

SANKHYA Translation Framework™

API Overview and Reference Manual

THIS DOCUMENT CONTAINS PROPRIETARY INFORMATION OF SANKHYA TECHNOLOGIES PRIVATE LIMITED. Use, duplication and disclosure are subject to license restrictions.

(C) Copyright 2001-2004 Sankhya Technologies Private Limited

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means viz., electronic, mechanical, photo-copying, recording, or otherwise, without the prior consent of the publisher.

SANKHYA, SANKHYA TECHNOLOGIES, Dynamically Targetable Tools Framework, SANKHYA Translation Framework, SANKHYA Software are Trademarks, Service Marks or Registered Trademarks of Sankhya Technologies Private Limited. All other brands and names are the property of their respective owners.

Part No. 10003164-002

SANKHYA Translation Framework™

API Overview and Reference Manual

Sankhya Technologies Private Limited

Part No. 10003164-002

Table 1: Revision History

Revision number	Revision History	Date
001	Created for STF 1.0 Beta3 Release	20 Oct 2003
002	Updated the create_stream() function with the STMLPropertyList*pl=0 parameter	8 Dec 2004

Contents

Preface

Part 1 - API Overview	1
1.1 Introduction	1
1.1.1 Overview.....	1
1.1.2 Hosts supported.....	2
1.2 Setting up STF host development environment	2
1.3 Usage Example - ‘string_stream’ sample	3
1.4 Creating an application	7
1.5 Creating a new stream	10
Part 2 - Reference Manual	19
2.1 Introduction	19
2.1.1 STF Initialization	19
2.1.1.1 stml_init(int argc, char **argv)	19
2.1.2 STF Cleanup	20
2.1.2.1 stml_final().....	20
2.1.3 Input Processing.....	21
2.1.3.1 process_file(const char *file, ostream &o).....	21
2.1.3.2 process_stream(istream &i, ostream &o)	21
2.1.3.3 process_stream(STMLInputStream &i, ostream &o).....	22

2.1.3.4	process_buffer(char *buffer, ostream &o).....	23
2.1.3.5	process_input(const char *s, ostream &o).....	23
2.1.4	STF Class Library	25
2.1.4.1	DTTFSymbolTable.....	25
2.1.4.2	DTTFSymbolTableEntry	29
2.1.4.3	DTTFTextSymbol.....	30
2.1.4.4	STMLStream	33
2.1.4.5	STMLInputStream	34
2.1.4.6	STMLInputFileStream.....	38
2.1.4.7	STMLInputStringStream	42
2.1.4.8	STMLStreamFactory	46
2.1.4.9	STMLFileStreamFactory	48
2.1.4.10	STMLStringStreamFactory	49
2.1.4.11	STMLSymbolStreamFactory	50
2.1.4.12	STMLStreamFactoryManager	51

INDEX

Preface

SANKHYA Translation Framework (STF) is a completely novel framework for building model-driven translation/transformation tools and applications. It can be used to automate EAI activities like document and message processing, protocol conversion, SQL database to XML transformations, C++ and Java code generation, server page processing, data conversion and adapter development.

STF includes a powerful translation modeling language - SANKHYA Translation Modeling Language (STML) and a set of tools (STML Line Translator, STML Server, STF Application Programming Interface (API)) that automatically converts information in one format to any other format described using STML.

This document describes the STF API provided as part of the STF Developer Edition. It explains the steps involved in creating applications using the STF API and also provides details of the API functions and class library.

This document contains two parts:

Part-1 *API Overview* explains with examples the steps involved in using the STF API to create applications.

Part-2 *Reference Manual* provides the details of the STF API including the library functions and the STF class library.

Audience

This document is intended for application developers with C/C++ programming experience. Familiarity with GNU C/C++ compilers and/or Microsoft Visual C++ compiler is assumed.

Notational Conventions

The guide follows the following conventions

- % - The ‘percentage’ sign denotes a Unix environment.
- > - The ‘greater than’ symbol represents a DOS/Windows environment.
- Italic* - The words given in *italics* represents an option.
- code - The guide differentiates the normal text from a program code through this color. Any code, part of a code, input, output, command line statements in this document, will be represented using this color.
- ... - Indicates that some portion of the material has been removed to simplify the description.
- [] - Indicates an optional argument that can be used in the command line.

API OVERVIEW

Sankhya Technologies Private Limited

Part 1 - API Overview

1.1 Introduction

Translators are tools which convert information present in an input representation to equivalent information in an output representation using a set of rules supplied as input. Starting from language translations, a translation framework can be used in diverse application areas such as document conversion, script conversion, data exchange in Enterprise Application Integration (EAI).

SANKHYA Translation Framework (STF) is a completely novel framework for describing and performing model-driven translations and transformations. STF can be used for automating EAI activities like document and message processing, protocol conversion, data conversion, adapter development and natural language translation.

Application Programming Interface (API) is a list of function definitions which can be used in an application to accomplish an operation. The function implementations will generally be provided as libraries. This eliminates the re-invention of basic supported functions.

1.1.1 Overview

The STF Developer Edition libraries provide a C++ API that allows developers to develop complex model-driven applications. These applications can automatically convert information in one representation to any other representation specified in the model described using the SANKHYA Translation Modeling Language (STML).

1.1.2 Hosts supported

The following hosts are supported by SANKHYA Translation Framework Developer Edition.

- Solaris 2.7
- Red Hat Linux 7.2
- Windows NT and Windows 2000

1.2 Setting up STF host development environment

To set the STF host development environment, the following environment variables should be set.

STF_HOME - Directory path pointing to the root of installation
LD_LIBRARY_PATH - Directory path of library files for Unix hosts.

For Unix hosts:

STF_HOME is <INSTALL_DIR>/sankhya/stf

LD_LIBRARY_PATH is /usr/local/lib:\$(STF_HOME)/lib/\$(HOST)

where,

HOST - Host name like sol2, linux.

For Windows hosts:

STF_HOME is <INSTALL_DIR>

where,

<INSTALL_DIR> is the directory where STF tools are installed.

By default STF, is installed under /opt in Unix and C:\sankhya\stf in Windows NT/2000.

In order to set this variable and PATH variable on Unix, source the Unix shell script *stf.csh* in the STF tools installation root directory.

```
% source <STF_HOME>/stf.csh
```

On Windows host, run *stf.cmd*, in the STF tools installation directory. In a Windows Command Prompt, type

```
> %STF_HOME%\stf.cmd
```

1.3 Usage Example - 'string_stream' sample

STF Developer Edition samples using STF API can be found under the directory \$(STF_HOME)/samples/applications. The 'string_stream' sample explained below uses the string stream feature of STF.

Here is the complete overview of the steps to be followed to use and STF based application.

- Step-1.** Create the application.
- Step-2.** Build the application
- Step-3.** Create a model file and input file for processing.
- Step-4.** Invoke the application to perform translation.

The below section explains each step specified in the overview in details for a string stream sample.

Step-1. Create the application.

Here is the contents of the application, st.cc

```
#include <stdlib.h>
#include <fstream>
#include "st.h"
#include "STMLInputStringStream.h"

int main()
{
    int arg_count = 3;
    char* arg[17] = {"st", "-m", "string_stream.md"};

    // Perform initializations.
    stml_init(arg_count, arg);

    STMLInputStringStream f("Mercadoria", 11);

    process_stream(f, cout);

    // Perform cleanup actions
    stml_final();
    return (0);
}
```

Step-2. Build the application.

On Unix, compile the application 'st.cc' as follows.

```
%g++ -o st st.cc -fPIC -Wall -ansi -pedantic -I $(STF_HOME)/include
$(STF_HOME)/lib/$(HOST)/stf.so $(STF_HOME)/lib/$(HOST)/libst.so -ldl
```

where,

STF_HOME - Directory path pointing to the root of STF installation.

HOST - Host name like sol2, linux.

On Windows, compilation can be done using VC++ as follows.

```
>cl.exe /nologo /ML /W3 /GR /GX /O2 /I "$(STF_HOME)\include"
/D "WIN32" /D "NDEBUG" /D "_CONSOLE" /D "_MBCS" /TP st.cc /link
/defaultlib:netapi32.lib $(STF_HOME)\lib\win32\libst.lib
$(STF_HOME)\lib\win32\stf.lib /nodefaultlib:libcd.lib
```

Step-3. Create the model file.

Contents of the model file string_stream.md

```
STMLTextTokens pr = { "Mercadoria", "destinada", "para", "consumo", "na",
"zona", "da", "Chennai" };
```

```
STMLTextTokens en = { "Merchandise", "destined", "for", "consumption", "in",
"zone", "the", "Chennai" };
```

```
STMLModel Translator {
```

```
STMLLeaf tr {  
  
    range (i = 0,7,1) {  
        input = { "$pr[$i]" };  
        output = { "$en[$i]" };  
    };  
  
};  
  
STMLNode Document {  
  
    tr s;  
  
    input = { s };  
    output = { s };  
};  
  
};
```

The above sample creates a stream for the string ‘Mercadoria’ and the input word to be translated will be obtained from the created string stream.

Step-4. Invoke the application to perform translation.

On Unix hosts,

```
%st
```

On Windows,

```
>st
```

The following output will be displayed in the screen.

```
Merchandise
```

1.4 Creating an application

STF Developer Edition libraries provides a powerful set of API to develop model driven applications. The following are the steps involved in creating a simple C++ application using STF API.

Step 1: Set the STF host development environment and C++ compiler (Microsoft Visual C++/GNU C++) environment.

Step 2: In the application, invoke the following function to initialize the STF library environment and to process the command-line arguments.

```
stml_init(int argc, char **argv)
```

Step 3: Invoke the following function to translate input from a file and redirect the output to a output stream.

```
process_input(const char *file, ostream &o)
```

If the argument 'o' is set to the C++ standard output stream 'cout', then the translated output will be obtained in the console.

Step 4: Perform cleanup of the memory and files used by the STF library, when no longer required, by invoking the following function.

```
stml_final ();
```

Following is an example of a simple application ‘st_app.cc’. The below application will parse the input from the source file specified in the command line and the translated output will be displayed in the console.

Example:

```
//st_app.cc
int main(int argc, char **argv)
{
    // Process command line arguments and perform other initialization
    stml_init(argc, argv);

    // Process the input and translate
    process_input(sourcefile, cout);

    //Cleanup action
    stml_final();
}
```

Step 5: Compile the created application ‘st_app.cc’ on Unix as follows.

```
% g++ -o st_app st_app.cc -fPIC -Wall -ansi -pedantic -I $(STF_HOME)/include
$(STF_HOME)/lib/$(HOST)/stf.so $(STF_HOME)/lib/$(HOST)/libst.so -ldl
```

where,

STF_HOME - Directory path pointing to the root of STF installation.

HOST - Host name like sol2, linux.

On Windows, compilation can be done using VC++ as follows.

```
>cl.exe /nologo /ML /W3 /GR /GX /O2 /I "$(STF_HOME)\include" /D "WIN32"  
/D "NDEBUG" /D "_CONSOLE" /D "_MBCS" /TP st_app.cc  
/link defaultlib:netapi32.lib $(STF_HOME)\lib\win32\libst.lib  
$(STF_HOME)\lib\win32\stf.lib /nodefaultlib:libcd.lib
```

The demo makefile provided as part of the directory \$(STF_HOME)/samples applications can be modified to build user applications.

Step 6: Invoke the application with the model file ‘test.md’ and input file ‘test.in’.

On Unix hosts,

```
%st_app -m test.md test.in
```

On Windows,

```
>st_app -m test.md test.in
```

The input file ‘test.in’ contents will be parsed and translation will be performed using the model file test.md. For more information on creating a model file, please refer “*SANKHYA Translation Framework User Guide and Reference Manual*”.

1.5 Creating a new stream

During translation, certain applications require inputs to be read from multiple streams (of different types) at different point of time. To support this, STF API supports the addition and integration of a new stream with STF.

STML Stream construct allows the specification of input streams from which data should be obtained during translation. The format for stream specification is as follows:

stream STRING '{' value_list '}'

where,

STRING is a quoted string of the form “**stream-id:path**”

where,

stream-id - string which identifies the stream type.

Example: 'file'.

path - specifies the location of the stream source or destination.

Example: 'path' could be the path name for file stream.

Here is the complete overview of the steps involved in the integration of a new stream with the STF.

Step 1: Develop the abstraction for the stream. If the stream is an input stream, then derive it from 'STMLInputStream'.

Step 2: Create a stream factory for that stream which should extend the 'STMLStreamFactory' class and implement the 'create_stream()' function.

Step 3: In the application, register the stream factory of the new stream by invoking,

```
STMLStreamFactoryManager::register_stream_factory(str,factory)
```

where,

str - identifier associated with the new stream.

The following explains each step specified in the overview in detail for adding a ODBC stream to STF.

Step 1:

Create a database stream class 'DataBaseStream' as a derived class of 'STMLInputStream'. The following is the basic structure of the database stream class.

Example:

```
class DataBaseStream : public STMLInputStream
{
    // define member functions
};
```

Implement the stream functionality in the argument constructor of the 'DataBaseStream' class and store the result in a stream.

For database stream, the functional implementation consists of the following steps.

- a. Establish a ODBC Connection
- b. Execute the query and retrieve the result.

Example:

```
//DataBaseStream.cc

//Implement the functionality of the stream
DataBaseStream::DataBaseStream(char * stream) : STMLInputStream()
{

    //Get the connection string from the stream object and establish the ODBC
    //connection
    string constr=get_connection_string(stream);
    establish_connection(constr);

    //Get the query string from the stream object and execute the query
    string q_str=get_query_string(stream);
    execute_query(q_str);

    //Retrieve the result of the executed query
    output=get_result();
    fs=new istrstream(output,strlen(output));

}
```

Implement all the virtual functions of the `STMLInputStream` in the derived class 'DataBaseStream'. Consider the function 'read' in `STMLInputStream` class. This can be implemented as explained below.

Example:

```
//DataBaseStream.cc
void DataBaseStream::read(stream_type* buffer, int n)
{
    fs->read(buffer,n);
}
```

Similarly implement all the virtual functions of the STMLInputStream class in the derived class DataBaseStream.

Step 2:

Create a stream factory for the database stream which should extend the 'STMLStreamFactory' class. Implement the 'create_stream()' function in the created streamfactory to provide an access to the 'DataBaseStream' class.

Example:

```
//DataBaseStreamFactory.cc
class DataBaseStreamFactory : public STMLStreamFactory{
    virtual STMLStream* create_stream(const char* s,
                                     DTTFSTable *st,
                                     STMLPropertyList* pl = 0)
    {
        //parse the string and store the required stream format
        s_val=st_manip(s);

        // create and return the new Stream object
        return new DataBaseStream(s_val);
    }
}
```

```
};
};
```

In the above class, 's_val' should contain the stream declaration in a format compatible with the 'DataBaseStream' class.

Step 3:

Define the function 'get_stream_factory()' in the created stream factory class.

Example:

```
//DataBaseStreamFactory.cc
extern "C" STMLStreamFactory* get_stream_factory()
{
    return new DataBaseStreamFactory();
}
```

For streams that are statically linked to the application, the above function is not necessary.

Step 4:

Create the application and register the stream factory of the new stream by invoking,

```
STMLStreamFactoryManager::register_stream_factory(str,factory)
```

where ,

str - identifier associated with the new stream.

The following shows a sample STF application that uses the database stream created

above.

Example:

```
//st_app.cc
int main(int argc, char **argv)
{

    //STML Initalization
    stml_init(argc, argv);

    //Register the DataBaseStream Factory
    DataBaseStreamFactory *dbf = new DataBaseStreamFactory();
    STMLStreamFactoryManager::register_stream_factory("db",dbf);

    //Process the input
    process_input(sourcefile,cout);

    //Perform cleanup actions
    stml_final();
    return(0);
}
```

Now STF will be able to use the DataBaseStreamFactory for obtaining DataBaseStream objects when stream "db:.." declarations are encountered in the input.

Alternatively, the new stream can be implemented as a dynamically linked library and can be registered with the stream factory manager as follows:

```
STMLStreamFactoryManager::register_stream_factory("db","libdb.so");
```

Step 5:

Set the STF host development environment and C++ compiler (Microsoft Visual C++/GNU C++) environment for building the database stream and the application. Please refer section 1.2 for details on setting up environment for STF.

Step 6:

Generate the library for the created database stream.

Create the library on Unix as follows using GCC.

```
% g++ -c -I $(STF_HOME)/include -o DataBaseStreamFactory.o -g
  DataBaseStreamFactory.cc
```

```
% g++ -c -I $(STF_HOME)/include -o DataBaseStream.o -g DataBaseStream.cc
```

```
% g++ -shared -o libdbstream.so DataBaseStreamFactory.o DataBaseStream.o -ldl
  -ldb
```

Create the library on Windows as follows using VC++.

```
> cl.exe /nologo /MT /W3 /GR /GX /O2 /I "$(STF_HOME)\include" /D "WIN32"
  /D "NDEBUG" /D "_WINDOWS" /D "_MBCS" /D "_USRDLL" /c /Tp
  STMLDataBaseStream.cc
```

```
> cl.exe /nologo /MT /W3 /GR /GX /O2 /I "$(STF_HOME)\include" /D "WIN32"
  /D "NDEBUG" /D "_WINDOWS" /D "_MBCS" /D "_USRDLL" /c /Tp
  STMLDataBaseStreamFactory.cc
```

```
> link.exe kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
$(STF_HOME)\lib\win32\libst.lib /nologo /dll /incremental:no /machine:I386
/nodfaultlib:"libcd.lib" /out:"libdbstream.dll" /implib:"libdbstream.lib"
STMLDataBaseStream.obj STMLDataBaseStreamFactory.obj
```

Step8:

Build and test the application 'st_app'.

On Unix, build the application using g++ as follows.

```
%g++ -o st -I $(STF_HOME)/include st_app.cc $(STF_HOME)/lib/$(HOST)/stf.so
$(STF_HOME)/lib/$(HOST)/libst.so ./libdbstream.so -ldl
```

where,

HOST - Host name like sol2, linux.

On Windows, compilation using VC++ can be done as follows.

```
>cl.exe /nologo /ML /W3 /GR /GX /O2 /I "$(STF_HOME)\include" /D "WIN32"
/D "NDEBUG" /D "_CONSOLE" /D "_MBCS" /TP st_app.cc /link
/defaultlib:netapi32.lib $(STF_HOME\lib\win32\libst.lib
$(STF_HOME)\lib\win32\stf..lib libdbstream.lib /nodfaultlib:libcd.lib
```

Create a model file 'db_html.md' with the stream declaration of type "db:...." .
Please refer "*SANKHYA Translation Framework User Guide and Reference*

Manual” for information on creating a model file.

Then the application 'st_app' can be tested as follows.

On Unix hosts,

```
%st_app -m db_html.md db_html.in
```

On Windows,

```
>st_app -m db_html.md db_html.in
```

where,

st_app - application using database stream

db_html.md - model file for database format to HTML conversion

db_html.in - input file

REFERENCE MANUAL

Sankhya Technologies Private Limited

Part 2 - Reference Manual

2.1 Introduction

STF API enables the developers to develop complex model driven applications using SANKHYA Translation Modeling Language (STML) that can automatically convert information in one representation to any other representation specified in the model file. The STF API library functions and classes are described below.

2.1.1 STF Initialization

When an application uses STF environment, it needs to be initialized. Once an application and its command line arguments are initialized in the STF environment, the input file can be processed and translated using the STML specification.

2.1.1.1 `stml_init(int argc, char **argv)`

Initializes the environment for translation. Processes the command line arguments, parses the input model description and registers basic stream factories.

Example:

```
int main(int argc, char **argv)
{
    //Initialize the STML environment
    stml_init(argc, argv);
    .....
}
```

2.1.2 STF Cleanup

When the input is translated using the STML specification, cleanup of the memory and files used by the STF library needs to be performed.

2.1.2.1 stml_final()

Performs final cleanup of memory, buffer used for translation.

Example:

```
int main(int argc, char **argv)
{
    //Process command line arguments
    stml_init(argc, argv);
    .....
    .....
    //Cleanup action
    stml_final();
}
```

2.1.3 Input Processing

The input from a file, stream or buffer are parsed and translated using the STML specification. The STF API library functions that performs the parsing of the input are described below.

2.1.3.1 `process_file(const char *file, ostream &o)`

Translates the input from 'file' according to prespecified model and provides the result in output stream 'o'.

Example:

```
int main(int argc, char **argv)
{
    //STML Initialization
    ....
    //Process file and translate
    process_file("a.txt", cout);
    ....
    //Cleanup action
}
```

2.1.3.2 `process_stream(istream &i, ostream &o)`

Translates the input from istream according to prespecified model and provides the result in output stream 'o'.

Example:

```
int main(int argc, char **argv)
{
    //STML Initialization
    ....
    //Create a stream
    istream f("test.in", ios::in);

    //Process Stream
    process_stream(f, cout);
    ....
    //Cleanup action
}
```

2.1.3.3 process_stream(STMLInputStream &i, ostream &o)

Translates the input from STMLInputStream according to prespecified model and provides the result in output stream 'o'.

Example:

```
int main(int argc, char **argv)
{
    //STML Initialization
    ....

    STMLInputFileStream f("test.txt", ios::in);
}
```

```
    //Process Stream
    process_stream(f, cout);
    ....
    //STML Cleanup
}
```

2.1.3.4 process_buffer(char *buffer, ostream &o)

Translates the input from 'buffer' according to prespecified model and provides the result in output stream 'o'.

Example:

```
int main(int argc, char **argv)
{
    //STML Initialization
    ....
    //Process the buffer and translate
    process_buffer(char *bfr, cout);
    ....
    //STML Cleanup

}
```

2.1.3.5 process_input(const char *s, ostream &o)

Translates the input obtained from the stream specified by 's' and places the result in the output stream 'o'.

The input stream 's' can be specified as follows:

[<stream-type>:]<stream-source>

where,

- <stream-type> - file, str, sym, user-defined
- <stream-source> - file path, string, symbol name

If stream-type is empty, then file stream is assumed. If 's' is null, then the standard input stream (cin) is used.

Example:

```
int main(int argc, char **argv)
{
    //STML Initialization
    ....
    //Process input file and translate
    process_input("file:test.in", cout);
    ...
}
```

2.1.4 STF Class Library

2.1.4.1 DTTFSymbolTable

Synopsis:

```
class DTTFSymbolTable
{
public:

    typedef DTTFSymMap::iterator Iterator;

    DTTFSymbolTable();

    DTTFSymbolTable(const DTTFSymbolTable& t );

    ~DTTFSymbolTable();

    void copy(DTTFSymbolTable *st, bool overwrite = true);

    void add_symbol(string s);

    void add_symbol(DTTFSymbolTableEntry* sym);

    DTTFSymbolTableEntry* get_symbol(string sym);

    DTTFSymbolTableEntry* get_symbol(DTTFSymbolTable::Iterator it);

    void delete_symbol(string s);
```

```
bool find_symbol(string sym);

void print();

int size();

Iterator begin();

Iterator end();

};
```

Description:

A symbol table class that holds symbols of type DTTFSymbolTableEntry.

Base Classes:

None

Member Functions:

Method	Description
DTTFSymbolTable()	Constructs a DTTFSymbolTable object.
DTTFSymbolTable(const DTTF-SymbolTable& t)	Constructs a DTTFSymbolTable which is a copy of 't'.

Method	Description
<code>~DTTFSymbolTable()</code>	Destructor
<code>void copy(DTTFSymbolTable *st, bool overwrite = true)</code>	Copies the contents of DTTFSymbolTable 'st' to this object. Existing symbols of the same name will be overwritten with new values if 'overwrite' is true. If 'overwrite' is 'false', existing values will be retained.
<code>void add_symbol(string s)</code>	Adds symbol 's' with empty value into the symbol table.
<code>void add_symbol(DTTFSymbolT- ableEntry* sym)</code>	Adds symbol table entry 'sym' into the symbol table.
<code>DTTFSymbolTableEntry* get_symbol(string sym)</code>	Returns the symbol table entry associated with symbol 'sym'.
<code>DTTFSymbolTableEntry* get_symbol(DTTFSymbolT- able::Iterator it)</code>	Returns the symbol table entry at the location pointed by iterator 'it'.
<code>void delete_symbol(string s)</code>	Deletes the symbol table entry for the symbol 's'.

Method	Description
<code>bool find_symbol(string sym)</code>	Returns true if symbol 'sym' is found in the table. Otherwise returns false.
<code>void print()</code>	Prints the contents of the symbol table.
<code>int size()</code>	Returns the number of entries in the symbol table.
<code>Iterator begin()</code>	Returns an iterator to the first entry of the symbol table.
<code>Iterator end()</code>	Returns an iterator to the last entry of the symbol table. This can be used as the termination value during iteration.

2.1.4.2 DTTFSymbolTableEntry

Synopsis:

```
class DTTFSymbolTableEntry
{
    public:

        virtual string get_name() = 0;

        virtual void set_name(string s) = 0;

        virtual string get_value() = 0;

        virtual void set_value(string s) = 0;

        virtual void print() = 0;

};
```

Description:

An abstract class for symbols that can be placed in the symbol table.

Base Classes:

None

Member Functions:

Method	Description
virtual string get_name() = 0	Returns the name of the symbol.
virtual void set_name(string s) = 0	Sets the symbol name to 's'.
virtual string get_value() = 0	Returns the value associated with the symbol.
virtual void set_value(string s) = 0	Sets the symbol's value to 's'.
virtual void print() = 0	Prints the symbol table entry.

2.1.4.3 DTTFTextSymbol**Synopsis:**

```

class DTTFTextSymbol : public DTTFSymbolTableEntry
{
public:

    DTTFTextSymbol();

    DTTFTextSymbol(string in_name);

```

```
DTTFTextSymbol(string in_name, string in_value);

virtual ~DTTFTextSymbol();

string get_name();

void set_name(string in_name) ;

string get_value();

void set_value(string in_value);

void print();

};
```

Description:

Represents a text symbol. This class contains the implementation for the class DTTFSymbolTableEntry.

Base Classes:

DTTFSymbolTableEntry

Member Functions:

Method	Description
DTTFTextSymbol()	Constructs a DTTFTextSymbol with empty name and value.
DTTFTextSymbol(string in_name)	Constructs a DTTFTextSymbol with name 'in_name' and empty value.
DTTFTextSymbol(string in_name, string in_value)	Constructs a DTTFTextSymbol with name 'in_name' and value 'in_value'.
virtual ~DTTFTextSymbol()	Destroys the DTTFTextSymbol Object.
string get_name()	Returns the name of the symbol.
void set_name(string in_name)	Sets the name of the symbol.
string get_value()	Returns the value associated with the symbol.
void set_value(string in_value)	Sets the value associated with the symbol.
void print()	Prints the symbol.

2.1.4.4 STMLStream

Synopsis:

```
class STMLStream
{
    public:

        typedef char stream_type;

        enum stream_kind { sk_input, sk_output, sk_inout };

        virtual ~STMLStream();

        virtual bool is_eos() const = 0;

        virtual bool fail() const = 0;

        virtual void clear() = 0;

};
```

Description:

Represents a stream that can be a source or sink of data.

Base Classes:

None.

Member Functions:

Method	Description
virtual ~STMLStream()	Virtual destructor to destroy the created STMLStream.
virtual bool is_eos() const = 0;	Returns true if end of stream is reached. Otherwise returns false.
virtual bool fail() const = 0;	Returns true if further operations on the stream are bound to fail. Otherwise, returns false.
virtual void clear() = 0;	Clears the state flags of the stream.

2.1.4.5 STMLInputStream**Synopsis:**

```

class STMLInputStream : virtual public STMLStream
{

public:

    STMLInputStream();

    STMLInputStream(istream *i);

```

```

virtual ~STMLInputStream();

virtual void read(stream_type* buffer, int n);

virtual void seekg(long pos, ios_base::seekdir d);

virtual long tellg() const;

virtual bool is_eos() const;

virtual bool fail() const;

virtual void clear();
};

```

Description:

Represents an input stream, a source of data .

Base Classes:

STMLStream

Member Functions:

Method	Description
STMLInputStream()	Constructs a STMLInputStream Object

Method	Description
<code>STMLInputStream(istream *i)</code>	Constructs a STMLInputStream object with the specified input stream.
<code>virtual ~STMLInputStream()</code>	Destroys the STMLInputStream Object
<code>virtual void read(stream_type* buffer, int n)</code>	Reads 'n' bytes from the source and places them in 'buffer'.
<code>virtual void seekg(long pos, ios_base::seekdir d);</code>	Seeks to position 'pos' in the stream. This form of the member function is specific to unix host. For more information on the parameter 'd', refer the windows specific form of this function.

Method	Description
virtual void seekg(long pos, ios::seek_dir d);	<p>Seeks to position 'pos' in the stream.</p> <p>The parameter 'd' specifies the base from which seeking is to be done:</p> <p>ios::beg - seek from the beginning of stream.</p> <p>ios::cur - seek from the current offset in the stream.</p> <p>ios::end - seek from the end of stream.</p> <p>This form of the member function is specific to windows.</p>
virtual long tellg() const;	Returns the current position of the stream pointer.
virtual bool is_eos() const;	Returns true if end of stream is reached. Otherwise returns false.
virtual bool fail() const;	Returns true if further operations on the stream are bound to fail. Otherwise, returns false.
virtual void clear();	Clears the state flags of the stream.

2.1.4.6 STMLInputFileStream

Synopsis:

```
class STMLInputFileStream : public STMLInputStream
{
public:

    STMLInputFileStream();

    STMLInputFileStream(ifstream* f);

    STMLInputFileStream(const char *path, int mode, int prot=0664);

    ~STMLInputFileStream();

    void open(const char *name);

    virtual void read(stream_type* buffer, int n);

    bool is_open();

    void close();

    virtual bool is_eos() const;

    virtual bool fail() const;

    virtual void clear();
```

```

void seekg(long pos, ios::seek_dir d);

long tellg() const;

};

```

Description:

Represents an input file stream.

Base Classes:

STMLInputStream

Member Functions:

Method	Description
STMLInputFileStream()	Constructs an STMLInputFileStream object.
STMLInputFileStream(ifstream* f)	Constructs an STMLInputFileStream object from the ifstream 'f'.
STMLInputFileStream(const char *path, int mode, int prot=0664)	Constructs an STMLInputFileStream object and associates it with the file whose path is specified by 'path'. The file is opened with mode 'mode' and permissions 'prot'.
~STMLInputFileStream()	Destroys the STMLInputFileStream object.

Method	Description
<code>void open(const char *name)</code>	Opens the file specified by 'path' and associates it with this stream.
<code>virtual void read(stream_type* buffer, int n)</code>	Reads 'n' bytes from the associated file and stores them in 'buffer'.
<code>bool is_open()</code>	Returns true if the associated file is open. Otherwise, returns false.
<code>void close()</code>	Closes the file stream.
<code>virtual bool is_eos() const</code>	Returns true if end of stream is reached. Otherwise returns false.
<code>virtual bool fail() const</code>	Returns true if further operations on the stream are bound to fail. Otherwise, returns false.
<code>virtual void clear()</code>	Clears the state flags of the stream.

Method	Description
void seekg(long pos, ios::seek_dir d)	<p>Seeks to position 'pos' in the stream.</p> <p>The parameter 'd' specifies the base from which seeking is to be done:</p> <p>ios::beg - seek from the beginning of stream</p> <p>ios::cur - seek from the current offset in the stream</p> <p>ios::end - seek from the end of stream.</p> <p>This form of the member function is specific to windows.</p>
void seekg(long pos, ios_base::seek_dir d)	<p>Seeks to position 'pos' in the stream. This form of the member function is specific to unix hosts.</p> <p>For more information on the parameter 'd', please refer the windows specific form of this function.</p>
long tellg() const	<p>Returns the current position of the stream pointer.</p>

2.1.4.7 STMLInputStringStream

Synopsis:

```
class STMLInputStringStream : public STMLInputStream
{
public:

    STMLInputStringStream();

    STMLInputStringStream(istream* s);

    STMLInputStringStream(char *buf, int len);

    ~STMLInputStringStream();

    virtual void read(stream_type* buffer, int n);

    void close();

    virtual bool is_eos() const;

    virtual bool fail() const;

    virtual void clear();

    void seekg(long pos, ios::seek_dir d);

    long tellg() const;
```

```

char* str() const;

};

```

Description:

Represents an input string stream.

Base Classes:

STMLInputStream

Member Functions:

Method	Description
STMLInputStringStream()	Constructs an STMLInputStringStream object.
STMLInputStringStream(istream* s)	Constructs an STMLInputStringStream object from the istream object 's'.
STMLInputStringStream(char *buf, int len)	Constructs an STMLInputStringStream object from the char buffer 'buf' of size 'len'.
~STMLInputStringStream()	Destroys the STMLInputStringStream object.
virtual void read(stream_type* buffer, int n)	Reads 'n' bytes from the associated file and stores them in 'buffer'.

Method	Description
void close()	Closes the file stream.
virtual bool is_eos() const	Returns true if end of stream is reached. Otherwise returns false.
virtual bool fail() const	Returns true if further operations on the stream are bound to fail. Otherwise, returns false.
virtual void clear()	Clears the state flags of the stream.
long tellg() const	Returns the current position of the stream pointer.
void seekg(long pos, ios::seek_dir d)	<p>Seeks to position 'pos' in the stream.</p> <p>The parameter 'd' specifies the base from which seeking is to be done:</p> <ul style="list-style-type: none"> ios::beg - seek from the beginning of the stream ios::cur - seek from the current offset in the stream ios::end - seek from the end of stream. <p>This form of the member function is specific to windows host.</p>

Method	Description
void seekg(long pos, ios_base::seekdir d);	<p>Seeks to position 'pos' in the stream. This form of the member function is specific to unix host.</p> <p>For more details on the parameter 'd', please refer the windows specific form of this function.</p>

2.1.4.8 STMLStreamFactory

Synopsis:

```
class STMLStreamFactory
{

public:

virtual STMLStream* create_stream(const char* s,
                                DTTFSymbolTable* st,
                                STMLPropertyList* pl = 0) = 0;

};
```

Description:

A stream factory class to create different STMLStream objects.

Base Classes:

None

Member Functions:

Method	Description
virtual STMLStream* create_stream(const char* s, DTTFSTable* st, STMLPropertyList* pl =0) = 0	<p>Creates a STMLStream object and returns it. Returns a NULL pointer if stream cannot be created.</p> <p>The parameter 's' specifies the stream identifier and stream source information in the following format:</p> <p style="text-align: center;">"<id>:<stream source>"</p> <p>where,</p> <p style="margin-left: 40px;"><id> - stream identifier</p> <p style="margin-left: 40px;"><stream source> - stream source location.</p> <p>The parameter 'st' specifies a symbol table which can be used to resolve any symbols specified in 's'.</p> <p>The parameter 'pl' specifies an optional list of properties.</p>

2.1.4.9 STMLFileStreamFactory

Synopsis:

```
class STMLFileStreamFactory : public STMLStreamFactory
{
public:

    virtual STMLStream* create_stream(const char* s,
                                     DTTFSymbolTable *st,
                                     STMLPropertyList* pl=0);

};
```

Description:

A factory for creating file stream objects.

Base Classes:

STMLStreamFactory

Member Functions:

Method	Description
virtual STMLStream* create_stream(const char* s, DTTFSTable *st, STMLPropertyList* pl = 0)	Creates a file stream object and returns it. The string 's' should be of the form "file:<path>" where, <path> - Path of the file Example: file:./test.txt The parameter 'pl' specifies an optional list of properties.

2.1.4.10 STMLStringStreamFactory

Synopsis:

```

class STMLStringStreamFactory : public STMLStreamFactory
{
public:

    virtual STMLStream* create_stream(const char* s,
                                     DTTFSTable *st,
                                     STMLPropertyList* pl = 0);

};

```

Description:

A factory for creating string stream objects.

Base Classes:

STMLStreamFactory

Member Functions:

Method	Description
virtual STMLStream* create_stream(const char* s, DTTFSymbolTable *st, STMLPropertyList* pl = 0)	<p>Creates a string stream object and returns it. Returns a NULL pointer if a stringstream could not be created.</p> <p>The string 's' should be of the form: "str:string list"</p> <p>Example: str:Mercadoria</p> <p>The parameter 'pl' specifies an optional list of properties.</p>

2.1.4.11 STMLSymbolStreamFactory**Synopsis:**

```
class STMLSymbolStreamFactory : public STMLStreamFactory
{
public:

virtual STMLStream* create_stream(const char* s,
                                DTTFSymbolTable *st,
                                STMLPropertyList* pl = 0);
```

```
};
```

Description:

A factory for creating a symbol stream.

Base Classes:

STMLStreamFactory

Member Functions:

Method	Description
virtual STMLStream* create_stream(const char* s, DTTFSymbolTable *st, STMLPropertyList* pl = 0)	<p>Creates a STMLStream object from the value of the symbol specified by 's' and returns it. Returns a NULL pointer if the stream could not be created. The symbol should be installed in the symbol table 'st' previously.</p> <p>The string 's' should be of the form: "sym:<symbol>".</p> <p>Example: <code>sym:portugese_text</code></p> <p>The parameter 'pl' specifies an optional list of properties.</p>

2.1.4.12 STMLStreamFactoryManager**Synopsis:**

```
class STMLStreamFactoryManager
```

```
{  
    public:  
  
        static void register_stream_factory(const char* id, STMLStreamFactory* f);  
  
        static void register_stream_factory(const char* id, const char* lib);  
  
        static STMLStreamFactory* get_stream_factory(const char* id);  
  
        static void unregister_stream_factory(const char* id);  
  
};
```

Description:

A class for managing STMLStreamFactory objects.

Base Classes:

None.

Member Functions:

Method	Description
static void register_stream_factory(const char* id, STMLStreamFactory* f)	Registers the stream factory 'f' and associates the stream identifier 'id' with the factory.
static void register_stream_factory(const char* id, const char* lib)	Registers the library 'lib' and associates the stream identifier 'id' with the library. The library should contain the stream and fac- tory implementation.
static STMLStreamFactory* get_stream_factory(const char* id)	Returns the stream factory associated with stream identifier 'id'. If no stream factory of this type is registered then returns a NULL pointer.
static void unregister_stream_factory(const char* id);	Unregisters the stream factory associated with stream type 'id'.

INDEX

A

Application Programming Interface 1

D

DTTFSymbolTable 25

DTTFSymbolTableEntry 29

DTTFTextSymbol 30

E

Enterprise Application Integration 1

P

process_buffer 23

process_file 21

process_input 23

process_stream 21, 22

S

SANKHYA Translation Framework 1

STF 1

STML Stream 10

stml_final 20

stml_init 19

STMLFileStreamFactory 48

STMLInputFileStream 38

STMLInputStream 34

STMLInputStringStream 42

STMLStream 33

STMLStreamFactory 46

STMLStreamFactoryManager 51

STMLStringStreamFactory 49

STMLSymbolStreamFactory 49

For More Information-

STF Download	http://www.sankhya.com/info/products/data/download.html
STF Documentation	http://www.sankhya.com/info/products/data/docs.html
STF Sales & Support	sales@sankhya.com

SANKHYA™

Sankhya Technologies Private Limited
#13/2, "JayaShree", Third Floor, First Street, Jayalakshmipuram,
Nungambakkam,
Chennai 600 034, INDIA
Tel: +91 44 2822 7358
Fax: +91 44 2822 7357

Sankhya Technologies India Operations Private Limited
#30-15-58, "Silver Willow", Third Floor,
Dabagardens,
Visakhapatnam 530 020, INDIA
Tel: +91 891 554 2666
Fax: +91 891 554 2665
Email: sales@sankhya.com
<http://www.sankhya.com>

SANKHYA, SANKHYA TECHNOLOGIES, SANKHYA Translation Framework, Dynamically Targetable Tools Framework, SANKHYA Software are Trademarks, Service Marks or Registered Trademarks of Sankhya Technologies Private Limited. OMG marks and logos are trademarks or registered trademarks, service marks and/or certification marks of Object Management Group, Inc. registered in the United States or other countries. All other brands and names are the property of their respective owners.
