create&pVersion=0046&contRep=K1&docId=4B7689654E73D21197E70060B0672A3C&compId=data&Content-Length=300`

As per the Interface 4.7 or 0047 with the additional parameter **scanPerformed**, the URL can be as below,

```
http://pswdf009:1080/ContentServer/ContentServer.dll?create&pVersion=0047&contRep=K1&docId=4B7689654E73D21197E70060B0672A3C&compId=data&Content-Length=300&scanPerformed=true
```

If the parameter **scanPerformed** is set as "true" in the URL, then it means that the scanning was performed by the ABAP layer and SAP Content Server would not do the scanning.

If the parameter **scanPerformed** is not present in the URL, and the URL contains the parameter **pVersion=0047** then it means that no scanning has been done by the ABAP layer, and scanning has to be done by SAP Content Server. This parameter is added to avoid performance problems which can arise due to double scanning at the ABAP layer as well as at the SAP Content Server. Hence, setting of this flag avoids scanning twice. Therefore, we recommend for third party Content Servers to have this as a functional requirement in case they would like to avoid performance issues.

The scanning feature is available only if the pVersion=0047. This feature is not available if the pVersion=0045 or pVersion=0046. This functionality is applicable only if the Virus Scan Integration is done at the Server layer. In case there is no Virus Scan integration done by the third party Content Server, then this double scanning would not be done. Only the scanning at the ABAP layer would be done depending on the settings at the ABAP Layer.

The document component "data" is stored. The component data is transferred as an entity body.

HTTP-POST multipart/form-data

The data is transferred as **HTTP-POST** and as **multipart/form-data**. The document header information is transferred in the URL. One or more components are transferred in the body. This version of the function is particularly suitable for transferring documents consisting of several components into the content repository, as a whole. The component information is specified in the header of each part; the data in the body.

In practice, this means that the URL contains the parameters **contRep docId, pVersion docProt accessMode authId expiration** and **secKey**. All the other parameters are in the body.

The request body is in **multipart/form-data** format. With this format, it is possible to transfer several independent parts to an HTTP content server. The individual parts have a header and a body and are in MIME format (RFC 2045, 2046). This MIME format enables several components to be transferred to the content server simultaneously. If an error occurs when storing a component, the entire action is cancelled.

The parameters **compId** and **Content-Type** are contained in the header of each part. The **CompId** is transferred in field **x-compId**. The component length is in the field **Content-Length**. The parameters **charset** and **version** can be appended to the **Content-Type**.

Example 1

```
http://pswdf009:1080/ContentServer/ContentServer.dll?create&pVersion=0046&contRep=K1&docId=4B7689654E73D21197E70060B0672A3C
```

A document consisting of one or more components is transferred in **multipart/form-data** format.

Document header

```
Content-Type:multipart/form-data;boundary= A495ukjfasdfddrg4hztzu898aA0jklm
...somemoreheaderinformation...
Content-Length:32413
```

Content part

```
--A495ukjfasdfddrg4hztzu898aA0jklm
X-compId:data
Content-Type:application/msword;charset=ISO-8859-1;version=6
Content-Length:4242
...4242BytesData...
--A495ukjfasdfddrg4hztzu898aA0jklm--
```

Example 2 (Create with 0 Components)

```
http://pswdf009:1080/ContentServer/ContentServer.dll?create&pVersion=0046&contRep=M1&docId=3810FF00804C257DE10000009B38FA09&docProt=ud&accessMode=c&authId=CN%3DKPR&expiration=19991025080635&secKey=MIIBlQYJKoZIhvcNA..
```

A document consisting of one or more components is transferred in **multipart / form-data** format.

Document header

```
Content-Type:multipart/form-data;boundary=KoZIhvcNAQcB
...somemoreheaderinformation...
Content-Length:38
```

Content part

```
--KoZIhvcNAQcB
--KoZIhvcNAQcB--
```

Server Client

The server answers the request with a response. The response status code indicates the outcome of the call.

HTTP Status Code	Meaning
201(created)	OK, document(s) created
400 (bad request)	Unknown function or unknown parameter
401 (unauthorized)	Breach of security
403 (forbidden)	Document already exists
500 (internal server error)	Internal error in content server

The content server must set the dates (**dateC** and **compDateC**) and the times (**timeC** and **compTimeC**) for creating components and the document.

mCreate

Effect

One or more documents each with one or more components are stored in the content repository. The function has a similar effect to that of several sequential **create** functions, but is significantly more efficient for signed URLs in storing many new documents at once.

Within an **mCreate** call, objects can only be stored in one and the same content repository. The content repository is described by the URL parameter **contRep**.

The individual components of the documents are transferred in a *multipart/form-data* entity body. Components of the same document must be transferred one after the other so that transfer of a document can be assumed to be complete, as soon as a component from a different document begins.

The parameter **docId** is absolutely necessary for all components (each multipart-part) and is entered as the headerfield **"X-docId"**

Storage of one document is performed within one transaction, but not the storage of all documents transferred in one **mCreate** call.

Access mode

Create (c)

Client Server

The client sends an **HTTP-POST-Request** . The following parameters exist:


Parameter	Header Field in Body	Optional/Mandatory	Default	Position	Sign
contRep		mandatory		URL	X
compId	"X-compId"	mandatory		body	
docId	"X-docId"	mandatory		body (1. docId also in URL)	X(1.docId)
pVersion		mandatory		URL	
Content-Type		optional		body	
charset		optional		body	
version		optional		body	
Content-Length		mandatory		body	
docProt		optional	server setting	URL	X
accessMode		s-mandatory		URL	X
authId		s-mandatory		URL	X
expiration		s-mandatory		URL	X
secKey		optional		URL	
scanPerformed		optional		URL	X

s-mandatory means that this parameter must only be specified if the URL is signed.

The parameters for which the request body is specified as position are transferred in the header field of the multipart part of the corresponding components. For special parameters in this interface, the name of the header field is in column 2 of the table.

Server Client

The server answers the request with a response. The response status code indicates the outcome of the call. A distinction is made between general status codes, which describe the success of the call as a whole, and specific status codes, which document the creation of individual documents.

General HTTP Status Codes	Meaning
201(created)	OK, all documents were created
250 (missing documents created)	OK, all missing documents were created  In practice, this status can only occur when <code>mcreate</code> is called more than once.
400 (bad request)	Unknown function or unknown parameter
401 (unauthorized)	Breach of security
500 (internal server error)	Internal error in content server

Specific HTTP Status Codes	Meaning
201(created)	OK, document created
403 (forbidden)	Document already exists
500 (internal server error)	Internal error in content server

An ASCII text must be returned whether the function is executed successfully or whether an error occurs. The documents stored (HTTP status code 201) and/or not stored (HTTP status code 403) are specified in this text. The following format is used:

```
docId="string";retCode="integerstring";errorDescription="string";CRLF
```

The value of the parameter `retCode` is the corresponding specific HTTP status code.

As a summary, the response body contains the following standard information about each document:

Keyword	Format	Meaning
<code>docId</code>	string	Document ID
<code>retCode</code>	Integerstring	HTTP status code
<code>errorDescription</code>	string	Text explaining the error (optional)

append

Effect

Data is appended to a content unit of a component in the content repository. The parameters `ContRep`, `docId` and `compId` describe the component (see [Definition of Terms \[page 5\]](#)). The document, addressed and the corresponding component must exist.

Default

Data is appended to the content unit of the addressed component.

Access mode

change (u)

Client Server

The client sends an **HTTP PUT-Request**. The URL contains the following parameters:

Parameter	Optional/Mandatory	Default	Position	Sign
contRep	mandatory		URL	X
docId	mandatory		URL	X
compId	mandatory		URL	X
pVersion	mandatory		URL	
appendMode	s-mandatory		URL	X
authId	s-mandatory		URL	X
expiration	s-mandatory		URL	X
secKey	optional		URL	
scanPerformed	optional		URL	X

s-mandatory means that this parameter must only be specified if the URL is signed.

The data to be appended is transferred as an entity body.

Example

<http://pswdf009:1080/ContentServer/ContentServer.dll?append&pVersion=0045&contRep=K1&docId=4B7689654E73D21197E70060B0672A3C&compId=data&Content-Length=980>

Data transferred in the request body is appended to the content unit of the component "data" in the specified document.

Server Client

The server answers the request with a response. The response status code indicates the outcome of the call.

HTTP Status Code	Meaning
200 (OK)	OK, data appended
400 (bad request)	Unknown function or unknown parameter
401 (unauthorized)	Breach of security
404 (not found)	Document/component not found
409 (conflict)	Document/component not accessible
500 (Internal Server Error)	Internal error in content server

The content server must set the dates (**dateM** and **compDateM**) and the times (**timeM** and **compTimeM**) for changing components and the document.

update

Effect

One or more components of a document in the content repository are overwritten. The parameters **ContRep docId** and **compId** describe the component (see [Definition of Terms \[page 5\]](#)). The function is used to modify existing documents and components and can be called once with an [HTTP-PUT \[page 29\]](#) or [HTTP-POST \[page 29\]](#). The function can be called once with a HTTP-PUT or POST (see below for details).

The variant **HTTP-PUT** is used to create or overwrite a component of an existing document.

The variant **HTTP-POST** (multipart/form_data) is used to bring an entire document up to date. When this variant is used, the entire document is overwritten, not only individual components.

Default

One or more components are stored in the content repository. Protection is according to the standard set on the content server.

Access mode

change (u);

Client Server

The following parameters exist:

Parameter	Optional/mandatory	Default	Position (POST/PUT)	Sign (POST/PUT)
contRep	mandatory		URL/URL	X/X
compId	mandatory		body/URL	-/X
docId	mandatory		URL/URL	X/X
pVersion	mandatory		URL/URL	
Content-Type	optional		body/body	
charset	optional		body/body	
version	optional		body/body	
Content-Length	mandatory		body/body	
accessMode	s-mandatory		URL/URL	X/X
authId	s-mandatory		URL/URL	X/X
expiration	s-mandatory		URL/URL	X/X
secKey	optional		URL/URL	
scanPerformed	optional		URL/URL	X/X

s-mandatory means that this parameter must only be specified if the URL is signed.

The function can be executed in two ways. Either a single component can be transferred using an [HTTP-PUT \[page 29\]](#) or an [HTTP-POST \[page 29\]](#) in **multipart/form-data** format is used. If the former version (**HTTP-PUT**) is used, only one component at a time can be loaded onto the content server. There is no such restriction if the latter version (**HTTP-POST**) is used.

HTTP PUT

This variant of the function is used to recreate or overwrite an individual component of a document. For further details, see the command [create \[page 23\]](#).

HTTP-POST multipart/form-data

Similarly to the `create` function, this variant of the function is used to replace a complete document with all its components in the content repository at once. Document components not already in the content repository are created if necessary. Components in the content repository that are not transferred when the `update` function is executed are considered obsolete and deleted. The structure details of the request is the same as for the function [create \[page 23\]](#).

Server Client

The server answers the request with a response. The response status code indicates the outcome of the call.

HTTP Status Code	Meaning
200 (OK)	OK, document(s)/component(s) changed
400 (bad request)	Unknown function or unknown parameter
401 (unauthorized)	Breach of security
404 (not found)	Document/component not found
409 (conflict)	Document/component not accessible
500 (Internal Server Error)	Internal error in content server

The content server must set the dates (`dateM`, `compDateM` and `compDateC`) and the times (`timeM`, `compTimeM` and `compTimeC`) for changing components and the document.

delete

Effect

A component or an entire document is deleted. A document to be deleted is addressed via `contRep` and `docId`. The parameters `contRep`, `docId` and `compId` identify the component to be deleted.

Default

The document, including all administrative data (document header and components) and the content, is deleted completely.

Access mode

delete (d)

Client Server

The client sends an **HTTP-GET-Request**. The URL contains the following parameters:

Parameter	Optional/Mandatory	Default	Sign
contRep	mandatory		X
docId	mandatory		X
compId	optional	all components	X
pVersion	mandatory		
accessMode	s-mandatory		X
authId	s-mandatory		X
expiration	s-mandatory		X
secKey	optional		



s-mandatory means that this parameter must only be specified if the URL is signed.

Example

http://pswdf009:1080/ContentServer/ContentServer.dll?delete&pVersion=0047&contRep=K1&docId=4B7689654E73D21197E70060B0672A3C&compId=data

Component "data" is deleted in the named document.

Server Client

The server answers the request with a response. The response status code indicates the outcome of the call.

HTTP Status Code	Meaning
200 (OK)	OK, document/component(s) deleted
400 (bad request)	Unknown function or unknown parameter
401 (unauthorized)	Breach of security
404 (not found)	Document/component not found
409 (conflict)	Document/component not accessible
500 (Internal Server Error)	Internal error in content server

search

Effect

This function searches for a text pattern in the content unit of a component. The range of the search can be restricted. The search begins at the point specified by **fromOffset** and continues until the **toOffset** point. If **fromOffset > toOffset**, the function searches the component backwards.

A text pattern is found if the following conditions are met:

if **fromOffset** ≤ **toOffset**

the location of first character of the text found is greater than or equal to **fromOffset**

the location of the last character of the text found is smaller than or equal to **toOffset**

if **fromOffset** ≥ **toOffset**

the location of the last character of the text found is less than or equal to **fromOffset**

the location of the first character of the text found is greater than or equal to **toOffset**

The pattern contains the string searched for. The string can contain blank characters.

The number of result entries and up to **numResults** hits are returned as the result. A hit is the entry of the character position. The character position is the position of the found location in relation to the start of the document. The position of the first character of the text searched for is defined as the position of the found location, irrespective of the search direction.

Default

The pattern is searched for in the whole addressed component.

Access mode

Read (r)

Client Server

The client sends an **HTTP-GET-Request**. The URL contains the following parameters:

Parameter	Optional/Mandatory	Default	Sign
contRep	mandatory		X
docId	mandatory		X
pattern	mandatory		
compId	mandatory		
pVersion	mandatory		
caseSensitive	optional	n	
fromOffset	optional	0	
toOffset	optional	-1	
numResults	optional	1	
accessMode	s-mandatory		X
authId	s-mandatory		X
expiration	s-mandatory		X
secKey	optional		



s-mandatory means that this parameter must only be specified if the URL is signed.

Example

`http://pswdf009:1080/ContentServer/ContentServer.dll?search&pVersion=0046&contRep=K1&docId=4B7689654E73D21197E70060B0672A3C&compId=data&pattern=Manfred%20M%FC1ler&fromOffset=80`

A search for “Manfred Mueller” is carried out in the component data of the named object from Offset=80 to the end.

Server Client

The server answers the request with a response. The response status code indicates the outcome of the call.

HTTP Status Code	Meaning
200 (OK)	OK, component was searched
400 (bad request)	Unknown function or unknown parameter
401 (unauthorized)	Breach of security
404 (not found)	Document/component not found
409 (conflict)	Document/component not accessible
500 (internal server error)	Internal error in content server

The result of the search is the number of hits and the Offset for each hit. An ASCII string with the following structure is returned: `number;offset;offset;...`

There are no blank characters between the individual characters. There is a semicolon between the values and at the end.



```
2;122;222;
```

attrSearch

This function is used for attribute-based searches in print lists (attribute search). It is a prerequisite of this search that a print list has a description file (`compId=descr`) as well as a data file (`compId=data`). Unlike [search \[page 30\]](#) this is a specific search, which is carried out in the description file of a print list (`compId=descr`). Only the description file is relevant for the implementation of an attribute search.

Basic Principles

The structure of the description file is explained below with the aid of an example.



Content of a description file (extract; the periods stand for blank characters):

```
072DPRL
730DKEYclient ..... 03
730DKEYcompany_code ..... 35
730DKEYaccount_number ..... 87
730DKEYcustomer_name ..... 1525
73138DAIN00100010147119Broeselplc
211120DAIN001000020147129Obelixplc
11471DEPL
```


The description file consists of a sequence of lines (**index lines**). These index lines describe attributes of a range of the relevant data file.

An index line consists of:

Offset and Length in the data file:

Specification of the Offset and the Length of the range described (in bytes) relative to the start of the data file.

Record type

Type of line. The record type consists of four bytes. The following record types are used:

DPRL

Prolog

DKEY

Description of attributes

DAIN

Value of attributes

DEPL

Epilog

The various record types occur in the description file in the order specified here. The fact that the DAIN lines come after the DKEY lines is of particular note.

Only DKEY and DAIN lines are decisive for the attribute search. The DKEY lines specify the attributes and how they are stored. The DAIN lines specify the attribute values.

Parameter

Remaining content of the index line dependent on the record type.

The individual index lines are closed with linefeed (0x0A). The value for the "Offset in the data file" increases steadily within the description file.



Interpreted content of the description file (extract):

Offset and Length in the Data File		Record Type	Parameter			
0	72	DPRL				
73	0	DKEY	Client	0	3	
73	0	DKEY	Company code	3	5	
73	0	DKEY	Account number	87		
73	0	DKEY	Customer name	15	25	
73	138	DAIN	001	0001	01471 19	Bröse lplc
211	120	DAIN	001	0002	01471 29	Obeli xplc
...						
1147	1	DEPL				

The structure of the DKEY and DAIN lines is described in detail below.

DKEY lines (description of attributes)

Content	Length (in Bytes)
Offset in data file	Variable
Separator (space)	1
Length in data file	Variable
Separator (space)	1
Record type ("DKEY")	4
Attribute names	40
Offset in the DAIN line parameter	3
Length in the DAIN line parameter	3

The DKEY lines specify the names (attribute names) and the structure (Offset and Length in the DAIN line parameter) of the attributes that occur in the DAIN lines. The Offset position is counted starting with 0. Each DKEY line describes one particular attribute.

The values "Offset and Length in data file" are not relevant here.

DAIN lines (values of attributes)

Content	Length (in Bytes)
Offset in data file	Variable
Separator (space)	1
Length in data file	Variable
Separator (space)	1
Record type ("DAIN")	4
Parameter	Variable

Each DAIN line specifies the attribute value for a specific range of the data file. The DAIN line parameter consists of the attribute values corresponding to the standards in the DKEY lines. Blank characters at the end of the DAIN lines are irrelevant. If the DAIN line contains less data than is specified in the DKEY lines, the attributes must be filled with blank characters.

Here, the specifications "Offset and Length in data file" are relevant. They relate to the data file and specify the range for which the given attribute values are valid.



The content of the above example is as follows:

Description of attributes

Attribute Name	Offset in the DAIN Line Parameter	Length in the DAIN Line Parameter
Client	0	3
Company code	3	5
Account number	8	7
Customer name	15	25

Value of attributes

Offset in Data File	Length in Data File	Attribute Name	Attribute Value
73	138	Client	"001"
		Companycode	"00001"
		Accountnumber	"0147119"
		Customername	"Bröselplc"
211	120	Client	"001"
		Companycode	"00002"
		Accountnumber	"0147129"
		Customername	Obelixplc
...

Effect

This function is used for attribute-based searches in print lists. The parameters **ContRep** and **docId** describe the component. The parameter **pattern** specifies (as well as the pattern) the attribute to be searched for. The attribute is described by its Offset and Length. The pattern is made up of the Offset, followed by the character "+", followed by the Length, followed by the character "+", followed by the attribute value. If several attributes are to be searched for, the individual **patterns** should be separated by a "_" (change as per interface version 4.7).

The patterns can contain any characters. Unsafe and reserved characters are coded as normal here. This is also true for the separator "_" if this occurs in the pattern (see Coding in the URL [page 11]). If the symbol "_" occurs as a separator in between the 2 patterns, then it will not be coded.

The result of the attribute search is the values for "Offset and Length in the data file" in the DAIN lines, that correspond to the pattern and are in the search range.

For a pattern to be found, the following conditions must be fulfilled:

The value for the "Offset in the data file" in the DAIN line is within the search range.

The attribute values given in the **pattern** match those in the DAIN line. This means that each attribute value specified by the **pattern** must be contained fully in the corresponding attribute value in the DAIN line.

The number of result entries (number of appropriate DAIN lines), as well as the result entries themselves (values for "Offset and Length in the data file" in the DAIN line) are returned as the result. The values are separated by a semicolon. The result entries are arranged according to the search direction. Control of the search direction is as for the [search \[page 30\]](#) function. The number of results can be restricted by the parameter **numResults**.

Default

The pattern is searched for in the document addressed by the parameters **contRep** and **docId**. The **compId** is not specified in the call. The function always searches in the description file (**compId=descr**).

Access mode

Read (r)

Client Server

The client sends an **HTTP-GET-Request**.

The URL contains the following parameters:

Parameter	Optional/Mandatory	Default	Sign
contRep	mandatory		X
docId	mandatory		X
pattern	mandatory		
pVersion	mandatory		
caseSensitive	optional	n	
fromOffset	optional	0	
toOffset	optional	- 1	
numResults	optional	1	
accessMode	s-mandatory		X
authId	s-mandatory		X
expiration	s-mandatory		X
secKey	optional		



s-mandatory means that this parameter must only be specified if the URL is signed.

Example

```
http://pswdf009:1080/ContentServer/ContentServer.dll?attrSearch&pVersion=0046&contRep=K1&docId=361A524A3ECB5459E0000800099245EC&pattern=3+5+12345_15+25+GmbH&numResults=5
```

A search is run in a component for *12345* in the attribute with Offset 3 and Length 5 (in the example, this is the *company code*) and also for the content *GmbH* in the attribute with Offset 15 and Length 25 (in the example, this is the *customer name*). Up to 5 hits are found.



As can be seen from the example, `_` is not coded.

Server Client

The server answers the request with a response. The response status code indicates the outcome of the call.

HTTP Status Code	Meaning
200 (OK)	OK, component was searched
400 (bad request)	Unknown function or unknown parameter
401 (unauthorized)	Breach of security
404 (not found)	Document/component not found
409 (conflict)	Document/component not accessible
500 (internal server error)	Internal error in content server

The result of the search is the number of hits and the Offset and Length for each hit:
number;offset;length;...



```
2;73;138;211;120;
```

If an attribute search cannot be carried out properly because the values in the attribute do not match the standards in the DKEY lines (for example, attribute names wrong or attribute value too long in the **pattern**), status code 400 (bad request) is returned.

If, however, nothing is found, the status code is set to 200 (OK) and 0 is returned as the result.

Administration Functions

In the current HTTP Content Server Interface, only two administrative functions are defined: **putCert** and **serverInfo**. Further administration functions will be defined in later versions of the interface.

putCert

Effect

The client certificate is transferred. The system identifies itself via its authenticity (**authId**). The client certificate ([see secKey \[page 7\]](#)) is decoded in the message body and transferred in binary format.

For reasons of security, it is recommended that after the certificate has been transferred, manual action by an administrator is necessary before access is actually allowed. This could be a public key fingerprint check or any other plausibility check.

The logon procedure therefore consists of two steps:

- Certificate is transferred and entered in a central location

- Administrator allows access via a tool

The client can only access after the second step of this procedure. After the first step, the certificate is only created.

Access mode

-

Client Server

The client sends an **HTTP-PUT-Request**.

Parameter	Optional/Mandatory	Sign
authId	mandatory	
pVersion	mandatory	
contRep	mandatory	

The certificate is transferred in the request body, all the other parameters are transferred in the URL. The URL does not contain a `secKey`.

Server Client

The server answers the request with a response. The response status code indicates the outcome of the call.

HTTP Status Code	Meaning
200 (OK)	OK
400 (bad request)	Unknown function or unknown parameter
406 (not acceptable)	Certificate cannot be recognized
500 (Internal Server Error)	Internal error in content server

serverInfo

Effect

The function supplies information about the status of the [Content Server \[external\]](#) and the content [Repositories \[external\]](#) that it manages.

Default

The standard information is returned to the content server addressed and the content repositories that it manages. The results are given in ASCII format.

Access mode

Client Server

The client sends an `HTTP-GET-Request`. The following parameters exist:

Parameter	Optional/Mandatory	Default	Sign
contRep	optional	all repositories	
pVersion	mandatory		
resultAs	optional	ascii	

Example

```
http://pswdf009:1080/ContentServer/ContentServer.dll?serverInfo&pVersion=0046
```

The example requests information about the content server and all the content repositories it manages.

Server Client

The server answers the request with a response. The response status code indicates the outcome of the call.

HTTP Status Code	Meaning
200 (OK)	OK
400 (bad request)	Unknown function or unknown parameter
500 (internal server error)	Internal error in content server

The following information about the content server status is provided:

Keyword	Format	Meaning
serverStatus		Status of content server: [running stopped error]
serverVendorId		Vendor and software version
serverVersion		Version of server
serverBuild		Build of server
serverTime	HH:MM:SS	Content server time (UTC)
serverDate	YYYY-MM-DD	Content server date (UTC)
serverStatusDescription		Text describing server status
pVersion		Content server interface version

The following information about the status of each content repository is provided:

Keyword	Format	Meaning
contRep		Content repository
contRepDescription		Text describing content repository content
contRepStatus		Status of content repository: [running stopped error]
contRepStatusDescription		Text describing content repository status



The specified parameters are mandatory. The list of parameters is designed with a view to extension.

For each function call, all information about the content server is provided. If no content repository is addressed, information on all content repositories is provided. **ContRep** can be used to limit the content repository information to a single content repository.

There are two ways of coding the results in the response body. The parameter **resultAs** controls coding.

1. **resultAs=ascii (default)**

A pure ASCII-Text is returned. The information about the content server is at the start of the string, the information about the content repositories follows. The following format is used:

For the content server:

```
serverStatus="string";serverVendorId="string";serverTime="string";serverDate="string";serverErrorDescription="string";pVersion="0045";CRLF
```

For each content repository:

```
contRep="string";contRepDescription="string";contRepStatus="string";pVersion="0045";CRLF
```

If no value is entered, the value remains free.

```
contRepDescription="";contRepStatus="string";...
```

The order of the key words does not matter but there must not be any blank characters. The key words (together with their values) are separated from each other by a semicolon. The corresponding values are in quotation marks.

2. resultAs=html

If `resultAs=html` is set, the server sends an HTML page. The structure of the HTML page is not specified and graphical elements can be used freely.

Error Codes

An error that occurs when a function is executed can be recognized in the HTTP status code.

HTTP Status Code	Meaning	Used For
200 (OK)	OK, information or component was delivered, transferred, changed, appended, or deleted	info, get, docGet, update, append, delete, putCert, search, attrSearch
201(created)	OK, component(s) created (if <code>create</code> was used) OK, (all) document(s) created (if <code>mCreate</code> was used)	create, mCreate
250 (missing documents created)	OK, all missing documents were created	mCreate
400 (bad request)	Unknown function or unknown parameter	All functions
401 (unauthorized)	Security breach	info, get, docGet, create, update, append delete, mCreate, search, attrSearch
403 (forbidden)	Document or component already exists	create, mCreate
404 (not found)	Document, component, or content repository not found	info, get, docGet, update, append, delete, search, attrSearch
406 (not acceptable)	Certificate not recoanized	putCert
409 (conflict)	Document, component, or administration data inaccessible	info, get, docGet, append, update, delete, search, attrSearch
500 (Internal Server Error)	Internal error on Content Server	All functions

If an error occurs, the content server must deliver an ASCII string describing the error. The error must be entered in the header field `X-ErrorDescription`.

SAP Cache Server

Use

You can use the SAP Cache Server (see also [SAP Cache Server \[external\]](#)) in conjunction with third-party content servers to cache documents while they are being read accessed.

Features

From release 4.6B, you can use the SAP Content Server (see [SAP Content Server \[external\]](#)) in conjunction with an SAP Cache Server (see also note 216419).

To allow you to use signed URLs between the SAP Content Server and the SAP Cache Server, the Content Server component of the SAP Content Server HTTP Interface has been extended. To differentiate the new extended version from the previous version (version 4.5 or **0045**), the new version is called version 4.6 or **0046**.

If you want to use a third-party content server in a heterogeneous environment consisting of SAP Content Server, SAP Cache Server, and a third-party content server, the third-party content server must be enhanced so that it supports all SAP Content Server HTTP Interface commands. The new command `getCert` must also be implemented so that the requirements for the new version of the interface are fulfilled.

For further information, see the documentation on the Content Management Service in the SAP Library in the sections [KPro and Caching \[external\]](#), [Cascaded Caches \[external\]](#), and [Content Server Aliases \[external\]](#). You can also get information from the SAP Integration and Certification Centers.

Appendix

Parameters and Keywords

A parameter appears no more than once in a URL. The data type of each parameter is given in square brackets. "String" refers to characters from the ASCII character set, and "integer string" refers to characters from the set {0,1,2,3,4,5,6,7,8,9,0}. Parameters and keywords are listed here in alphabetical order.

accessMode [string]

Access mode

authId [string]

Client ID

caseSensitive [y|n]

Determines whether the search is case-sensitive. The default value is **n**.

caseSensitive=n

Search is not case-sensitive

caseSensitive=y

Search is case-sensitive

charset [string]

Describes the character set used to encode the component content (for example, ISO-8859-1; see also RFC 2046). Other values can be defined. Note that an **x-** must be placed before these values.

The character set is transferred as a **Content-Type** parameter.

compDateC [string]

Creation date of a component in UTC format, that is, YYYY-MM-DD

compDateM [string]

Date component was last changed, in UTC format.

compId [string]

Identifies a component within a document.



Additional information for partners who already support the SAP ArchiveLink interface: Data files for stored print lists and outgoing documents are interpreted as **compId** "data". The corresponding description files as interpreted as **compId** "descr". Notes have **compId** "note".

Predefined values for components:

- "data" data file
- "descr" attribute file
- "note" note

compTimeC [string]

Time the component was created, in UTC format: HH:MM:SS

compTimeM [string]

Time the component was last changed, in UTC format

compStatus [online|offline]

Status of component in the content repository. Meaning:

online

Component known and accessible

offline

Component known and currently inaccessible

Content-Disposition

Content-Disposition can be transferred with **compId** as an additional parameter if documents are being transferred as **multipart/form-data** (see also [HTTP-POST multipart/form-data \[page 23\]](#)). If **Content-Disposition** is used in this way, it must be consistent with **X-compId**



This parameter can be ignored.

Content-Length [integerstring]

Size of body or component in bytes. The parameter **Content-Length** can occur both in the response header and in part headers.

Content-Type [string]

Identifies the **Content-Type** of a component or a transferred document. Can occur in the response header and in part headers. With **charset**, the character set used to write the component content can be specified as the **Content-Type** parameter.

contRep [string]

Specifies the content repository

contRepDate [string]

Content repository date in UTC format

contRepDescription [string]

Text describing content repository content

contRepErrorDescription [string]

Text describing a content repository error

contRepStatus [running|stopped|error]

Content repository status. Meaning:

running

Content repository is running

stopped

Content repository has been stopped

error

Error in content repository

contRepTime [string]

Content repository time in UTC format

contRepVendorId [string]

Manufacturer and version of content repository software

dateC [string]

Creation date of a document, in UTC format

dateM [string]

Date the document was last changed, in UTC format

docID [string]

Unique identifier for document header

docProt [string]

Degree of protection of a document. **docProt** controls the degree of protection of a document and its information. **docProt** is a combination of the access rights **r**, **c**, **u**, and **d**. The parameter can overwrite the default set on the content server.

For example, the combination **docProt=rcud** fully protects a document.

docStatus [online|offline]

Status of document in the content repository. Meaning:

online

Document known and accessible

offline

Document known and inaccessible

expiration[*string*]

Expiry time of a signed URL, in UTC format

fromOffset [*integerstring*]

Specifies the starting point for a search or the beginning of a byte range within the component. The default is 0. This parameter is needed to be able to read parts of the component (with print lists, for example).

numComps [*integerstring*]

Number of components in a document

numResults [*integerstring*]

Determines the maximum number of results (hits) a search can return

pattern [*string*]

Character pattern used for an unrestricted search. See the function [search \[page 30\]](#).

The character patterns must be replaced by corresponding **escape** code in the form **escape = % HEX HEX**.

The disallowed characters must be replaced by corresponding **escape** coding in the form **escape = "% " HEX HEX**.

The counterpart of **pattern [*string*]** in the attribute search is

pattern [*integerstring+integerstring+string*] .

pattern [*integerstring+integerstring+string*]

Attribute pattern searched for in the attribute search. See function [attrSearch \[page 32\]](#).

The counterpart of **pattern [*integerstring+integerstring+string*]** in the unrestricted search is **pattern [*string*]** .

pVersion [*string*]

Specifies the interface version. Versions 0021, 0030 and 0031 were defined for the SAP ArchiveLink interface. The HTTP Content Server interface begins with version 0045.

resultAs [*string*]

Chooses the presentation form for the result of the **info** function. The default is **ASCII** .

resultAs=ascii

Results given in ASCII format

resultAs=html

Results given in HTML format

retCode [*integerstring*]

Part of the ASCII string sent by the content server to the client after an **mCreate** call. Contains the HTTP status code for the corresponding document.

secKey [*string*]

Specifies an access key that can be used to check access authorization. The SAP System generates the access key.

serverDate [*string*]

Content server date in UTC format

serverErrorDescription[*string*]

Text describing confirmed content server error

serverStatus[*running|stopped|error*]

Content server status. Meaning:

running

Content server is running

stopped

Content server has been stopped

error

Content server error

serverTime[*string*]

Content server time in UTC format

serverVendorId[*string*]

Manufacturer and version of content server software

timeC[*string*]

Creation date of a document, in UTC format

timeM[*string*]

Time the document was last changed, in UTC format

toOffset [*integerstring*]

Specifies the end of a byte range within the component. The default -1 means that the search should continue to the end of the component. **toOffset** has priority over any content range that may be set.

version[*string*]

Describes the application version used to create a document or component. The version is transferred as the **Content-Type** parameter. For some MIME types, version numbers are registered at IANA, so that they are always used consistently. For example, versions 2w, 4, 5 and 6 are used for application/msword.

serverStatusDescription[*string*]

Header field in which the content server enters an explanatory text if an error occurs.

scanPerformed[*bool*]

Specifies whether the content has to be scanned by SAP Content Server, or was it already scanned by the ABAP layer before storing. By default this parameter would not be present in the URL, as scanning needs to be enabled explicitly. (For more details see HTTP_PUT page 23)



Migrating Existing Archives

This section is only relevant for partners who supported the SAP ArchiveLink interface in the past and who now want to support the new HTTP Content Server Interface.

When a component is stored, the [MIME \[external\]](#) type is transferred by means of the field **Content-Type** in the request header (see RFCs 2045 and 2046). The content server then contains the MIME type, but does not evaluate it. The MIME type is used as the response **Content-Type** (for the **get** function, for example).

In older archives, only document classes such as ALF, FAX, and DOC were recognized. Therefore, a MIME type must be derived from these old document classes, so that previously stored documents can be accessed via the new interface.

Document Structure

There are many documents in the old SAP ArchiveLink interface that consist of several components. This is particularly true of documents in the document classes FAX, OTF and ALF. As well as one or more (FAX) data files, these documents sometimes also contain a description file (ALF) and a note file.

When existing archives are being migrated, a component ID (`compId`) must be assigned to each component. The procedure is as follows.

The `compId` data is assigned to the data file. If several data files exist (for document class FAX, for example, if there are several pages and each page is saved separately as a TIFF file), they are assigned to the components "data1", "data2", and so on. In this case there is no component "data".

A description file (ALF only) is assigned to the component "descr".

Lastly, the note file is assigned to the component "note".

Deriving MIME Types from Document Classes

When an archive is migrated, the MIME type of the component "data" (or "data1", "data2", and so on) is derived from the document class.

Derivation rules are shown in the following table. The order of the entries is important. The first appropriate entry from the top is used for derivation.

Document Class	MIME Type
FAX	image/tiff
BIN	application/octet-stream
DOC	application/msword
PDF	application/pdf
PS	application/postscript
RTF	application/rtf
XLS	application/vnd.ms-excel
MPP	application/vnd.ms-project
PPT	application/vnd.ms-powerpoint
ALF	application/x-alf
OTF	application/x-otf
RAW	application/x-raw
REO	application/octet-stream
SCR	application/x-scr
BMP	image/bmp
GIF	image/gif
JPG	image/jpeg

PCX	image/pcx
TIFF	image/tiff
TIF	image/tiff
HTM	text/html
TXT	text/plain



No MIME type is set for document classes not listed here.

The following MIME types are derived from the components “descr” (ALF only) and “note”:

compId	MIME Type
note	application/x-note
descr	application/x-alf-descr