

SNiFF+™

Version 3.2 for Unix and Windows

Symbol Table API



TakeFive Software, Inc.
Cupertino, CA
E-mail: info@takefive.com

TakeFive Software GmbH
5020 Salzburg, Austria
E-mail: info@takefive.co.at

Copyright

Copyright © 1992–1999 TakeFive Software Inc.

All rights reserved. TakeFive products contain trade secrets and confidential and proprietary information of TakeFive Software Inc. Use of this copyright notice is precautionary and does not imply publication or disclosure.

Parts of SNIFF+:

Copyright 1991, 1992, 1993, 1994 by Stichting Mathematisch Centrum,
Amsterdam, The Netherlands.

Portions copyright 1991-1997 Compuware Corporation.

Trademarks

SNIFF+ is a trademark of TakeFive Software Inc.

Other brand or product names are trademarks or registered trademarks of their respective holders.

Credits

The first version of Sniff was developed at the Informatics Laboratory of the Union Bank of Switzerland. Its development was considerably facilitated by the public domain application framework ET++.

Authors of the first version:

Walter Bischofberger (Sniff)

Erich Gamma (Sniffgdb)

Erich Gamma and André Weinand (ET++)

Table of Contents

Symbol Table API	5
Introduction	5
How the SNIFF+ Symbol Table API queries the Symbol Table	5
List of queryable symbols in the Symbol Table	6
Using the SNIFF+ Symbol Table API	7
Limitations of the current release of the SNIFF+ Symbol Table API	7
Files that are part of the Symbol Table API	7
Linking the Sample Program	8
Starting the Sample Program	8
Command Reference	9
Startup	9
Connecting to SNIFF+	9
Closing a project	10
Closing sniffapi server	10
Closing the connection	11
Opening a project in apiserver.	11
Performing a query	12
Getting a hashed item	13
Interpreting the query result	14
Description of Queries	20
Hierarchy of a class	20
Query Full Hierarchy	20
Get rootclass	20
Get Superclasses	21
Get Subclasses	21
Classes referred by item	21
Classes referring to item	21
Query for members	22
Find all symbols	22
Find File	22
Items, eProc and eMethodI references to	23
Items referring to an item.	23
Get version information	23

Symbol Table API

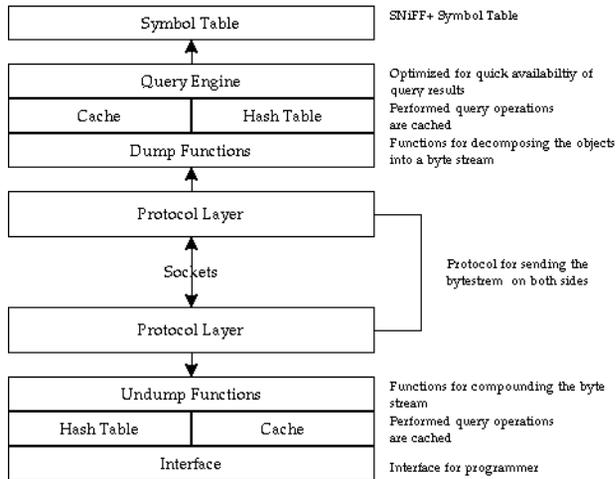
Introduction

The SNIFF+ Symbol Table API allows you to access symbol information from the Symbol Table of a project and to use this information for writing your own applications. For a definition of the Symbol Table, please refer to **User's Guide > Glossary**. The functions for writing your own applications are contained in the SNIFF+ Symbol Interface Library. These functions have C linkage and are based on the idea that requests to the Symbol Table are performed as queries.

How the SNIFF+ Symbol Table API queries the Symbol Table

The diagram on the following page indicates how queries to the Symbol Table are made via the SNIFF+ Symbol Table API. A query goes through a number of stages (or layers), starting from the programmer interface (Symbol Table API) and ending up at the Symbol Table. Some of the important features of the query process are:

- Query operations are cached both at the interface level and at the Symbol Table level. This results in reduced data traffic and faster query processing times.
- Data exchange between the Symbol Table API and the Symbol Table is managed by two protocol layers.
- The Query Engine to the Symbol Table is optimized. This results in a quick availability of query results.



List of queryable symbols in the Symbol Table

You can query the Symbol Table for the following symbols:

- *Class* scope:
 - instance variable
 - method definition
 - method implementation
 - template declaration strings
 - superclasses
 - subclasses
 - references
- *File* scope:
 - functions
 - variables
 - types
 - enumerations
 - macros
 - includes

Other queryable items

- Version configuration symbols and references

2

Using the SNIFF+ Symbol Table API

Please refer to the *Sample Interface Definition File* (`c_symTab_API.h`) and *Sample Source File* (`test.c`) in `<your_sniff_installation_directory>/symbol_API` directory. The file `test.c` gives detailed information about how to retrieve information from the Symbol Table of a project. The file `c_symTab_API.h` contains declarations of functions and data types that are contained in the SNIFF+ Symbol Interface Library.

Limitations of the current release of the SNIFF+ Symbol Table API

The current release of the SNIFF+ Symbol Table API has some limitations associated with it. These are:

- SNIFF+ must run on local machine as the ApiServer.
- When SNIFF+ runs in API mode, it is blocked for user interaction during the query.
- SNIFF+ must be running before a query can be run.

Files that are part of the Symbol Table API

The following files specific to the SNIFF+ Symbol Table API are created in your `$SNIFF_DIR/symbol_API` and `$SNIFF_DIR/bin` directories:

In `$SNIFF_DIR/symbol_API`:

- | | |
|--|---|
| ■ <code>README</code> | README file |
| ■ <code>c_symTab_API.h</code> | Interface definition file |
| ■ <code>test.c</code> | Sample source |
| ■ <code>lib.<platform>/lib-SniffApi.a</code> | SNIFF+ Symbol Interface Library platform-specific library |

Linking the Sample Program

You will have to link the sample program to the SNIFF+ Symbol Interface Library.

Examples:

on SunOs:

```
gcc -o test test.c libSniffApi.a -lm
```

on Solaris:

```
gcc -o test test.c libSniffApi.a -lm -lsocket -lnsl
```

on Windows:

```
cl -o test.exe test.c libSniffApi.a wsock32.lib advapi32.lib
```

Starting the Sample Program

To start the Sample Program, do the following:

1. Start SNIFF+.
2. In the Launch Pad, select **Tools > Working Environments**.
The Working Environments tool appears.
3. In the Working Environments tool, select the Working Environment that you will be working in.
4. Select **Tools > Log**.
5. In the Log tool that appears, make a note of the session number.

In a Command Shell

Navigate to the `symbol_API` directory.

On Windows, type

```
■ test.exe -s<session_number> -p<project>
```

On Unix, type

```
■ ./test -s<session_number> -p<project>
```

`<session_number>` is the session number that you got from the Log tool, e.g., session2.

`<project>` is the absolute path to the project.

Note

DO NOT LEAVE ANY SPACES when typing `-s` followed by the session number as well as `-p` followed by the absolute path to the project.

The test program then opens the project and dumps the current version of all files, dumps the class hierarchy and prints the information about all classes to the Shell. For details, please refer to the `test.c` file in your `SNIFF_DIR/symbol_API` directory.

3

Command Reference

Please refer to the *Sample Interface Definition File* in

`<your_sniff_installation_directory>/symbol_API/`

directory for a description of the declarations of functions and data types that are contained in the SNIFF+ Symbol Interface Library.

Startup

Function

```
void __si__module__init(void)
```

Description

must be called once at startup before any interface action is performed

Return Value

```
void
```

Parameter

```
void
```

Connecting to SNIFF+

Function

```
SNIFFACCESS si_open(char* session, char*host)
```

Description

opens a connection to a running SNIFF+

Return Value

```
SNIFFACCESS :must be given as a parameter for each access function
```

Parameter

`char * host` :string that gives the name of the machine where sniffapi is running. When `host` is set to `NULL`, the local host is used. This parameter is not yet implemented to run on a remote host

`char * session` :the SNIFF+ session represented in the SNIFF+ Log window. When `session` is set to `NULL`, then `session0` is assumed

Note

SNIFF+ must be up and running for the API connection to work.

Closing a project

Function

```
si_bool si_close_project (SNIFFACCESS, char* proj)
```

Description

closes a project

Return Value

`si_bool` :tells if function code was successfully sent

Parameters

`SNIFFACCESS` :handle for the desired connection

`char* proj` :name of project to be closed

Closing sniffapi server

Function

```
si_bool si_quit (SNIFFACCESS)
```

Description

closes sniffapi server

Return Value

`si_bool` :tells if function code was successfully sent

Parameter

`SNIFFACCESS` :handle for the desired connection

Closing the connection

Function

```
si_bool si_exit (SNIFFACCESS)
```

Description

closes connection

Return Value

`si_bool` :tells if function code was successfully sent

Parameter

`SNIFFACCESS` :handle for the desired connection

Opening a project in apiserver

Function

```
si_bool si_open_project SNIFFACCESS,char* proj
```

Description

opens project in apiserver

Return Value

`si_bool` :tells if function code was successfully sent

Parameter

`SNIFFACCESS` :handle for the desired connection

`char* proj` :same name of project as in `si_open`

Performing a query

Function

```
si_Collection*si_Query (si_QType, si_Scope s, si_Item* i)
```

Description

performs query

Return Value

pointer to `si_Collection` :returned by query operation contains all of the `si_Item`'s

Parameter

`si_QType` :type of query

`si_Scope` :scope to which the query is expanded

`si_Item` :item that the query is based on

Note

When `si_QType` is `eQFindSym`, you must set the second parameter (`si_Scope`) to the type (`si_Type`) that you want the Symbol Table to return.

Also, depending on your compiler, you may have to convert `si_Type` into `si_Scope` (by means of an explicit cast).

The following is an example of such an explicit cast:

```
(si_Scope) eTFile
```

Getting a hashed item

Function

```
si_item* getHashed (si_item*)
```

Description

```
si_item*X      :parameter
```

```
si_item*Y      :return value
```

```
si_item*Y = getHashed(si_item*X)
```

To access next item in collection, use `si_item*X`. To access data cached on the client side, use `si_item*Y`. If you use `si_item*Y -> next`, you will access a different collection.

Return Value

```
si_item*      :the item from the cache
```

Parameter

```
si_item*      :a cached item
```

4

Interpreting the query result

The results of your queries to the Symbol Table are data structures. The values of the elements of the data structures depend on the nature of the query made to the Symbol Table.

A general description of the elements is given followed by a description of the elements of the data structures after each type of query to the Symbol Table.

General description of the data structure elements

General description of the elements of `si_Item`

- `struct si_Item*next`
used in `si_Collection`
do not modify
- `struct si_Item*prev`
used in `si_Collection`
do not modify
- `si_uchar*name`
name of the item (string)
- `si_Type type`
type of the item (one of `si_Type`'s)
- `si_Resolve take`
see the description of `si_Resolve` below
- `si_ulong _hash`
do not modify

General description of the elements of `si_Resolve`

- `si_Ingredients*_incl`
dynamically points to `si_Ingredients` if `_hash == 0`
- `si_bool _hash`
if `_hash == 1`, `_incl` is null
if `_hash == 0`, `si_Item` is hashed. To get the value of the expanded pointer, call `getHashed(si_Item*)`

General description of the elements of `si_Ingredients`

- `si_Collection*subItems`
list sub items of this item
- `si_Collection*subClass`
list of sub classes for `eTClass`
- `si_Collection*superClass`
list of super classes for `eTClass`
- `struct si_Item*from`
in special cases a hashed item where this item comes from (e.g., nested class)

`si_Scope scope`

scope of the item (one of `si_Scope`)

- `si_ulongPos`
absolute start position in file
- `si_ulongPos`
absolute end position in file
- `si_uchar* typeStr`
type string as used in SNIFF+ (formatted string)

`si_uchar* declStr`

full string as appears in declaration (formatted string)

- `si_Flagflags`
flags of this item
use the `is...(item)` functions (e.g., `isOverloaded(si_Item*)`) to evaluate the flags
- `si_ulong__magic__cookie__`
do not modify

Description of the data structure elements when `si_Item` is `eTFile`

Description of selected elements of `si_Item`

- `struct si_Item*next`
empty
- `struct si_Item*prev`
empty
- `si_uchar*name`
name of file

- `si_Typetype`
eTFile

Description of selected elements of `si_Ingredients`

- `si_Collection *subItems`
all items in this file
- `si_uchar*typeStr`
language string of file (e.g., Ansi C/C++)
- `si_uchar*declStr`
absolute path of file

Description of the data structure elements when `si_Item` is `eTClass`

Description of selected elements of `si_Item`

- `si_uchar* name`
name of the class
- `si_Typetype`
eTClass

Description of selected elements of `si_Ingredients`

- `si_Collection*subItems`
list of all items of eTClass
- `si_Collection*subClass`
list of sub classes of eTClass
- `si_Collection*superClass`
list of all super classes whose node is preceded by a node of type `etScope`, which has the following structure:
Description of selected elements of `si_Item`
 - `si_uchar* name`
visibility of the inheritance (e.g., public, protected, etc.)
 - `si_Type type`
eTScope
Description of selected elements of `si_Ingredients`
 - `si_uchar* typeStr`
scope string (e.g., in class `B: public A::nested`, "A" is the scope string)

- `si_uchar* declStr`

template argument list string (e.g., in `class b: public A<X,Y,Z>`, “X,Y,Z” is the argument list string)

- `si_Flag flags`

Note

If inheritance is virtual, the corresponding flag (`isVirtual(si_Item*)`) is set

- `struct si_Item *from`
if scope is `eSFile`, `*from` is a null pointer.
if scope is `eSClass`, the class is nested, and `*from` is a hashed item to the class in which `eSClass` is nested
- `si_Scope scope`
`eSFile` or `eSClass`, see above

Description of the data structure elements when `si_Item` is a version configuration

Description of selected elements of `si_Item`:

- `si_uchar* name`
version of queried file
- `si_Type type`
`eTHead`, `eTSymbolicName`, `eTBranch`

Description of selected elements of `si_Ingredients`

- `si_uchar* declStr`
symbolic version string

Description of the data structure elements when `si_Item` is `eTMethodD`, `eTMethodI`, or `eTProc`

Description of selected elements of `si_Item`:

- `si_uchar* name`
name
- `si_Type type`
`eTMethodD`, `eTMethodI`, or `eTProc`

Description of selected elements of `si_Ingredients`

There is a special node in this collection:

Description of selected elements of `si_Item`

- `si_uchar*` name

`()`

- `si_Type` type

`eTParam`

Description of selected elements of `si_Ingredients`

- `si_ulong`sPos

starting position

- `si_ulong`ePos

ending position

In the following example:

```
func | (arguments) |
```

the pipes (`|`) indicate the starting and ending positions

- `si_Item*` from

In case the item is a `eTMethodI`, the `from` field is a pointer to the symbol `eTMethodD` in `SNIFF+`. To get the full representation of `eTMethodD`, call the following query:

```
si_Query(eQFindSym, eSGlobal, from);
```

Description of the data structure elements when `si_Item` is `eTParam`

Description of selected elements of `si_Item`:

- `si_uchar*` name

name

- `si_Type` type

`eTParam`

Description of selected elements of `si_Ingredients`

There is a special item `()` representing the start and end position of the argument brackets where the start and end position is stored in the `sPos` and `ePos` of the structure `si_ingredients`.

Description of selected elements of `si_Ingredients`

- `si_uchar*` typeStr

full string as stored in `SNIFF+`

- `si_uchar*` declStr

internal representation of the format character string used by `SNIFF+`

- `si_Flag eFConst`
for constant parameters
- `si_Flag eFTemplate`
for template parameters
- `si_Flag eFDefault`
for parameters that have a default value

For a description of the data structure elements of all other values of `si_Item`, please refer to [General description of the data structure elements — page 14](#).

5

Description of Queries

Hierarchy of a class

```
eQOOHierarchy
```

Description

Queries for the Hierarchy of one Item which is a class.

Example:

```
si_Collection*result= si_Query(eQOOHierarchy,eSGlobal,item);
```

Item is a class returned from a previous query.

Query Full Hierarchy

```
eQOOFullHierarchy
```

Description

Queries for the whole hierarchy in the loaded project.

Example:

```
si_Collection*result= si_Query(eQOOFullHierarchy,eSHierarchy,0);
```

Get rootclass

```
eQOOGetroot
```

Description

Get rootclass of Item, if NO_ITEM the root, or dummy root of hierarchy.

Example:

```
si_Collection*result= si_Query(eQOOGetroot, eSGlobal, item);
```

item is a class returned from a previous query.

Get Superclasses

eQ00GetSuper

Description

Get all Superclasses of Item, if NO_ITEM all superclasses found will be returned.

Example:

```
si_Collection*result= si_Query(eQ00GetSuper, eSGlobal, item);
```

item is a class returned from a previous query.

Get Subclasses

eQ00GetSub

Description

Get all Subclasses of Item.

Example:

```
si_Collection*result= si_Query(eQ00GetSub, eSGlobal, item);
```

item is a class returned from a previous query.

Classes referred by item

eQReferredBy

Description

Find all classes referred by Item.

Example:

```
si_Collection*result= si_Query(eQReferredBy, eSGlobal, item);
```

item is a method and procedure returned from a previous query.

Classes referring to item

eQReferresTo

Description

Find all classes referring to Item.

Example:

```
si_Collection*result= si_Query(eQReferresTo,eSGlobal,item);  
item is a symbol returned from a previous query.
```

Query for members

eQMembers

Description

Queries for all Members of specified Item.

Example:

```
si_Collection*result= si_Query(eQMembers,eSGlobal,item);  
item is a symbol returned from a previous query.
```

Find all symbols

eQFindSym

Description

Find all symbols in given scope.

Example:

```
si_Collection*result= si_Query(eQFindSym, eSGlobal, item);  
item is an empty item where only the name and the type are replaced with that of the  
symbol you want to search.
```

Find File

eQImplFiles

Description

Find all Items of type File in global scope.

Example:

```
si_Collection*result= si_Query(eQImplFiles, eSGlobal, 0);
```

Items, eProc and eMethodI references to

eQReferencesTo

Description

Find all Items, an eProc or eMethodI references to.

Example:

```
si_Collection*result= si_Query(eQReferencesTo, eSGlobal,  
    item);
```

item is a method and procedure returned from a previous query.

Items referring to an item

eQReferencedBy

Description

Find all Items referring to an Item, not a supported and tested feature.

Example:

```
si_Collection*result= si_Query(eQReferencedBy, eSGlobal,  
    item);
```

item is a symbol returned from a previous query.

Get version information

eQFileVersionConfig

Description

Get version information for a file

Example:

```
si_Item * req = si_cloneItem (fileItem);  
req->name = version;  
req->type = type;
```

fileItem	is an eTFile item from which you want to get the version control information
version	is the symbolic name of the version which you want to query e.g., HEAD
type	is either eTSymbolicName, eTHead or eTBranch

```
si_Collection*result= si_Query(eQFileVersionConfig, 0, req);
```