software AG

Tamino API for C

Version 9.5 SP1

November 2013

tamino XML Server

This document applies to Tamino API for C Version 9.5 SP1.

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

Document ID: TAC-DOC-95SP1-20131030

Table of Contents

Tamino API for C

This documentation provides information about the Tamino API for C. This API offers a variety of methods and properties that enable any application that can call C functions to access and manipulate documents in a Tamino database.

This documentation is intended for software developers who wish to create C/C++-based applications that access XML databases stored in a Tamino XML Server. It is assumed that you are familiar with using the Tamino Manager for creating databases, and with using the Tamino Interactive Interface for loading schemas and data and performing XML database queries.

This documentation covers the following topics:

I Introduction

1 Introduction

The Tamino API for C allows client applications to access a Tamino XML Server without going through a web server. This distinguishes it from most other Tamino APIs. Since this functionality has first been introduced with Tamino 4.1, access to a Tamino XML Server with a version number of 4.1 or above only is supported by this API.

Since this API is entirely written in the C programming language, it is very well suited for client applications written in C or C++. However, any programming language that allows external C routines to be called can take advantage of the functionality provided by the Tamino API for C. The Tamino API for C is available for Windows as well as for UNIX platforms.

The Tamino API for C provides roughly the same functionality as the X-Machine programming interface; see *Requests using X-Machine Commands* for more details. However, in certain cases it provides a more convenient interface for communicating with a Tamino XML Server. For example, in a session context, the API does the bookkeeping for the Tamino session ID and session key automatically.

II Component Profile and Setup

2 Component Profile and Setup

This chapter provides information about the contents of this Tamino component and the installation.

Component Profile

Here you will find general information about the API and how to use it.

The Tamino API for C is automatically with the Tamino XML Server installation.

Tamino Component Profile for the Tamino API for C	
Supported Platforms	All platforms supported by Tamino XML Server.
Location of Installed Component	*<TaminoInstallDir>/SDK/TaminoAPI4C* (henceforth called *<TaminoAPI4CDir>*).

Structure of the Tamino API for C

The Tamino API for C consists of two layers.

The first layer provides roughly the same functionality as can be achieved by issuing X-Machine commands directly. For more details, see *Requests using X-Machine Commands*. This layer operates on byte buffers and is distributed as a shared library.

The second layer provides convenience functions. The convenience functions mostly deal with file I/O. They are described in detail in the section ***File Handling Reference***. This layer can be easily adapted to the programmer's specific needs since it is also distributed in source code.

Working with the Tamino API for C

You need a running Tamino database server to work with the Tamino API for C. If you want to access an existing Tamino database, you need to specify the database name only, e.g. "mydb", since the Tamino database is accessed via the Software AG product eXtended Transport Services (XTS).

In order to run an existing application that uses the Tamino API for C:

- Make sure that the module for webserverless access to Tamino has been installed on the computer on which the application runs. This includes a local installation of XTS. Please note that no license file is required.
- Make sure that your `PATH` variable (Windows) or `LD_LIBRARY_PATH variable` (UNIX) includes the directory containing the provided library file.

On platforms supporting 32-bit processing the PATH variable must include the directory *$INODIR/$INOVERS/sagcom32/*.

On platforms supporting 64-bit processing and if running the application in 64-bit mode, the PATH variable must include the directory *$INODIR/$INOVERS/sagcom64/*. If running the application in 32-bit mode on platforms supporting 64-bit processing, the PATH variable must include the directory *$INODIR/$INOVERS/sagcom32/*

- If you want to recompile your application, see ***Using the API***.

III Using the API

3 Using the API

An installed Tamino API for C delivers the following directory structure:

```
<TaminoInstallDir>
       |
       --- SDK
            |
            --- TaminoAPI4C
                      |
                      --- Documentation
                      --- include
                      --- lib
                      --- examples
```

- **Documentation**
  This directory contains the API's documentation in HTML and PDF.
- **include**
  This directory contains the header file *TaminoAPI4C.h* and *TaminoAPI4C_2.h*, which lists pre-defined constants and the function prototypes.
- **lib**
  This directory contains the platform-dependent shared libraries, e.g. *TaminoAPI4C.lib* and *TaminoAPI4C.dll* for Windows platforms, and *libTaminoAPI4C.so* for some UNIX platforms.
- **examples**
  This directory contains some examples on how to use the Tamino API for C.

This chapter provides detailed information on how to use the API in a C language context. If you want to call the functions provided by the Tamino API for C from another programming language, consult the language reference documentation for information about calling external C functions contained in shared libraries.

The following topics are covered below:

Writing and Compiling an Application

An application written in C that uses the Tamino API for C should follow the ANSI C specifications and include the provided header file *TaminoAPI4C.h* to obtain the function prototypes of the API.

The API is initialized by a call to `tac_init()`, which takes a database name as its argument. The corresponding server on which the database resides is found through eXtended Transport Services (XTS), a Software AG product delivered with Tamino XML Server. XTS either consults the local computer or a predefined server to obtain this information. If initialization is successful , `tac_init()` returns a non-negative handle. This handle is used as the first argument to all subsequent API calls, up to a final `tac_end()` call which finalizes the API.

Each handle can only be used by one thread. In a multi-threading environment, each thread must issue its own `tac_init()` command in order to obtain its own handle that must be kept track of by the programmer. Between the `tac_init()` call and the corresponding `tac_end()` call, different operations can be performed on the database, for example, inserting and deleting XML and non-XML documents. For more details, see the section ***Examples*** or *Requests using X-Machine Commands*. When working in a session context by issuing a `tac_connect()` command, the session ID and the session key are automatically kept track of in the Tamino API for C until the corresponding `tac_disconnect()` call is submitted.

Linking an Application

32-bit Platforms

When linking an application written in the C programming language with the Tamino API for C and which will run on a 32-bit platform, the shared library definition file *lib32/TaminoAPI4C.lib* (Windows) or the shared library *lib32/libTaminoAPI4C.so* (UNIX) must be known to the linker. These files can be found in the *lib/* directory of the API distribution. This shared library needs some libraries that are distributed with the Tamino XML Server CD, most notably the WSL (webserverless API) and XTS (eXtended Transport Services) libraries. Additionally the directory *$INODIR/$INOVERS/sagcom32/* must be known to the linker.

64-bit Platforms

Since applications running on 64-bit platforms can be run either in 32-bit mode or 64-bit mode one must decide before linking in which mode the application is to be executed. If the application is to be run in 32-bit mode then the instructions of the previous sections describing linking on 32-bit platforms applies. If the application it to be run in 64-bit mode then the shared library *lib64/libTaminoAPI4C.so* must be known to the linker. The file can be found in the *lib/* directory of the API distribution. This shared library needs some libraries that are distributed with the Tamino XML Server CD, most notably the WSL (webserverless API) and XTS (eXtended Transport Services) libraries. Additionally the directory *$INODIR/$INOVERS/sagcom64/* must be known to the linker.

Running an Application

An application that uses the Tamino API for C must know the path to the provided shared library. This can be achieved by setting the `PATH` variable (Windows) or the `LD_LIBRARY_PATH` variable (UNIX) to the directory that contains the library, e.g. *<TaminoInstallDir>/SDK/TaminoAPI4C/lib* (UNIX) following the directory structure given above. The Tamino API for C shared library needs some libraries that are distributed with the Tamino XML Server CD. See the previous section ***Linking an Application***. These directories must also be included in the library path.

IV Examples

4 Examples

The source code of the examples and the data needed to run them can be found in the following directory of the Tamino API for C documentation: <TaminoInstallDir>/SDK/TaminoAPI4C/Documentation/inoapi4c/examples/. All examples use a database with the default name "mydb"; the name can be changed by changing the argument to the `tac_init()` calls.

The following topics are covered below:

Determine the Tamino Version

As a first example of using the Tamino API for C, we shall illustrate how to ask a specific database for the corresponding Tamino version number. The source code reads:

```
01:  /*********************************************************
02:   *  Copyright (c) 2004 SOFTWARE AG, All Rights reserved
03:   *********************************************************
04:   *  - ask a Tamino database for its version number
05:   *********************************************************/
06:
07:  #include <stdio.h>
08:  #include "TaminoAPI4C.h"
09:
10:  #define checkERROR(a) if(a != TAC_SUCCESS) goto error
11:
12:  int main()
13:  {
14:     TAC_HANDLE handle;
15:     const char *response;
16:
17:     /* initialize the API */
18:     if ((handle = tac_init("mydb")) < 0) goto error;
19:
20:     /* ask for the Tamino version */
21:     checkERROR(tac_diagnose(handle, "version"));
22:
23:     /* display the Tamino response */
24:     checkERROR(tac_last_xml_response(handle, &response));
25:     printf("%s", response);
26:
27:     /* finalize the API */
28:     checkERROR(tac_end(handle));
29:
30:     return 0;
31:
32:     /* error handling */
33:  error:
34:     (void) tac_get_messagetext(handle, &response);
35:     printf("%s\n", response);
```

```
36:     (void) tac_end(handle);
37:
38:     return 1;
39:  }
```

In line 08, we include the provided header file *TaminoAPI4C.h*, which contains predefined constants and function prototypes. The macro in line 10 makes the error handling easier and the source code more readable. The API is initialized in line 18 through a call to `tac_init()`. If the returned handle is valid, it has a non-negative value that will be used as the first argument to subsequent API calls. The Tamino version is determined in line 21 by a call to `tac_diagnose()` and the result is displayed as an XML string through a call to `tac_last_xml_response()` in lines 24-25. The API is finalized in line 28 by calling `tac_end()`. Error handling is done in lines 33-36 by calling `tac_get_messagetext()` and printing the result to stdout.

When compiling the above example, link it with the shared library from the Tamino API for C, this is either *TaminoAPI4C.lib* (Windows) or *libTaminoAPI4C.so* (UNIX).

Load and Retrieve XML and Non-XML Documents

The next example loads XML and non-XML (binary) data into a Tamino database. The structure is given by a Tamino schema and all data is loaded from files using the file handling routines described in detail in the section ***File Handling Reference***. The source code reads:

```
01:  /*********************************************************
02:   *  Copyright (c) 2004 SOFTWARE AG, All Rights reserved
03:   *********************************************************
04:   *  - load and retrieve data into/from Tamino database
05:   *********************************************************/
06:  #include <stdio.h>
07:  #include "TaminoAPI4C.h"
08:  #include "TaminoAPI4C_2.h"
09:
10:  #define checkERROR(a)   if(a != TAC_SUCCESS) goto error
11:
12:  int main()
13:  {
14:     TAC_HANDLE handle;
15:     const char *response;
16:
17:     /* initialize the API */
18:     if ((handle = tac_init("mydb")) < 0) goto error;
19:
20:     /* load Tamino schema definition from file */
21:     checkERROR(tac_define_from_file(handle, "TAC_samples.tsd"));
22:
23:     /* load XML entries from file into Tamino collection */
```

```
24:     checkERROR(tac_process_from_file(handle, "RealEstate",
25:                                      "TAC_samples.xml"));
26:
27:     /* load non-XML data from file into Tamino collection */
28:     checkERROR(tac_load_from_file(handle, "RealEstate",
29:                "images/sample1.jpg", "TAC_sample1.jpg", "image/jpeg"));
30:     checkERROR(tac_load_from_file(handle, "RealEstate",
31:                "images/sample2.jpg", "TAC_sample2.jpg", "image/jpeg"));
32:
33:     /* check for entries in the Tamino collection */
34:     checkERROR(tac_xql(handle, "RealEstate", "/images"));
35:
36:     /* display the Tamino response */
37:     checkERROR(tac_last_xml_response(handle, &response));
38:     printf("%s\n", response);
39:
40:     /* undefine the above defined schema (clean-up) */
41:     checkERROR(tac_undefine(handle, "RealEstate"));
42:
43:     /* finalize the API */
44:     checkERROR(tac_end(handle));
45:
46:     return 0;
47:
48:  /* error handling */
49:  error:
50:     (void) tac_get_messagetext(handle, &response);
51:     printf("%s\n", response);
52:     tac_end(handle);
53:
54:     return 1;
55:  }
```

In line 08, we include the provided header file *TaminoAPI4C.h*, which contains predefined constants and function prototypes. The macro in line 10 makes the error handling easier and the source code more readable. The API is initialized in line 18 through a call to `tac_init()`. If the returned handle is valid, it has a non-negative value that will be used as the first argument to subsequent API calls. The schema of the XML data is loaded from a file in line 21 and the corresponding data is loaded in line 24. The schema also allows for non-XML data to be inserted: two images are loaded in lines 28-31. In order to check whether the documents were loaded into the database, we submit a query in line 34 which asks for the entries of the two binary images and displays the XML result document in lines 37-38. The XML entries can be retrieved by changing the last argument in `tac_xql()`, for example, into "/person", "/property" or simply "*". We do a clean-up in line 41 by deleting the collection "RealEstate" which also deletes all of the previously inserted documents. The API is finalized in line 44 by calling `tac_end()`. The error handling is done in lines 48-54 by calling `tac_get_messagetext()` and printing the result to stdout.

When compiling the above example, link it with the shared library from the Tamino API for C, this is either *TaminoAPI4C.lib* and *TaminoAPI4C_2.lib* on Windows platforms or *libTaminoAPI4C.so* and *libTaminoAPI4C_2.so* on UNIX platforms.

List All Collections in a Database

The following example lists all collections available in a given Tamino database. It uses the XML parser Expat (**http://www.libexpat.org/**) to process the response from Tamino. The source code reads:

```
01:  /*******************************************************
02:   *  Copyright (c) 2004 SOFTWARE AG, All Rights reserved
03:   *******************************************************
04:   * - list all collections in a Tamino database
05:   *******************************************************/
06:
07:  #include <stdio.h>
08:  #include <string.h>
09:  #include "expat.h" /* use Expat XML parser for Tamino responses */
10:  #include "TaminoAPI4C.h"
11:
12:  #define checkERROR(a)   if(a != TAC_SUCCESS) goto error
13:
14:  #ifdef _WIN32
15:  #  define strcasecmp(a, b) _stricmp(a, b) /* non-case sensitive co. */
16:  #endif
17:
18:  static void CollectionElement
19:              (void *userData, const char *name, const char **atts)
20:  {
21:    int I = 0;
22:    int *number = (int *) userData;
23:
24:    if (strcasecmp(name, "tsd:collection") == 0) {
25:      *number += 1;
26:      printf("- info: Collection #%d:", *number);
27:      while (atts[i] != NULL) {
28:        if (strcasecmp(atts[i], "name") == 0) {
29:          printf("\t%s\n", atts[i+1]);
30:        }
31:        I+=2;
32:      }
33:    }
34:  }
35:
36:  static int find_collections(TAC_HANDLE handle)
37:  {
38:    XML_Parser parser;
39:    int ret_code1, ret_code2, number = 0;
40:    const char *response;
41:
42:    /* query the database */
```

```
43:    ret_code1 = tac_xql(handle, "ino:collection", "tsd:collection");
44:    if (ret_code1 == TAC_SUCCESS) {
45:      /* process the response */
46:      parser = XML_ParserCreate(NULL);
47:      XML_SetUserData(parser, &number);
48:      XML_SetElementHandler(parser, CollectionElement, NULL);
49:      (void) tac_last_xml_response(handle, &response);
50:      if ((ret_code2 =
51:           XML_Parse(parser, response, strlen(response), 1)) == 0) {
52:        fprintf(stderr,
53:          "- error: while parsing Tamino response (%d)\n", ret_code2);
54:      }
55:      XML_ParserFree(parser);
56:    }
57:    return ret_code1;
58:  }
59:
60:  int main()
61:  {
62:     TAC_HANDLE handle;
63:     const char *response;
64:
65:     /* initialize the API */
66:     if ((handle = tac_init("mydb")) < 0) goto error;
67:
68:     /* ask for all collections in the Tamino database */
69:     checkERROR(find_collections(handle));
70:
71:     /* finalize the API */
72:     checkERROR(tac_end(handle));
73:
74:     return 0;
75:
76:  /* error handling */
77:  error:
78:     (void) tac_get_messagetext(handle, &response);
79:     printf("%s\n", response);
80:     tac_end(handle);
81:
82:     return 1;
83:  }
```

The necessary header file to use the XML parser Expat is included in line 09. In line 10 we include the provided header file *TaminoAPI4C.h*, which contains predefined constants and function prototypes. The macro in line 12 makes the error handling easier and the source code more readable. The macro in line 14 is necessary to allow for case-insensitive string comparisons under Windows. The main program starts in line 60 and the API is initialized in line 66 through a call to `tac_init()`. If the returned handle is valid, it has a non-negative value that it is used as the first argument to subsequent API calls. In line 69, we call our function `find_collections`, which was defined in lines 36-58. In this function, we query the Tamino database "mydb" for its collections by calling the function `tac_xql()` in line 43. Our function `CollectionElement`, which is defined in lines 18-

34, is registered to the XML parser in line 48. The actual parsing is initiated in line 51, where we provide the response from the previous `tac_xql()` call as argument. Whenever a new opening element tag is recognized by the XML parser in the source file, the registered function `CollectionElement()` is called, where we do a string comparison to find the correct entries that correspond to Tamino collections. All collection names are then printed to standard output. The API is finalized in line 72 by calling `tac_end()`. Error handling is done in lines 77-82 by calling `tac_get_messagetext()` and printing the result to stdout.

When compiling the above example, link it with the static or shared library from the Expat distribution (**http://www.libexpat.org/**) and also with the shared library from the Tamino API for C, which is either *TaminoAPI4C.lib* (Windows) or *libTaminoAPI4C.so* (UNIX).

V Tamino API for C Reference

This reference provides the following information:

5 Return and Error Codes

Each Tamino API for C function except `tac_init()` returns an integer value.

Upon success, this value is the predefined constant `TAC_SUCCESS`, which is defined as 0 in the header file *TaminoAPI4C.h*.

If a function call fails, an error code is returned. A negative value indicates that the API already detected an error without communicating with Tamino. An error code between 100 and 999 indicates a communication problem, see *HTTP Status Codes* for more details. An error code greater than 999 indicates a processing problem in the Tamino XML Server; more details can be obtained by calling the function `tac_get_messagetext()`.

6 Basic Commands

The following commands are available:

tac_admin

```
int tac_admin(TAC_HANDLE handle, const char *request)
```

Submits an administration command to the Tamino XML Server.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
request - a string containing the request. See *Requests using X-Machine Commands, The _admin command* for more details.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_define

```
int tac_define(TAC_HANDLE handle, const char *schema)
```

Defines or redefines a schema or collection in the Tamino XML Server.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
schema - a string buffer containing the schema definition.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_delete

```
int tac_delete(TAC_HANDLE handle, const char *collection, const char *query)
```

Deletes document(s) from the Tamino XML Server.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
collection - a string containing the name of the collection to be used.
query - a string containing an X-Query expression. See *Requests using X-Machine Commands, The _delete command* for more details.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_diagnose

```
int tac_diagnose(TAC_HANDLE handle, const char *request)
```

Submits a diagnose command to the Tamino XML Server.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
request - a string containing the request, e.g. "version". See *Requests using X-Machine Commands, The _diagnose command* for more details.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_end

```
int tac_end(TAC_HANDLE handle)
```

Closes a specific connection to a Tamino XML Server. It has no X-Machine equivalent. In a multi-threading environment, each thread must issue its own `tac_end()` command.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or a negative value upon failure.

tac_get_encoding

```
int tac_get_encoding(TAC_HANDLE handle, const char **encoding)
```

Gets the encoding of the current Tamino response, if it was in XML format.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
encoding - a pointer to a string that will contain the encoding of the Tamino response, if it was in XML format; otherwise a pointer to an empty string is returned.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or a negative value upon failure.

tac_get_inoID

```
int tac_get_inoID(TAC_HANDLE handle, long int *ino_id)
```

Gives the ino:id in the argument ino_id under which the last document was stored in the Tamino XML Server.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
ino_id - will contain the ino:id of the last Tamino request if available, otherwise a value of -1; mostly used after a `tac_load()` request.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or a negative value upon failure.

tac_get_messagetext

```
int tac_get_messagetext(TAC_HANDLE handle, const char **message)
```

Returns an XML string that contains the message text from the last request to the Tamino XML Server which contains helpful information if a Tamino error occurred. See ***Return and Error Codes*** for more details.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
message - a pointer to the message text as an XML string.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or a negative value upon failure.

tac_init

```
TAC_HANDLE tac_init(const char *dbname)
```

Prepares a new connection to a Tamino XML Server. It has no X-Machine equivalent. In a multi-threading environment, each thread must issue its own `tac_init()` command.

Takes Parameters:

dbname - a string containing the name of the Tamino database used for this handle, where the name was registered through Tamino from the start-up of the database via XTS.

Returns:

A valid handle upon success, or the predefined constant `TAC_INVALID_HANDLE` upon failure to allocate a new handle structure.

tac_last_xml_response

```
int tac_last_xml_response(TAC_HANDLE handle, const char **response)
```

Returns the XML response from the last request to the Tamino XML Server.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
response - a pointer to a string that will contain the Tamino response as an XML string if available; otherwise a pointer to an empty string.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or a negative value upon failure.

tac_load

```
int tac_load(TAC_HANDLE handle, const char *collection, const char *docname, const char *document, long int size, const char *mime_type)
```

Inserts a single document into the Tamino XML Server.

The single command transaction parameters are not passed with this command. If required, use `tac_process()` instead for XML documents.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
collection - a string containing the name of the collection to be used.
docname - a string containing the doctype, a slash ('/'), and the name of the document, e.g. "images/property1.jpg".
document - a pointer to the byte buffer that contains the document.
size - the size of the buffer in bytes.
mime_type - a string containing the MIME-type of the document, e.g. "image/jpeg".

Returns:

The predefined constant `TAC_SUCCESS` upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_process

```
int tac_process(TAC_HANDLE handle, const char *collection, const char *xml)
```

Inserts one or more XML documents into the Tamino XML Server.

For multiple documents one must use the file format as specified in the section *Tamino Data Loaders* of the Tamino XML Server documentation.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
collection - a string containing the name of the collection to be used.
xml - a string buffer containing the XML document(s).

Returns:

The predefined constant `TAC_SUCCESS` upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_retrieve

```
int tac_retrieve(TAC_HANDLE handle, const char *collection, const char *docname,
const char **document, long int *size, const char **mime_type)
```

Gets a single document from the Tamino XML Server.

The single command transaction parameters are not passed with this command. If required, use `tac_xql()` or `tac_xquery()` instead for XML documents.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
collection - a string containing the name of the collection to be used.
docname - a string containing the doctype, a slash ('/'), and the name of the document, e.g. "property/property1.jpg".
document - a pointer to the byte buffer that contains the document.
size - the size of the buffer in bytes.
mime_type - a pointer to a string that will contain the MIME-type of the returned document, e.g. "image/jpeg".

Returns:

The predefined constant `TAC_SUCCESS` upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_set_encoding

```
int tac_set_encoding(TAC_HANDLE handle, const char *encoding)
```

Sets the desired encoding for the subsequent Tamino responses. If not used, the default encoding of the Tamino API for C uses "utf-8".

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
encoding - a string specifying the desired encoding, e.g. "iso-8859-1".

Returns:

The predefined constant `TAC_SUCCESS` upon success, or a negative value upon failure.

tac_set_user_info

```
int tac_set_user_info(TAC_HANDLE handle, const char *username, const char *password)
```

Sets the credentials for the Tamino communication. It has no X-Machine equivalent. Please note that Tamino XML Server 4.1 does not use SSL for communications.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
username - a string containing the username, which can include the domain.
password - a string containing the password.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or a negative value upon failure.

tac_undefine

```
int tac_undefine(TAC_HANDLE handle, const char *name)
```

Undefines a collection, schema or doctype in the Tamino XML Server.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.

name - a string containing the name to be deleted. For more details on the syntax, see *Requests using X-Machine Commands, The _undefine command* for more details.

Returns:

The predefined constant TAC_SUCCESS upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_xql

```
int tac_xql(TAC_HANDLE handle, const char *collection, const char *query)
```

Submits a query in X-Query syntax to the Tamino XML Server.

Takes Parameters:

handle - a valid handle obtained from tac_init().
collection - a string containing the name of the collection to be used for the query.
query - the query string, e.g. "/property"; for more details, see the *X-Query Reference Guide* in the Tamino XML Server documentation.

Returns:

The predefined constant TAC_SUCCESS upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_xquery

```
int tac_xquery(TAC_HANDLE handle, const char *collection, const char *query)
```

Submits a query in XQuery 4 syntax to the Tamino XML Server.

Takes Parameters:

handle - a valid handle obtained from tac_init().
collection - a string containing the name of the collection to be used for the query.
query - the query string, e.g. "input()/property"; for more details, see the *XQuery 4 Reference Guide* in the Tamino XML Server documentation.

Returns:

The predefined constant TAC_SUCCESS upon success, or an error code upon failure (see ***Return and Error Codes***).

7 Transaction-Related Commands

The transaction processing mechanisms using the Tamino API for C as described in this chapter allow a transaction to relate to a single database only. They provide the same functionality as the X-Machine commands, see *Requests using X-Machine Commands* for more details.

This chapter describes the following functions and commands:

Transaction Parameter Handling

We distinguish two modes:

- A session context identified by the predefined constant `TAC_SESSION`. This mode is initiated by `tac_connect()` and terminated by `tac_disconnect()`.
- A single command context. This could be a command issued in a non-session context (sometimes called autocommit mode), but it can also refer to a single command issued in a session. This single command context is identified by the predefined constant `TAC_COMMAND`.

Each context has its own set of transaction parameters. These can be set via the functions:

- `tac_set_isolation_level()`
- `tac_set_lock_mode()`
- `tac_set_lockwait()`
- `tac_set_nonactivity_timeout()`
- `tac_set_transaction_timeout()`

Each of these functions takes as its first parameter a valid handle, and as its second parameter either the keyword `TAC_SESSION` or the keyword `TAC_COMMAND`. The appropriate Tamino default parameter will be used if `TAC_DEFAULT` is specified as the third argument of any of these functions.

If mode `TAC_SESSION` is used, the newly-set values will be used in the next `tac_connect()` call, and also in the first request after a `tac_commit()` or `tac_rollback()` call i.e. at the beginning of the next transaction.

If the keyword `TAC_COMMAND` is used to set a value to a non-default value, these transaction parameters will be submitted along with each subsequent request. A call to `tac_connect()` is an exception: the values set via the `TAC_SESSION` keyword will be transmitted.

The current value of the transaction parameters can be obtained by calling `tac_get_transaction_parameters()` with the following arguments:

1. a valid handle;
2. either the keyword `TAC_COMMAND` or the keyword `TAC_SESSION`;
3. a pointer to a structure that contains the five parameter fields.

tac_commit

```
int tac_commit(TAC_HANDLE handle)
```

Commits a transaction within a Tamino session and starts a new transaction.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_connect

```
int tac_connect(TAC_HANDLE handle)
```

Starts a Tamino session and switches from autocommit mode to local mode. The Tamino session ID and session key are handled automatically as well.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_disconnect

```
int tac_disconnect(TAC_HANDLE handle)
```

Closes a Tamino session and switches back from local mode to autocommit mode.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_get_transaction_parameters

```
int tac_get_transaction_parameters(TAC_HANDLE handle, int mode, TAC_TRANSACTION_T
*trans)
```

Gets the current values for the transaction parameters used in Tamino requests.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
mode - either `TAC_SESSION` or `TAC_COMMAND` For more information, see **Transaction Parameter Handling**.
trans - a pointer to the predefined structure `TAC_TRANSACTION_T` containing the five current transaction parameters for the specified mode.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or a negative value upon failure.

tac_rollback

```
int tac_rollback(TAC_HANDLE handle)
```

Discards all changes made during the current transaction within a Tamino session.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_set_isolation_level

```
int tac_set_isolation_level(TAC_HANDLE handle, int mode, int isolation_level)
```

Sets the `isolationLevel` parameter for Tamino requests.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
mode - either `TAC_SESSION` or `TAC_COMMAND` For more information, see **Transaction Parameter Handling**.
isolation_level - one of the predefined constants `TAC_DEFAULT`, `TAC_ISOLATIONLEVEL_NONE`, `TAC_ISOLATIONLEVEL_UNCOMMITTED`, `TAC_ISOLATIONLEVEL_COMMITTED`, `TAC_ISOLATIONLEVEL_CURSOR`, `TAC_ISOLATIONLEVEL_DOCUMENT` or `TAC_ISOLATIONLEVEL_SERIALIZABLE`.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or a negative value upon failure.

tac_set_lock_mode

```
int tac_set_lock_mode(TAC_HANDLE handle, int mode, int lock_mode)
```

Sets the `lockMode` parameter for Tamino requests.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
mode - either `TAC_SESSION` or `TAC_COMMAND`. For more information, see **Transaction Parameter Handling**.
lock_mode - one of the predefined constants `TAC_DEFAULT`, `TAC_LOCKMODE_SHARED`, `TAC_LOCKMODE_PROTECTED` or `TAC_LOCKMODE_UNPROTECTED`.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or a negative value upon failure.

tac_set_lockwait

```
int tac_set_lockwait(TAC_HANDLE handle, int mode, int lockwait)
```

Sets the `lockwait` parameter for Tamino requests.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
mode - either `TAC_SESSION` or `TAC_COMMAND`. For more information, see **Transaction Parameter Handling**.
lockwait - one of the predefined constants `TAC_DEFAULT`, `TAC_LOCKWAIT_YES` or `TAC_LOCKWAIT_NO`.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or a negative value upon failure.

tac_set_nonactivity_timeout

```
int tac_set_nonactivity_timeout(TAC_HANDLE handle, int mode, int timeout)
```

Sets the `nonactivityTimeout` parameter for Tamino requests.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
mode - either `TAC_SESSION` or `TAC_COMMAND`. For more information, see **Transaction Parameter Handling**.
timeout - the desired non-activity timeout in seconds. The predefined constant `TAC_DEFAULT`, which is set to zero, can also be used.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or a negative value upon failure.

tac_set_transaction_timeout

```
int tac_set_transaction_timeout(TAC_HANDLE handle, int mode, int timeout)
```

Sets the `transactionTimeout` parameter for Tamino requests.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
mode - either `TAC_SESSION` or `TAC_COMMAND`. For more information, see **Transaction Parameter Handling**.
timeout - the desired maximum transaction duration in seconds. The predefined constant `TAC_DEFAULT`, which is set to zero, can also be used.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or a negative value upon failure.

8 Cursor-Related Commands

The following commands are available:

tac_cursor_close

```
int tac_cursor_close(TAC_HANDLE handle, int cursor)
```

Closes a Tamino cursor. For more details, see *Requests using X-Machine Commands*, *The _cursor command*.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
cursor - the cursor identification obtained from a call to `tac_cursor_open_xql()` or `tac_cursor_open_xquery()`.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_cursor_fetch

```
int tac_cursor_fetch(TAC_HANDLE handle, int cursor, int position, int quantity)
```

Fetches an existing Tamino cursor to browse through a result set. For more details, see *Requests using X-Machine Commands*, *The _cursor command*.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
cursor - the cursor identification obtained from a call to `tac_cursor_open_xql()` or `tac_cursor_open_xquery()`.
position - the starting point within the result set.
quantity - the number of entries to be returned; if set to 0 or a negative value, the default of 10 is used.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_cursor_new_xql

```
int tac_cursor_new_xql(TAC_HANDLE handle, int *cursor, const char *collection, const
char *query, int scroll, int sensitive, int count, int position, int quantity)
```

Opens and fetches a Tamino cursor for the specified X-Query string. For more details, see *Requests using X-Machine Commands, The _cursor command*.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
cursor - a pointer to an integer variable which will receive the cursor identification.
collection - a string containing the name of the collection to be used.
query - the X-Query string.
scroll - one of the predefined constants `TAC_YES` or `TAC_NO`.
sensitive - only the predefined constant `TAC_VAGUE` is permitted here.
count - one of the predefined constants `TAC_NO` or `TAC_CHEAP`.
position - the starting point within the result set.
quantity - the number of entries to be returned; if set to 0 or a negative value, the default of 10 is used.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_cursor_new_xquery

```
int tac_cursor_new_xquery(TAC_HANDLE handle, int *cursor, const char *collection,
const char *query, int scroll, int sensitive, int count, int position, int quantity)
```

Opens and fetches a Tamino cursor for the specified XQuery 4 string. For more details, see *Requests using X-Machine Commands, The _cursor command*.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
cursor - a pointer to an integer variable which will receive the cursor identification.
collection - a string containing the name of the collection to be used.
query - the XQuery string.
scroll - one of the predefined constants `TAC_YES` or `TAC_NO`.
sensitive - one of the predefined constants `TAC_VAGUE` or `TAC_NO`.
count - one of the predefined constants `TAC_NO` or `TAC_CHEAP`.

position - the starting point within the result set.
quantity - the number of entries to be returned; if set to 0 or a negative value, the default of 10 is used.

Returns:

The predefined constant TAC_SUCCESS upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_cursor_open_xql

```
int tac_cursor_open_xql(TAC_HANDLE handle, int *cursor, const char *collection,
const char *query, int scroll, int sensitive, int count)
```

Opens a Tamino cursor for the specified X-Query string. For more details, see *Requests using X-Machine Commands, The _cursor command*.

Takes Parameters:

handle - a valid handle obtained from tac_init().
cursor - a pointer to an integer variable which will receive the cursor identification.
collection - a string containing the name of the collection to be used.
query - the X-Query string.
scroll - one of the predefined constants TAC_YES or TAC_NO.
sensitive - only the predefined constant TAC_VAGUE is permitted here.
count - one of the predefined constants TAC_NO or TAC_CHEAP.

Returns:

The predefined constant TAC_SUCCESS upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_cursor_open_xquery

```
int tac_cursor_open_xquery(TAC_HANDLE handle, int *cursor, const char *collection,
const char *query, int scroll, int sensitive, int count)
```

Opens a Tamino cursor for the specified XQuery 4 string. For more details, see *Requests using X-Machine Commands, The _cursor command*.

Takes Parameters:

handle - a valid handle obtained from tac_init().
cursor - a pointer to an integer variable which will receive the cursor identification.

collection - a string containing the name of the collection to be used.
query - the XQuery string.
scroll - one of the predefined constants TAC_YES or TAC_NO.
sensitive - one of the predefined constants TAC_VAGUE or TAC_NO.
count - one of the predefined constants TAC_NO or TAC_CHEAP.

Returns:

The predefined constant TAC_SUCCESS upon success, or an error code upon failure (see ***Return and Error Codes***).

VI File Handling Reference

9 File Handling Reference

As an example of how to extend the functionality of the Tamino API for C, we provide several convenience functions for file I/O. They are all part of the second layer and are also available in source code so that they can be adapted to individual needs.

The following functions are provided:

tac_define_from_file

```
int tac_define_from_file(TAC_HANDLE handle, const char *filename)
```

Defines a schema to the Tamino XML Server by reading it from a file.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
filename - a string containing the full name of the file that contains the schema.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_load_from_file

```
int tac_load_from_file(TAC_HANDLE handle, const char *collection, const char *docname,
const char *filename, const char *mime_type)
```

Inserts a single document from a file into the Tamino XML Server; mostly used for non-XML documents but also useable for XML data.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
collection - a string containing the name of the collection to be used.
docname - a string containing the doctype, a slash ('/'), and the name of the document, e.g. "images/property1.jpg".
filename - a string containing the full name of the file that contains the document.
mime_type - a string containing the MIME-type of the document, e.g. "image/jpeg".

Returns:

The predefined constant `TAC_SUCCESS` upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_process_from_file

```
int tac_process_from_file(TAC_HANDLE handle, const char *collection, const char
*filename)
```

Loads XML document(s) from a file into the Tamino XML Server.

For multiple documents in one single file, one must use the file format as specified in the *Tamino Data Loaders* section of the Tamino XML Server documentation

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
collection - a string containing the name of the collection to be used.
filename - a string containing the full name of the file that contains the document(s).

Returns:

The predefined constant `TAC_SUCCESS` upon success, or an error code upon failure (see ***Return and Error Codes***).

tac_retrieve_to_file

```
int tac_retrieve_to_file(TAC_HANDLE handle, const char *collection, const char
*docname, const char **mime_type, const char *filename)
```

Gets a single document from the Tamino XML Server and writes it to a file.

Takes Parameters:

handle - a valid handle obtained from `tac_init()`.
collection - a string containing the name of the collection to be used.
docname - a string containing the doctype, a slash ('/'), and the name of the document, e.g. "images/property1.jpg".
mime_type - a pointer to a string that will contain the MIME-type of the returned document, e.g. "image/jpeg".
filename - a string containing the full name of the file that will contain the document.

Returns:

The predefined constant `TAC_SUCCESS` upon success, or an error code upon failure (see ***Return and Error Codes***).

Index