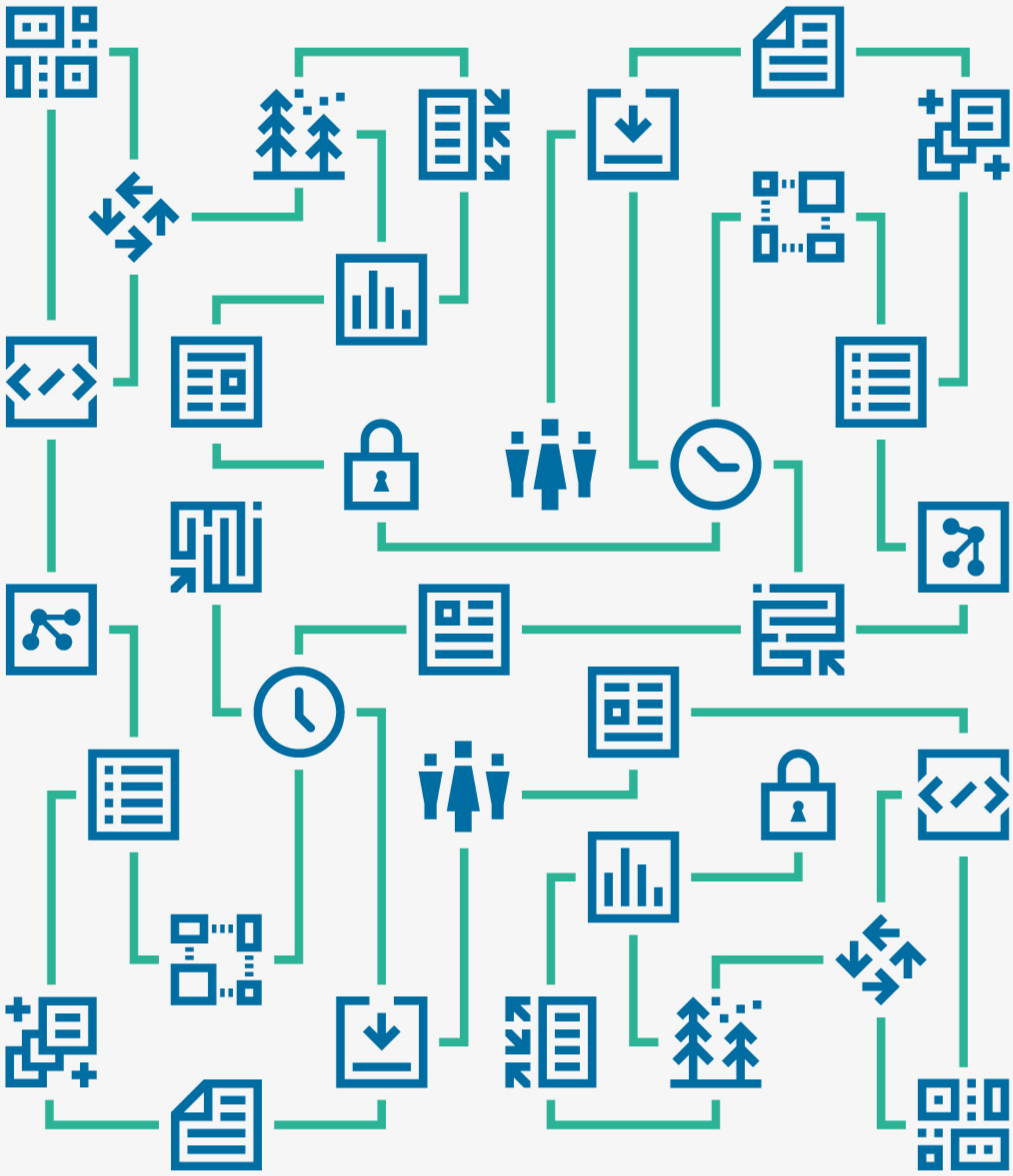
Starting Guide

Sparksee by **Sparsity Technologies**



Contents

Getting started

This is the starting guide for Sparksee graph database. We will guide you during the creation of your first Sparksee graph, from downloading Sparksee to the execution of your graph. It should not take you much time but if you want to start faster, download Sparksee, download the example and run it now!.

If you are not familiar with graph databases or graph concepts, please visit this article.

The first thing you should know about the Sparksee graph you are going to build is that we define it as a labeled and directed attributed mutigraph. It is **labeled** because all the nodes and edges can have a type (a label) to classify them; **directed** because it supports edges with direction from the tail node to the head node, of course we support undirected edges too!; **attributed** because both nodes and edges can have one or more properties; and, finally, it is a **multigraph** because there are no restrictions on the number of edges between two nodes.

Figure 1 is an example of a Sparksee multigraph. Here there are two types of nodes (PEOPLE represented by a star icon and MOVIE shown as a clapperboard icon) which both have an attribute (called respectively NAME and TITLE) with a value for each of them. For instance you can see a *Scarlett Johansson* (NAME) node which will be of type PEOPLE (star icon). Also we can see two types of edges (DIRECTS shown in blue and CAST shown in orange). A directed edge has an arrow pointing to its head node (DIRECTS), while an undirected edge does not have any arrows (CAST). As many attributes as desired could be added to both edges and nodes.

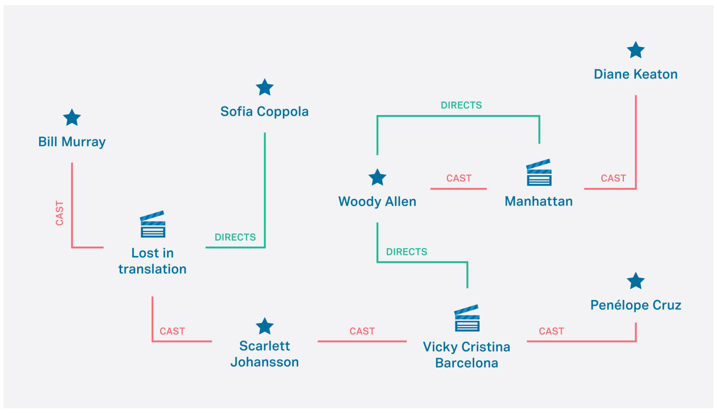


Figure 1: Hello Sparksee

Figure 1 above illustrates the Sparksee labeled and directed attributed multigraph definition. It has labels, as nodes and edges have types, is directed because edge DIRECTS has a certain direction and is a multigraph because node *Woody Allen* with node *Manhattan*, for instance, has two edges.

Following the steps in this guide will help you construct the graph illustrated in **Figure 1**.

It is also relevant to know that Sparksee is an embedded database, so from this point on, you should take into account the fact that installation and deployment of your graph database has to be made considering your programming language preference and framework.

Installation

Let's start with the installation of Sparksee graph database. Sparksee only needs to be downloaded and unpacked. Depending on which language you prefer, a few settings or considerations may need to be taken into account too.

By default Sparksee comes with a free personal evaluation license. Learn more about it in the last section of this chapter.

Download

Sparksee is available for several programming languages, so the first thing you should do is download the right package for your development platform.

All the packages can be downloaded here.

Available platforms:

- **Java:** This is the package for all new Sparksee Java projects.
- **Microsoft .NET:** Sparksee is natively available for .NET developers.
- **C++:** A C++ interface is also available. Sparksee core is C++, so we punch it another level to the API.
- **Python:** Sparksee is available for Python 2.7 developers.
- **Objective-C** An Objective-C api is provided for iOS and MacOS developers. The C++ interface can still be used in Objective-C projects.

Java developers can also get Sparksee through Apache Maven instead of manually downloading the packages. More information in Sparksee maven project.

Unpacking

Once you have downloaded Sparksee and unpacked it, the content of most packages should look like this:

- **doc:** This is the directory that contains Sparksee API **reference** documentation in html or another specific format for each platform.
- **lib:** A directory with Sparksee libraries. It may have subdirectories when different files are required regarding each operating system.
- **ReleaseNotes.txt:** This is a text file with basic information on the latest release changes.
- **LICENSE.txt:** Contains Sparksee licensing information.

According to the programming language particularities, some packages may include additional contents.

Unpack the downloaded Sparksee package and it's ready to be used. No further installation is required, with the exception of Objective-C on MacOS, where an installer is provided to easily copy the framework to the right standard directory.

However, some additional steps can be taken to make Sparksee usage easier for your platform.

Java

All the libraries required to develop a java application with Sparksee are contained in a jar file located at the lib directory (sparkseejava.jar).

If you are not using Maven, you may want to add the path to this file into the CLASSPATH environment variable. However, to avoid any misunderstandings, in this document we will explicitly use the file to compile the examples.

.NET

The .NET package contains two subdirectories in the lib directory (windows32 and windows64) for 32 or 64 bit systems. In each one the main library included is sparkseenet.dll. This is the library that you will need to include in your .NET projects.

All the other available libraries in the package are native dlls that sparkseenet.dll must be able to find at run time. Although it is not required, you may want to copy all these libraries to your system folder.

C++

The C++ lib folder contains native libraries for each available platform (Windows 32/64 bits, Linux 32/64 bits, MacOS 64 bits,...).

In Linux and MacOS, you may want to add the path to the correct subdirectory to your LD_LIBRARY_PATH (linux) or DYLD_LIBRARY_PATH(MacOS) environment variables.

In Windows you can copy the libraries to the system folders or just be sure to always include them in your projects.

The rest of files included in this package, like the ones in the includes directory, will be needed at compilation time. See Chapter 4.

For iOS you have to uncompress the ".dmg" file to get the **Sparksee.framework** directory. This directory contains the include files, the static library and the documentation. More information on how to use it can be found on Chapter 4. Please consider using the Objective-C version of Sparksee instead.

Python

Python lib contains the native libraries for each available platform (Windows 32/64 bits, Linux 32/64 bits and MacOS 64 bits) in addition to the Python module.

Make sure to include both the Python module (called sparksee.py) and the wrapper library named "_sparksee" in an accessible folder for Python, please refer to Python module search path for more information about the search path.

The other available library included in Sparksee's Python distribution is the dynamic native library. This must be located in a specific system directory defined by the following:

- DYLD_LIBRARY_PATH for MacoOSX systems.
- LD_LIBRARY_PATH for Linux systems.
- PATH environment variable for Windows systems.

Objective-C

For the Objective-C, the content of the dmg file is slightly different because, instead of the lib and doc directories, you will find the **Sparksee.framework** directory or its installer regarding if its the iOs or MacOS versions. The documentation will be available directly on the framework subdirectory `Resources/Documentation`.

- **MacOS**

  The dmg file for Mac OS contains an installer (`SparkseeInstaller.pkg`) that you should run. This will extract the framework directory to the right location (`/Library/Frameworks/Sparksee.framework`).

  If you are asked for a target disk, you must choose the main system disk. Your application will depend on the Sparksee dynamic library, so it will need to find it in the right location at runtime.

- **iOS**

  The dmg file for iOS contains the **Sparksee.framework** directory without an installer because there is not the same location restriction as for the MacOS version.

  When you build your application the Sparksee library will be embedded; since it's not a dynamic library framework it doesn't need to be found at any specific location at runtime.

License

Every Sparksee download comes with a limited personal evaluation license included that does not require any further configuration. The personal evaluation license allows the construction of graphs with up to 1M objects, which is more than enough to construct the example explained in this guide.

The mobile versions (Android, iOS, BB) require the explicit use of a license key.

If you have commercial interest or need to deal with larger databases, check Sparksee's website license section for licenses quotation on the latest release of Sparksee.

If you hold a license key, later in this document you will learn how to use your license for a project.

Hello Sparksee

If have reached this chapter you should now have Sparksee correctly installed in your computer and be ready to work in your development framework.

You have previously been introduced to Sparksee APIs and you should have chosen the language with which you feel most familiar. Here we will explain the complete construction of the HelloSparksee application, including the building of the database plus your first queries. For each step in the development we will show an example using all the available Sparksee programming languages; just focus on your chosen language.

Let's say Hello to Sparksee!

Setting up

The first step is the creation of a directory for this project and the new sample application in your development environment. We will come later in the compile and run chapter with certain modifications in the project in order to be able to run Sparksee.

As part of the setup, you could create a text file with the name "**sparksee.cfg**" (or any other name, if you explicitly load it) in the same folder where the executable file will be. This configuration file will establish the default Sparksee settings. This is not a required task, because you can modify these settings directly in the source code using the SparkseeConfig class methods.

These are the most common settings that you may want to set in this file:

- **sparksee.license** : This option is used to set your commercial license key. By default you do not need to provide a license key.
- **sparksee.io.cache.maxsize** : Sets the maximum size for the cache (all pools) in megabytes. By default Sparksee uses almost all of the available memory, just leaving enough memory for other needs. If you are running several memory consuming applications at the same time you must consider adjusting this parameter.
- **sparksee.log.file** : Changes the log file path. The default value is sparksee .log.
- **sparksee.log.level** : The level of detail provided by the log file can be modified with this option. Valid values are: Off, Severe, Warning, Info, Config, Fine and Debug. The default level is Info. For HelloSparksee you will not need to change this level.

A sparksee.cfg file could, for instance, look like this one:

```
sparksee.license=Your-license-key
sparksee.io.cache.maxsize=2048
sparksee.log.file=HelloSparksee.log
```

Where Your−license−key is the alphanumeric key provided by Sparsity Technologies when you acquire a license, the cache assigned for Sparksee is 2GB and the log file name is changed for HelloSparksee.log.

Following this guide we will construct the HelloSparksee example step by step, if you want to go faster, once you are done with the set up you can download the complete application, copy it in your sample application and jump to the compile and run chapter. However we recommend following the guide for a complete understanding of each of the steps of creating a graph database and querying it.

Let's now finish the set up by moving from the directories to your development framework and start coding. Before starting to create your database you should first include references to Sparksee: this is mandatory.

Java

```java
import com.sparsity.sparksee.gdb.*;
```

[C#]

```csharp
using com.sparsity.sparksee.gdb;
```

C++

```cpp
#include "gdb/Sparksee.h"
#include "gdb/Database.h"
#include "gdb/Session.h"
#include "gdb/Graph.h"
#include "gdb/Objects.h"
#include "gdb/ObjectsIterator.h"
```

Python

```python
import sparksee
```

Objective-C

```objc
#import <Sparksee/Sparksee.h>
```

Create a Sparksee database

Now that we have the set up complete and we have started coding, let's create our first database. In this chapter we are going to create a simple database containing information about certain movies, their actors and directors.

The first thing you should do is to create a SparkseeConfig object to establish the Sparksee main settings. That object will be created using the global

SparkseeProperties settings (initially loaded from a sparksee.cfg file). You do not have to change any setting directly here, but its creation is a must before creating the Sparksee object.

If you have a commercial license and you have not yet used it in the configuration file, you may set the license in the SparkseeConfig using the setLicense method.

The newly created SparkseeConfig object will be used to create a Sparksee object. Once you have created this object, you can create your databases.

A **new database** needs a file path (HelloSparksee.gdb) and we can give it a name (HelloSparksee). This database file is where all the persistent information will be stored.

Java

```
SparkseeConfig cfg = new SparkseeConfig();
// The setLicense method is only performed if you have a license key and
// it has not been activated using the configuration file (sparksee.cfg)
cfg.setLicense("Your license key");
Sparksee sparksee = new Sparksee(cfg);
Database db = sparksee.create("HelloSparksee.gdb", "HelloSparksee");
```

[C#]

```
SparkseeConfig cfg = new SparkseeConfig();
// The setLicense method is only performed if you have a license key and
// it has not been activated using the configuration file (sparksee.cfg)
cfg.SetLicense("Your license key");
Sparksee sparksee = new Sparksee(cfg);
Database db = sparksee.Create("HelloSparksee.gdb", "HelloSparksee");
```

C++

```
cfg = sparksee.SparkseeConfig();
// The setLicense method is only performed if you have a license key and
// it has not been activated using the configuration file (sparksee.cfg)
cfg.SetLicense("Your license key");
Sparksee *sparksee = new Sparksee(cfg);
Database *db = sparksee->Create(L"HelloSparksee.gdb", L"HelloSparksee");
```

Python

```
cfg.SparkseeConfig()
# The setLicense method is only performed if you have a license key and
# it has not been activated using the configuration file (sparksee.cfg)cfg.
     set_license("Your license key");
sparks = sparksee.Sparksee(cfg)
db = sparks.create("Hellosparksee.gdb", "HelloSparksee")
```

Objective-C

```
STSSparkseeConfig *cfg = [[STSSparkseeConfig alloc] init];
// The setLicense method is only performed if you have a license key and
// it has not been activated using the configuration file (sparksee.cfg)
[cfg setLicense: @"Your license key"];
STSSparksee *sparksee = [[STSSparksee alloc] initWithConfig: cfg];
```

```
// If you are not using Objective-C Automatic Reference Counting, you
// may want to release the configuration here.
//[cfg release];
STSDatabase *db = [sparksee create: @"HelloSparksee.gdb" alias: @"HelloSparksee"];
```

Sessions and transactions

All the manipulation of a database must be enclosed within a session. A Session should be initiated from a Database instance. This is where you can get a Graph instance which represents the persistent graph (the graph database).

The Graph instance is needed to manipulate the Database as a graph.

Also, **temporary data** is associated to the Session, thus when a Session is closed, all the temporary data associated to that Session is removed too. Objects or Values instances or even session attributes are an example of temporary data.

You must take into account the fact that a Session is exclusive for a thread, we do not encourage sharing it among threads as it will definitely raise unexpected errors.

A Session manages the transactions, allowing the execution of a set of graph operations as a single execution unit. A transaction encloses all the graph operations between the Session Begin and Commit or Rollback methods, or just a single operation in autocommit mode (where no begin/commit/rollback methods are used).

Initially, a transaction starts as a read transaction and only when there is a call to a method which updates the persistent graph database, it automatically becomes a write transaction. To become a write transaction it must wait until all other read transactions have finished. But a transaction can also be started as a write transaction using the BeginUpdate method instead of Begin.

Since HelloSparksee is a simple example we are going to use autocommit because we don't need more complex transactions.

For more information about Sparksee transactions please take a look at the Session class in the reference documentation of your chosen Sparksee API.

Java

```
Session sess = db.newSession();
Graph g = sess.getGraph();
```

[C#]

```
Session sess = db.NewSession();
Graph g = sess.GetGraph();
```

C++

```
Session *sess = db->NewSession();
Graph *g = sess->GetGraph();
```

Python

```
sess = db.new_session()
graph = sess.get_graph()
```

Objective-C

```
STSSession *sess = [db createSession];
STSGraph *g = [sess getGraph];
```

Set the schema

The HelloSparksee sample application is a very simple movie database where we store information about movies, their actors and directors. All of them are nodes in the graph whereas the relationships between them are edges.

In a graph database we can represent movies, actors and directors using two node types:

- **MOVIE**: This type represents a movie.
- **PEOPLE**: This type represents a person involved in a movie, be they cast or crew member.

We can enrich these by adding more information such as the movie title, year of production, or in the case of the people only their names. This information are attributes of the recently created node types:

For each **MOVIE** we may need the following attributes:

- **ID**: Unique identification for a movie.
- **TITLE**: Original title.
- **YEAR**: Year of production.

For **PEOPLE** we need these attributes:

- **ID**: Unique identification for a person.
- **NAME**: Complete name.

Note that the **ID** attribute is not required, but it is always useful to have a unique attribute value in order to identify each node, as the other attributes (name and title) can't be considered unique.

The **relationships** are represented by edges between the former two types of nodes. We need two types of edges:

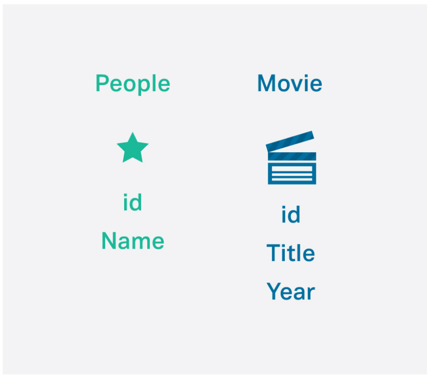


Figure 2: HelloSparksee node schema

- **CAST**: Represents the relationship between **PEOPLE** and **MOVIE** meaning that the person is part of the cast of that movie.
- **DIRECTS**: Represents the relationship between **PEOPLE** and **MOVIE** meaning that the person is the director of that particular movie.

The first one (**CAST**) is going to be an **undirected edge and it will have an attribute** (**CHARACTER**) to store the name of the role performed in that movie. In fact it is not a restricted edge, so we could use it between other types of nodes too. For example, later on we could add a new node type "ANIMAL" and use the same "CAST" edge type between "MOVIE" and "ANIMAL" nodes.

The other (**DIRECTS**) is going to be a **restricted directed edge without attributes**. It is restricted because it can only be used between **PEOPLE** and **MOVIE** nodes.

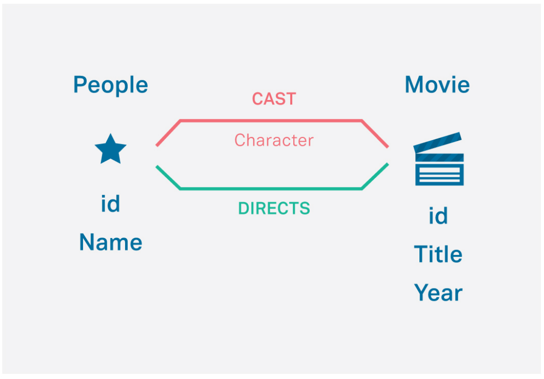


Figure 3: HelloSparksee schema

Create node types

Let's now construct the node types **MOVIE** and **PEOPLE** and then add the desired attributes to each type definition, which we have defined in our schema.

There are three types of attributes:

- **Basic**: This type of attributes cannot be used for query operations (just get and set attribute values).
- **Unique**: Attributes that work as a primary key, which means that two objects cannot have the same value for an attribute (but NULL). They can be used for query operations.
- **Indexed**: Attributes that can be used for query operations.

Also you have to choose a datatype for the attributes. Available datatypes for attributes are: Boolean, Integer, Long, Double, Timestamp, String, Text and OID.

The **ID** attributes are going to be numeric (Long) unique values. The **TITLE** and **NAME** attributes are going to be String values and both are going to be Indexed. For the **YEAR** attribute we are going to use an Integer type and this is also going to be Indexed.

Java

```
// Add a node type for the movies, with a unique identifier and two indexed
     attributes
int movieType = g.newNodeType("MOVIE");
int movieIdType = g.newAttribute(movieType, "ID", DataType.Long, AttributeKind.
     Unique);
int movieTitleType = g.newAttribute(movieType, "TITLE", DataType.String,
     AttributeKind.Indexed);
int movieYearType = g.newAttribute(movieType, "YEAR", DataType.Integer,
     AttributeKind.Indexed);

// Add a node type for the people, with a unique identifier and an indexed
     attribute
int peopleType = g.newNodeType("PEOPLE");
int peopleIdType = g.newAttribute(peopleType, "ID", DataType.Long, AttributeKind.
     Unique);
int peopleNameType = g.newAttribute(peopleType, "NAME", DataType.String,
     AttributeKind.Indexed);
```

[C#]

```
// Add a node type for the movies, with a unique identifier and two indexed
     attributes
int movieType = g.NewNodeType("MOVIE");
int movieIdType = g.NewAttribute(movieType, "ID", DataType.Long, AttributeKind.
     Unique);
int movieTitleType = g.NewAttribute(movieType, "TITLE", DataType.String,
     AttributeKind.Indexed);
int movieYearType = g.NewAttribute(movieType, "YEAR", DataType.Integer,
     AttributeKind.Indexed);

// Add a node type for the people, with a unique identifier and an indexed
     attribute
int peopleType = g.NewNodeType("PEOPLE");
int peopleIdType = g.NewAttribute(peopleType, "ID", DataType.Long, AttributeKind.
     Unique);
```

```
int peopleNameType = g.NewAttribute(peopleType, "NAME", DataType.String,
     AttributeKind.Indexed);
```

C++

```
// Add a node type for the movies, with a unique identifier and two indexed
     attributes
type_t movieType = g->NewNodeType(L"MOVIE");
attr_t movieIdType = g->NewAttribute(movieType, L"ID", Long, Unique);
attr_t movieTitleType = g->NewAttribute(movieType, L"TITLE", String, Indexed);
attr_t movieYearType = g->NewAttribute(movieType, L"YEAR", Integer, Indexed);

// Add a node type for the people, with a unique identifier and an indexed
     attribute
type_t peopleType = g->NewNodeType(L"PEOPLE");
attr_t peopleIdType = g->NewAttribute(peopleType, L"ID", Long, Unique);
attr_t peopleNameType = g->NewAttribute(peopleType, L"NAME", String, Indexed);
```

Python

```
# Add a node type for the movies, with a unique identifier and two indexed
     attributes
movie_type = graph.new_node_type(u"MOVIE")
movie_id_type = graph.new_attribute(movie_type, u"ID", sparksee.DataType.LONG,
     sparksee.AttributeKind.UNIQUE)
movie_title_type = graph.new_attribute(movie_type, u"TITLE", sparksee.DataType.
     STRING, sparksee.AttributeKind.INDEXED)
movie_year_type = graph.new_attribute(movie_type, "uYEAR", sparksee.DataType.
     INTEGER, sparksee.AttributeKind.INDEXED)

# Add a node type for the people, with a unique identifier and an indexed
     attribute
people_type = graph.new_nodetype(u"PEOPLE")
people_id_type = graph.new_attribute(people_type, u"ID", sparksee.DataType.LONG,
     sparksee.AttributeKind.UNIQUE)
people_name_type = graph.new_attribute(people_type, u"NAME", sparksee.DataType.
     STRING, sparksee.AttributeKind.INDEXED)
```

Objective-C

```
// Add a node type for the movies, with a unique identifier and two indexed
     attributes
int movieType = [g createNodeType: @"MOVIE"];
int movieIdType = [g createAttribute: movieType name: @"ID" dt: STSLong kind:
     STSUnique];
int movieTitleType = [g createAttribute: movieType name: @"TITLE" dt: STSString
     kind: STSIndexed];
int movieYearType = [g createAttribute: movieType name: @"YEAR" dt: STSInteger
     kind: STSIndexed];

// Add a node type for the people, with a unique identifier and an indexed
     attribute
int peopleType = [g createNodeType: @"PEOPLE"];
int peopleIdType = [g createAttribute: peopleType name: @"ID" dt: STSLong kind:
     STSUnique];
int peopleNameType = [g createAttribute: peopleType name: @"NAME" dt: STSString
     kind: STSIndexed];
```

Create edge types

Now that we have created our nodes let's add the relationships we have previously explained in our schema.

We have defined edge types **CAST** and **DIRECTS**. We stated during the schema explanation that **CAST** will be undirected while **DIRECTS** will be directed. Let's explain a little bit more about this classification of our edge types.

Directed edges have a node which is the *tail* (the source of the edge) and a node which is the *head* (the destination of the edge). In case of **undirected** edges, each node at the extreme of the edge plays two roles at the same time, head and tail. Whereas undirected edges express navigation in any side, directed edges show the direction of the edge but they can also be navigated through that natural direction or the opposite one.

Also, edges can be classified as restricted or unrestricted. **Restricted** edges define which must be the type of the *tail* and *head* nodes, thus edges will only be allowed between those specified type of nodes. In case of **unrestricted** edges, there is no restriction, and edges are allowed between nodes belonging to any type. It is important to note that restricted edges must be directed edges.

In addition, in our schema we have decided that CAST will have an attribute called **CHARACTER**. The **CHARACTER** attribute is going to be a String and we are going to set it as Basic. See the previous section for more information about the attribute types.

Java

```
// Add an undirected edge type with an attribute for the cast of a movie
int castType = g.newEdgeType("CAST", false, false);
int castCharacterType = g.newAttribute(castType, "CHARACTER", DataType.String,
     AttributeKind.Basic);

// Add a directed edge type restricted to go from people to movie for the director
     of a movie
int directsType = g.newRestrictedEdgeType("DIRECTS", peopleType, movieType, false)
     ;
```

[C#]

```
// Add an undirected edge type with an attribute for the cast of a movie
int castType = g.NewEdgeType("CAST", false, false);
int castCharacterType = g.NewAttribute(castType, "CHARACTER", DataType.String,
     AttributeKind.Basic);

// Add a directed edge type restricted to go from people to movie for the director
     of a movie
int directsType = g.NewRestrictedEdgeType("DIRECTS", peopleType, movieType, false)
     ;
```

C++

```
// Add an undirected edge type with an attribute for the cast of a movie
type_t castType = g->NewEdgeType(L"CAST", false, false);
attr_t castCharacterType = g->NewAttribute(castType, L"CHARACTER", String, Basic);
```

```
// Add a directed edge type restricted to go from people to movie for the director
      of a movie
type_t directsType = g->NewRestrictedEdgeType(L"DIRECTS", peopleType, movieType,
     false);
```

Python

```
# Add an undirected edge type with an attribute for the cast of a movie
cast_type = graph.new_edge_type(u"CAST", False, False)
cast_character_type = graph.new_attribute(cast_type, u"CHARACTER", sparksee.
     DataType.STRING, sparksee.AttributeKind.BASIC)

# Add a directed edge type restricted to go from people to movie for the director
     of a movie
directs_type = graph.new_restricted_edge_type(u"DIRECTS", people_type, movie_type,
      False)
```

Objective-C

```
// Add an undirected edge type with an attribute for the cast of a movie
int castType = [g createEdgeType: @"CAST" directed: FALSE neighbors: FALSE];
int castCharacterType = [g createAttribute: castType name: @"CHARACTER" dt:
     STSString kind: STSBasic];

// Add a directed edge type restricted to go from people to movie for the director
      of a movie
int directsType = [g createRestrictedEdgeType: @"DIRECTS" tail: peopleType head:
     movieType neighbors: FALSE];
```

Adding data

Once the schema has been created the next step is to add data. It is interesting to note that the schema can be modified later with no great impact to the database, which is an important characteristic of the Sparksee graph database.

In the HelloSparksee example we are adding enough information to be able to perform some simple queries afterwards.

Although is out of the scope of this guide, it is worth noting that you can use Sparksee loaders if you are dealing with large data sets.

Add nodes

We are going to add information about these movies:

- Lost in Translation
- Vicky Cristina Barcelona
- Manhattan

And some of the people that worked in these movies:

- Scarlett Johansson
- Bill Murray
- Sofia Coppola
- Woody Allen
- Penélope Cruz
- Diane Keaton

In a previous section we have seen that all this information is stored as nodes with attributes. So, we have to create 3 **MOVIE** nodes for the films, which will each have the attributes **ID**, **Title** and **Year**. Also we have to create 6 **PEOPLE** nodes for the cast and crew, where each will have the attributes **ID** and **Name**. Notice that we use the class Value to set the attributes values, and the same Value object is reused for all the attributes.

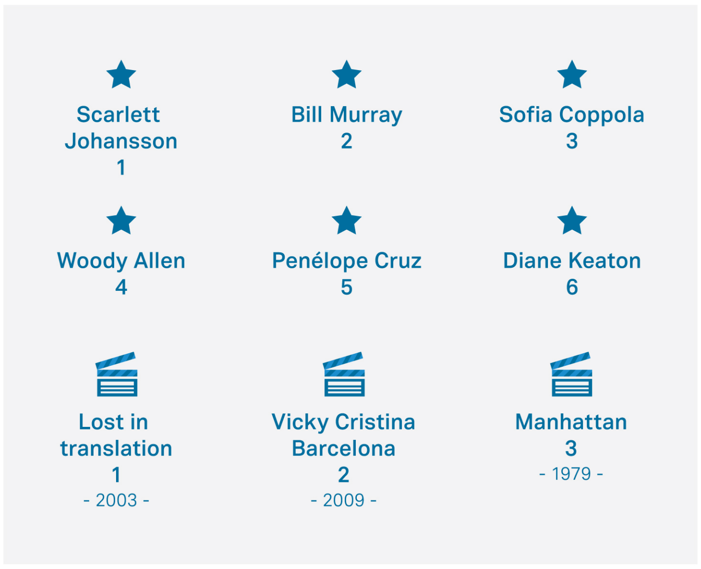


Figure 4: Adding nodes

Java

```
// Add some MOVIE nodes
Value value = new Value();

long mLostInTranslation = g.newNode(movieType);
g.setAttribute(mLostInTranslation, movieIdType, value.setLong(1));
g.setAttribute(mLostInTranslation, movieTitleType, value.setString("Lost in
    Translation"));
g.setAttribute(mLostInTranslation, movieYearType, value.setInteger(2003));

long mVickyCB = g.newNode(movieType);
g.setAttribute(mVickyCB, movieIdType, value.setLong(2));
g.setAttribute(mVickyCB, movieTitleType, value.setString("Vicky Cristina Barcelona
    "));
g.setAttribute(mVickyCB, movieYearType, value.setInteger(2008));

long mManhattan = g.newNode(movieType);
g.setAttribute(mManhattan, movieIdType, value.setLong(3));
g.setAttribute(mManhattan, movieTitleType, value.setString("Manhattan"));
```

```
g.setAttribute(mManhattan, movieYearType, value.setInteger(1979));


// Add some PEOPLE nodes
long pScarlett = g.newNode(peopleType);
g.setAttribute(pScarlett, peopleIdType, value.setLong(1));
g.setAttribute(pScarlett, peopleNameType, value.setString("Scarlett Johansson"));

long pBill = g.newNode(peopleType);
g.setAttribute(pBill, peopleIdType, value.setLong(2));
g.setAttribute(pBill, peopleNameType, value.setString("Bill Murray"));

long pSofia = g.newNode(peopleType);
g.setAttribute(pSofia, peopleIdType, value.setLong(3));
g.setAttribute(pSofia, peopleNameType, value.setString("Sofia Coppola"));

long pWoody = g.newNode(peopleType);
g.setAttribute(pWoody, peopleIdType, value.setLong(4));
g.setAttribute(pWoody, peopleNameType, value.setString("Woody Allen"));

long pPenelope = g.newNode(peopleType);
g.setAttribute(pPenelope, peopleIdType, value.setLong(5));
g.setAttribute(pPenelope, peopleNameType, value.setString("Penélope Cruz"));

long pDiane = g.newNode(peopleType);
g.setAttribute(pDiane, peopleIdType, value.setLong(6));
g.setAttribute(pDiane, peopleNameType, value.setString("Diane Keaton"));
```

[C#]

```
// Add some MOVIE nodes
Value value = new Value();

long mLostInTranslation = g.NewNode(movieType);
g.SetAttribute(mLostInTranslation, movieIdType, value.SetLong(1));
g.SetAttribute(mLostInTranslation, movieTitleType, value.SetString("Lost in
     Translation"));
g.SetAttribute(mLostInTranslation, movieYearType, value.SetInteger(2003));

long mVickyCB = g.NewNode(movieType);
g.SetAttribute(mVickyCB, movieIdType, value.SetLong(2));
g.SetAttribute(mVickyCB, movieTitleType, value.SetString("Vicky Cristina Barcelona
     "));
g.SetAttribute(mVickyCB, movieYearType, value.SetInteger(2008));

long mManhattan = g.NewNode(movieType);
g.SetAttribute(mManhattan, movieIdType, value.SetLong(3));
g.SetAttribute(mManhattan, movieTitleType, value.SetString("Manhattan"));
g.SetAttribute(mManhattan, movieYearType, value.SetInteger(1979));


// Add some PEOPLE nodes
long pScarlett = g.NewNode(peopleType);
g.SetAttribute(pScarlett, peopleIdType, value.SetLong(1));
g.SetAttribute(pScarlett, peopleNameType, value.SetString("Scarlett Johansson"));

long pBill = g.NewNode(peopleType);
g.SetAttribute(pBill, peopleIdType, value.SetLong(2));
g.SetAttribute(pBill, peopleNameType, value.SetString("Bill Murray"));

long pSofia = g.NewNode(peopleType);
g.SetAttribute(pSofia, peopleIdType, value.SetLong(3));
g.SetAttribute(pSofia, peopleNameType, value.SetString("Sofia Coppola"));

long pWoody = g.NewNode(peopleType);
g.SetAttribute(pWoody, peopleIdType, value.SetLong(4));
g.SetAttribute(pWoody, peopleNameType, value.SetString("Woody Allen"));

long pPenelope = g.NewNode(peopleType);
g.SetAttribute(pPenelope, peopleIdType, value.SetLong(5));
g.SetAttribute(pPenelope, peopleNameType, value.SetString("Penélope Cruz"));
```

```
long pDiane = g.NewNode(peopleType);
g.SetAttribute(pDiane, peopleIdType, value.SetLong(6));
g.SetAttribute(pDiane, peopleNameType, value.SetString("Diane Keaton"));
```

C++

```
// Add some MOVIE nodes
Value *value = new Value();

oid_t mLostInTranslation = g->NewNode(movieType);
g->SetAttribute(mLostInTranslation, movieIdType, value->SetLong(1));
g->SetAttribute(mLostInTranslation, movieTitleType, value->SetString(L"Lost in
     Translation"));
g->SetAttribute(mLostInTranslation, movieYearType, value->SetInteger(2003));

oid_t mVickyCB = g->NewNode(movieType);
g->SetAttribute(mVickyCB, movieIdType, value->SetLong(2));
g->SetAttribute(mVickyCB, movieTitleType, value->SetString(L"Vicky Cristina
     Barcelona"));
g->SetAttribute(mVickyCB, movieYearType, value->SetInteger(2008));

oid_t mManhattan = g->NewNode(movieType);
g->SetAttribute(mManhattan, movieIdType, value->SetLong(3));
g->SetAttribute(mManhattan, movieTitleType, value->SetString(L"Manhattan"));
g->SetAttribute(mManhattan, movieYearType, value->SetInteger(1979));


// Add some PEOPLE nodes
oid_t pScarlett = g->NewNode(peopleType);
g->SetAttribute(pScarlett, peopleIdType, value->SetLong(1));
g->SetAttribute(pScarlett, peopleNameType, value->SetString(L"Scarlett Johansson")
     );

oid_t pBill = g->NewNode(peopleType);
g->SetAttribute(pBill, peopleIdType, value->SetLong(2));
g->SetAttribute(pBill, peopleNameType, value->SetString(L"Bill Murray"));

oid_t pSofia = g->NewNode(peopleType);
g->SetAttribute(pSofia, peopleIdType, value->SetLong(3));
g->SetAttribute(pSofia, peopleNameType, value->SetString(L"Sofia Coppola"));

oid_t pWoody = g->NewNode(peopleType);
g->SetAttribute(pWoody, peopleIdType, value->SetLong(4));
g->SetAttribute(pWoody, peopleNameType, value->SetString(L"Woody Allen"));

oid_t pPenelope = g->NewNode(peopleType);
g->SetAttribute(pPenelope, peopleIdType, value->SetLong(5));
g->SetAttribute(pPenelope, peopleNameType, value->SetString(L"Penélope Cruz"));

oid_t pDiane = g->NewNode(peopleType);
g->SetAttribute(pDiane, peopleIdType, value->SetLong(6));
g->SetAttribute(pDiane, peopleNameType, value->SetString(L"Diane Keaton"));
```

Python

```
# Add some MOVIE nodes
value = sparksee.Value()

lost_in_translation_movie = graph.new_node(movie_type)
graph.set_attribute(lost_in_translation_movie, movie_id_type, value.set_long(1))
graph.set_attribute(lost_in_translation_movie, movie_title_type, value.set_string(
     u"Lost in Translation"))
graph.set_attribute(lost_in_translation_movie, movie_year_type, value.set_integer
     (2003))

vicky_cb_movie = graph.new_node(movie_type)
```

```
graph.set_attribute(vicky_cb_movie, movie_id_type, value.set_long(2))
graph.set_attribute(vicky_cb_movie, movie_title_type, value.set_string(u"Vicky
     Cristina Barcelona"))
graph.set_attribute(vicky_cb_movie, movie_year_type, value.set_integer(2008))

manhattan_movie = graph.new_node(movie_type)
graph.set_attribute(manhattan_movie, movie_id_type, value.set_long(3))
graph.set_attribute(manhattan_movie, movie_title_type, value.set_string(u"
     manhattan_movie"))
graph.set_attribute(manhattan_movie, movie_year_type, value.set_integer(1979))


# Add some PEOPLE nodes
scarlett_people = graph.new_node(people_type)
graph.set_attribute(scarlett_people, people_id_type, value.set_long(1))
graph.set_attribute(scarlett_people, people_name_type, value.set_string(u"Scarlett
      Johansson"))

bill_people = graph.new_node(people_type)
graph.set_attribute(bill_people, people_id_type, value.set_long(2))
graph.set_attribute(bill_people, people_name_type, value.set_string(u"Bill Murray"
     ))

sofia_people = graph.new_node(people_type)
graph.set_attribute(sofia_people, people_id_type, value.set_long(3))
graph.set_attribute(sofia_people, people_name_type, value.set_string(u"Sofia
     Coppola"))

woody_people = graph.new_node(people_type)
graph.set_attribute(woody_people, people_id_type, value.set_long(4))
graph.set_attribute(woody_people, people_name_type, value.set_string(u"Woody Allen
     "))

penelope_people = graph.new_node(people_type)
graph.set_attribute(penelope_people, people_id_type, value.set_long(5))
graph.set_attribute(penelope_people, people_name_type, value.set_string(u"Penélope
      Cruz"))

diane_people = graph.new_node(people_type)
graph.set_attribute(diane_people, people_id_type, value.set_long(6))
graph.set_attribute(diane_people, people_name_type, value.set_string(u"Diane
     Keaton"))
```

Objective-C

```
// Add some MOVIE nodes
STSValue *value = [[STSValue alloc] init];

long long mLostInTranslation = [g createNode: movieType];
[g setAttribute: mLostInTranslation attr: movieIdType value: [value setLong: 1]];
[g setAttribute: mLostInTranslation attr: movieTitleType value: [value setString:
     @"Lost in Translation"]];
[g setAttribute: mLostInTranslation attr: movieYearType value: [value setInteger:
     2003]];

long long mVickyCB = [g createNode: movieType];
[g setAttribute: mVickyCB attr: movieIdType value: [value setLong: 2]];
[g setAttribute: mVickyCB attr: movieTitleType value: [value setString: @"Vicky
     Cristina Barcelona"]];
[g setAttribute: mVickyCB attr: movieYearType value: [value setInteger: 2008]];

long long mManhattan = [g createNode: movieType];
[g setAttribute: mManhattan attr: movieIdType value: [value setLong: 3]];
[g setAttribute: mManhattan attr: movieTitleType value: [value setString: @"
     Manhattan"]];
[g setAttribute: mManhattan attr: movieYearType value: [value setInteger: 1979]];


// Add some PEOPLE nodes
long long pScarlett = [g createNode: peopleType];
[g setAttribute: pScarlett attr: peopleIdType value: [value setLong: 1]];
```

```
[g setAttribute: pScarlett attr: peopleNameType value: [value setString: @"
     Scarlett Johansson"]];

long long pBill = [g createNode: peopleType];
[g setAttribute: pBill attr: peopleIdType value: [value setLong: 2]];
[g setAttribute: pBill attr: peopleNameType value: [value setString: @"Bill Murray
     "]];

long long pSofia = [g createNode: peopleType];
[g setAttribute: pSofia attr: peopleIdType value: [value setLong: 3]];
[g setAttribute: pSofia attr: peopleNameType value: [value setString: @"Sofia
     Coppola"]];

long long pWoody = [g createNode: peopleType];
[g setAttribute: pWoody attr: peopleIdType value: [value setLong: 4]];
[g setAttribute: pWoody attr: peopleNameType value: [value setString: @"Woody
     Allen"]];

long long pPenelope = [g createNode: peopleType];
[g setAttribute: pPenelope attr: peopleIdType value: [value setLong: 5]];
[g setAttribute: pPenelope attr: peopleNameType value: [value setString: @"
     Penélope Cruz"]];

long long pDiane = [g createNode: peopleType];
[g setAttribute: pDiane attr: peopleIdType value: [value setLong: 6]];
[g setAttribute: pDiane attr: peopleNameType value: [value setString: @"Diane
     Keaton"]];
```

Add edges

Now that we have all the nodes in the database we can start adding the relationships between them. As previously explained in the Set the schema section we are going to build two types of relationships depending on whether the person attached to a movie is part of the cast (edge **CAST**) or its director (edge **DIRECTS**).

We are going to add an edge of type **CAST** between each node of type **PEOPLE** and each node of type **MOVIE** when the person worked as an actor in that movie. Then, we will set the edge attribute **CHARACTER** as the name of the character played by that actor in the movie.

After this, we will create an edge of type **DIRECTS** between a node of type **PEOPLE** and a node of type **MOVIE** for the director of each movie.

Java

```
// Add some CAST edges
// Remember that we are reusing the Value class instance to set the attributes
long anEdge;
anEdge = g.newEdge(castType, mLostInTranslation, pScarlett);
g.setAttribute(anEdge, castCharacterType, value.setString("Charlotte"));

anEdge = g.newEdge(castType, mLostInTranslation, pBill);
g.setAttribute(anEdge, castCharacterType, value.setString("Bob Harris"));

anEdge = g.newEdge(castType, mVickyCB, pScarlett);
g.setAttribute(anEdge, castCharacterType, value.setString("Cristina"));

anEdge = g.newEdge(castType, mVickyCB, pPenelope);
g.setAttribute(anEdge, castCharacterType, value.setString("Maria Elena"));

anEdge = g.newEdge(castType, mManhattan, pDiane);
g.setAttribute(anEdge, castCharacterType, value.setString("Mary"));
```

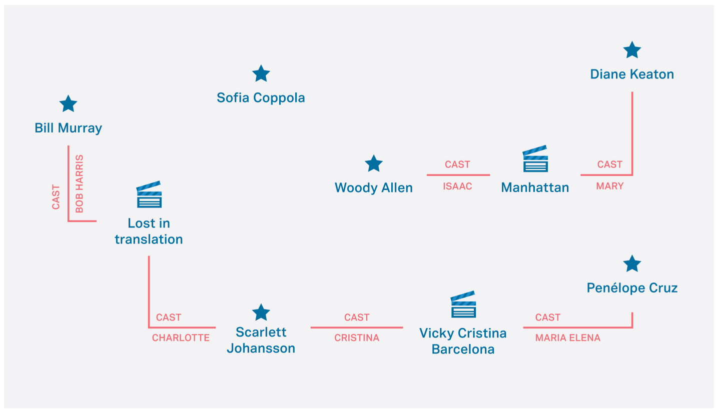


Figure 5: Adding CAST edges (Notice that we have omitted the attributes of the nodes)

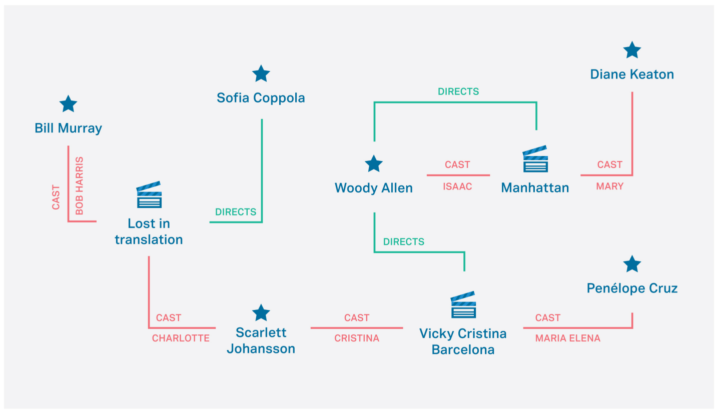


Figure 6: Adding DIRECTS edges (Notice that we have omitted the attributes of the nodes)

```
anEdge = g.newEdge(castType, mManhattan, pWoody);
g.setAttribute(anEdge, castCharacterType, value.setString("Isaac"));



// Add some DIRECTS edges
anEdge = g.newEdge(directsType, pSofia, mLostInTranslation);

anEdge = g.newEdge(directsType, pWoody, mVickyCB);

anEdge = g.newEdge(directsType, pWoody, mManhattan);
```

[C#]

```
// Add some CAST edges
// Remember that we are reusing the Value class instance to set the attributes
long anEdge;
anEdge = g.NewEdge(castType, mLostInTranslation, pScarlett);
g.SetAttribute(anEdge, castCharacterType, value.SetString("Charlotte"));

anEdge = g.NewEdge(castType, mLostInTranslation, pBill);
g.SetAttribute(anEdge, castCharacterType, value.SetString("Bob Harris"));

anEdge = g.NewEdge(castType, mVickyCB, pScarlett);
g.SetAttribute(anEdge, castCharacterType, value.SetString("Cristina"));

anEdge = g.NewEdge(castType, mVickyCB, pPenelope);
g.SetAttribute(anEdge, castCharacterType, value.SetString("Maria Elena"));

anEdge = g.NewEdge(castType, mManhattan, pDiane);
g.SetAttribute(anEdge, castCharacterType, value.SetString("Mary"));

anEdge = g.NewEdge(castType, mManhattan, pWoody);
g.SetAttribute(anEdge, castCharacterType, value.SetString("Isaac"));



// Add some DIRECTS edges
anEdge = g.NewEdge(directsType, pSofia, mLostInTranslation);

anEdge = g.NewEdge(directsType, pWoody, mVickyCB);

anEdge = g.NewEdge(directsType, pWoody, mManhattan);
```

C++

```
// Add some CAST edges
// Remember that we are reusing the Value class instance to set the attributes
oid_t anEdge;
anEdge = g->NewEdge(castType, mLostInTranslation, pScarlett);
g->SetAttribute(anEdge, castCharacterType, value->SetString(L"Charlotte"));

anEdge = g->NewEdge(castType, mLostInTranslation, pBill);
g->SetAttribute(anEdge, castCharacterType, value->SetString(L"Bob Harris"));

anEdge = g->NewEdge(castType, mVickyCB, pScarlett);
g->SetAttribute(anEdge, castCharacterType, value->SetString(L"Cristina"));

anEdge = g->NewEdge(castType, mVickyCB, pPenelope);
g->SetAttribute(anEdge, castCharacterType, value->SetString(L"Maria Elena"));

anEdge = g->NewEdge(castType, mManhattan, pDiane);
g->SetAttribute(anEdge, castCharacterType, value->SetString(L"Mary"));

anEdge = g->NewEdge(castType, mManhattan, pWoody);
g->SetAttribute(anEdge, castCharacterType, value->SetString(L"Isaac"));
```

```
// Add some DIRECTS edges
anEdge = g->NewEdge(directsType, pSofia, mLostInTranslation);

anEdge = g->NewEdge(directsType, pWoody, mVickyCB);

anEdge = g->NewEdge(directsType, pWoody, mManhattan);
```

Python

```
# Add some CAST edges
# Remember that we are reusing the Value class instance to set the attributes
an_edge = graph.new_edge(cast_type, lost_in_translation_movie, scarlett_people)
graph.set_attribute(an_edge, cast_character_type, value.set_string(u"Charlotte"))

an_edge = graph.new_edge(cast_type, lost_in_translation_movie, bill_people)
graph.set_attribute(an_edge, cast_character_type, value.set_string(u"Bob Harris"))

an_edge = graph.new_edge(cast_type, vicky_cb_movie, scarlett_people)
graph.set_attribute(an_edge, cast_character_type, value.set_string(u"Cristina"))

an_edge = graph.new_edge(cast_type, vicky_cb_movie, penelope_people)
graph.set_attribute(an_edge, cast_character_type, value.set_string(u"Maria Elena")
     )

an_edge = graph.new_edge(cast_type, manhattan_movie, diane_people)
graph.set_attribute(an_edge, cast_character_type, value.set_string(u"Mary"))

an_edge = graph.new_edge(cast_type, manhattan_movie, woody_people)
graph.set_attribute(an_edge, cast_character_type, value.set_string(u"Isaac"))



# Add some DIRECTS edges
an_edge = graph.new_edge(directs_type, sofia_people, lost_in_translation_movie)

an_edge = graph.new_edge(directs_type, woody_people, vicky_cb_movie)

an_edge = graph.new_edge(directs_type, woody_people, manhattan_movie)
```

Objective-C

```
// Add some CAST edges
long long anEdge;
anEdge = [g createEdge: castType tail: mLostInTranslation head: pScarlett];
[g setAttribute: anEdge attr: castCharacterType value: [value setString: @"
     Charlotte"]];

anEdge = [g createEdge: castType tail: mLostInTranslation head: pBill];
[g setAttribute: anEdge attr: castCharacterType value: [value setString: @"Bob
     Harris"]];

anEdge = [g createEdge: castType tail: mVickyCB head: pScarlett];
[g setAttribute: anEdge attr: castCharacterType value: [value setString: @"
     Cristina"]];

anEdge = [g createEdge: castType tail: mVickyCB head: pPenelope];
[g setAttribute: anEdge attr: castCharacterType value: [value setString: @"Maria
     Elena"]];

anEdge = [g createEdge: castType tail: mManhattan head: pDiane];
[g setAttribute: anEdge attr: castCharacterType value: [value setString: @"Mary"
     ]];

anEdge = [g createEdge: castType tail: mManhattan head: pWoody];
```

```
[g setAttribute: anEdge attr: castCharacterType value: [value setString: @"Isaac"
    ]];


// Add some DIRECTS edges
anEdge = [g createEdge: directsType tail: pSofia head: mLostInTranslation];

anEdge = [g createEdge: directsType tail: pWoody head: mVickyCB];

anEdge = [g createEdge: directsType tail: pWoody head: mManhattan];
```

First queries

If you have successfully completed all the previous steps in this chapter you have now created your first graph, congratulations! The graph should look exactly like Figure 7.

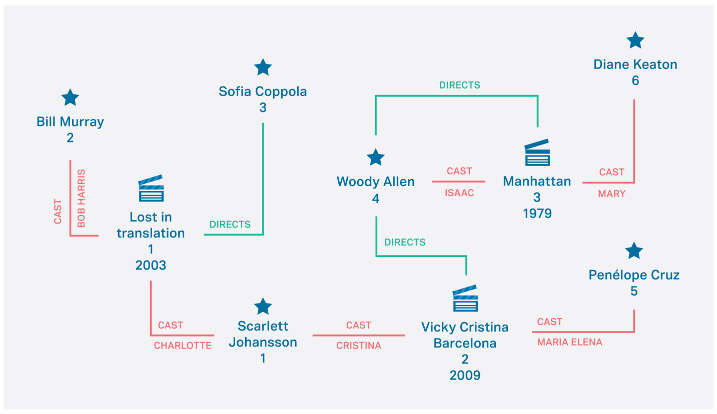


Figure 7: HelloSparksee complete graph

Although you have accomplished the main objective of this guide we encourage you to follow the final steps: querying the graph and the necessary procedure of closing your graph database.

Let's propose a simple example that illustrates how to query a Sparksee graph database. For instance you may be interested in finding out who acted in movies directed by Woody Allen and also acted in movies directed by Sofia Coppola. We are able to establish the connection between these two excellent directors.

We could start by searching for the "Woody Allen" node using the FindObject method. However, as we have just created the graph we know that we already have that information in a variable called pWoody because we kept it when adding the node.

As we already have the "Woody Allen" node, our first query is to obtain the collection of movies directed by him. For each of his movies, there is an edge of type **DIRECTS** that starts from his node to a node of type **MOVIE**. To retrieve this information we will use the Neighbors method against the "Woody Allen" node through the edge **DIRECTS**. As we are only interested in the head (hence the movie) of this directed edge we truncate this retrieval to "Outgoing" only.

Java

```
// Get the movies directed by Woody Allen
Objects directedByWoody = g.neighbors(pWoody, directsType, EdgesDirection.Outgoing
    );
```

[C#]

```
// Get the movies directed by Woody Allen
Objects directedByWoody = g.Neighbors(pWoody, directsType, EdgesDirection.Outgoing
    );
```

C++

```
// Get the movies directed by Woody Allen
Objects *directedByWoody = g->Neighbors(pWoody, directsType, Outgoing);
```

Python

```
# Get the movies directed by Woody Allen
directed_by_woody = graph.neighbors(woody_people, directs_type, sparksee.
    EdgesDirection.OUTGOING)
```

Objective-C

```
// Get the movies directed by Woody Allen
STSObjects *directedByWoody = [g neighbors: pWoody etype: directsType dir:
    STSOutgoing];
```

The result of the query is an Objects class instance. It stores a collection of Sparksee object identifiers as a set; in this way we do not obtain duplicated elements.

Now that we have found all the movies directed by Woody Allen, we can use them to find the people that acted in them. To do so, we can use the Neighbors operation again, but from the collection of movies of Woody Allen and using the **CAST** edge type. In this case, we will use Any edge direction because it is not a directed edge.

All the Objects instances should be closed when no longer needed, therefore we can close the directedbyWoody collection just after retrieving its cast.

Java

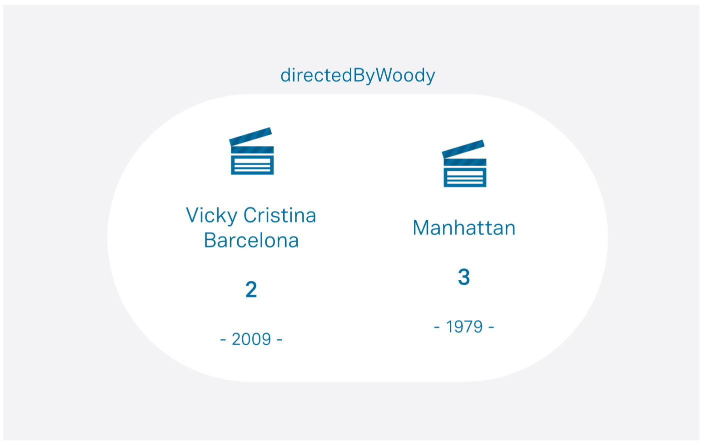


Figure 8: Movies directed by Woody Allen

```
// Get the cast of the movies directed by Woody Allen
Objects castDirectedByWoody = g.neighbors(directedByWoody, castType,
     EdgesDirection.Any);

// We don't need the directedByWoody collection anymore, so we should close it
directedByWoody.close();
```

[C#]

```
// Get the cast of the movies directed by Woody Allen
Objects castDirectedByWoody = g.Neighbors(directedByWoody, castType,
     EdgesDirection.Any);

// We don't need the directedByWoody collection anymore, so we should close it
directedByWoody.Close();
```

C++

```
// Get the cast of the movies directed by Woody Allen
Objects *castDirectedByWoody = g->Neighbors(directedByWoody, castType, Any);

// We don't need the directedByWoody collection anymore, so we should delete it
delete directedByWoody;
```

Python

```
# Get the cast of the movies directed by Woody Allen
cast_directed_by_woody = graph.neighbors(directed_by_woody, cast_type, sparksee.
     EdgesDirection.ANY)

# We don't need the directed_by_woody collection anymore, so we should close it
directed_by_woody.close()
```

Objective-C

```
// Get the cast of the movies directed by Woody Allen
STSObjects *castDirectedByWoody = [g neighborsWithObjects: directedByWoody etype:
     castType dir: STSAny];

// We don't need the directedByWoody collection anymore, so we should close it
[directedByWoody close];
```

We now have all the people that acted in movies directed by Woody Allen (Figure 9) but we only wanted to know who also acted in movies directed by Sofia Coppola.

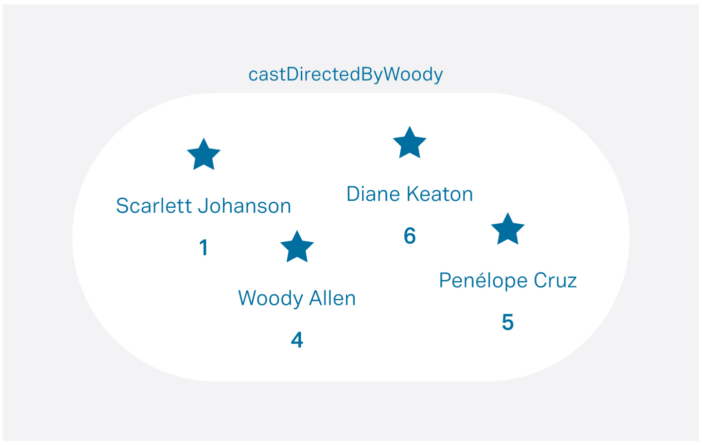


Figure 9: Cast in movies by Woody Allen

To match the cast in movies directed by Woody Allen with the cast in movies directed by Sofia Coppola we have to repeat the same queries previously performed for Woody Allen: the first is the retrieval of movies of Sofia Coppola followed by the retrieval of the cast of each of her movies. The result is shown in Figure 10.

Java

```
// Get the movies directed by Sofia Coppola
Objects directedBySofia = g.neighbors(pSofia, directsType, EdgesDirection.Outgoing
     );

// Get the cast of the movies directed by Sofia Coppola
Objects castDirectedBySofia = g.neighbors(directedBySofia, castType,
     EdgesDirection.Any);

// We don't need the directedBySofia collection anymore, so we should close it
directedBySofia.close();
```

[C#]

```
// Get the movies directed by Sofia Coppola
Objects directedBySofia = g.Neighbors(pSofia, directsType, EdgesDirection.Outgoing
     );
```

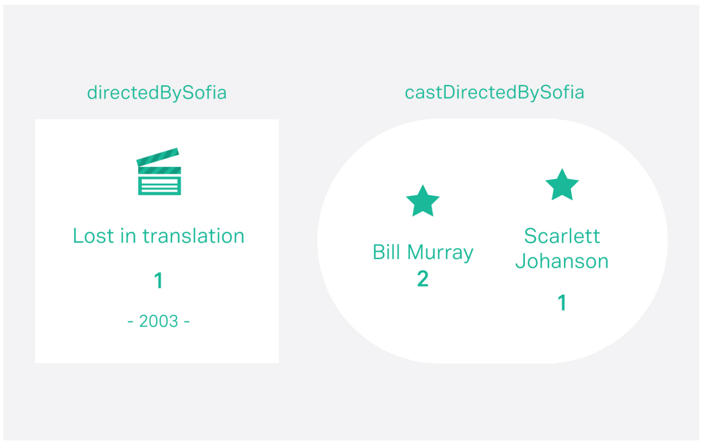


Figure 10: Movies and cast for Sofia Coppola

```
// Get the cast of the movies directed by Sofia Coppola
Objects castDirectedBySofia = g.Neighbors(directedBySofia, castType,
     EdgesDirection.Any);

// We don't need the directedBySofia collection anymore, so we should close it
directedBySofia.Close();
```

C++

```cpp
// Get the movies directed by Sofia Coppola
Objects *directedBySofia = g->Neighbors(pSofia, directsType, Outgoing);

// Get the cast of the movies directed by Sofia Coppola
Objects *castDirectedBySofia = g->Neighbors(directedBySofia, castType, Any);

// We don't need the directedBySofia collection anymore, so we should delete it
delete directedBySofia;
```

Python

```python
# Get the movies directed by Sofia Coppola
directed_by_sofia = graph.neighbors(sofia_people, directs_type, sparksee.
     EdgesDirection.OUTGOING)

# Get the cast of the movies directed by Sofia Coppola
cast_directed_by_sofia = graph.neighbors(directed_by_sofia, cast_type, sparksee.
     EdgesDirection.ANY)

# We don't need the directed_by_sofia collection anymore, so we should close it
directed_by_sofia.close()
```

Objective-C

```objc
// Get the movies directed by Sofia Coppola
STSObjects *directedBySofia = [g neighbors: pSofia etype: directsType dir:
     STSOutgoing];
```

```
// Get the cast of the movies directed by Sofia Coppola
STSObjects *castDirectedBySofia = [g neighborsWithObjects: directedBySofia etype:
     castType dir: STSAny];

// We don't need the directedBySofia collection anymore, so we should close it
[directedBySofia close];
```

In the collections called castDirectedByWoody and castDirectedBySofia we now have all the cast in movies directed by each director respectively. But the objective of the query was to find out who acted in movies directed by both of them. To achieve this we only need to calculate the intersection of these two sets of **PEOPLE** nodes.

Java

```
// We want to know the people that acted in movies directed by Woody AND in movies
      directed by Sofia.
Objects castFromBoth = Objects.combineIntersection(castDirectedByWoody,
     castDirectedBySofia);

// We don't need the other collections anymore
castDirectedByWoody.close();
castDirectedBySofia.close();
```

[C#]

```
// We want to know the people that acted in movies directed by Woody AND in movies
      directed by Sofia.
Objects castFromBoth = Objects.CombineIntersection(castDirectedByWoody,
     castDirectedBySofia);

// We don't need the other collections anymore
castDirectedByWoody.Close();
castDirectedBySofia.Close();
```

C++

```
// We want to know the people that acted in movies directed by Woody AND in movies
      directed by Sofia.
Objects *castFromBoth = Objects::CombineIntersection(castDirectedByWoody,
     castDirectedBySofia);

// We don't need the other collections anymore
delete castDirectedByWoody;
delete castDirectedBySofia;
```

Python

```
# We want to know the people that acted in movies directed by Woody AND in movies
     directed by Sofia.
cast_from_both = sparksee.Objects.combine_intersection(cast_directed_by_woody,
     cast_directed_by_sofia)

# We don't need the other collections anymore
cast_directed_by_woody.close()
cast_directed_by_sofia.close()
```

Objective-C

```
// We want to know the people that acted in movies directed by Woody AND in movies
      directed by Sofia.
STSObjects *castFromBoth = [STSObjects combineIntersection: castDirectedByWoody
     objs2: castDirectedBySofia];

// We don't need the other collections anymore
[castDirectedByWoody close];
[castDirectedBySofia close];
```

Remember that you should close the Objects when you are not going to use them anymore.

We think we may have the result that we were looking for. But how do we check the information from that Objects collection? You must use ObjectsIterator that will traverse all the elements inside the set. With this iterator we can get all the node identifiers in the result (**PEOPLE** node identifiers), one by one. Then, for each one, we can get their attributes. We are only interested in the name of the actor.

Here you have it! Scarlett Johansson is the link between both directors.

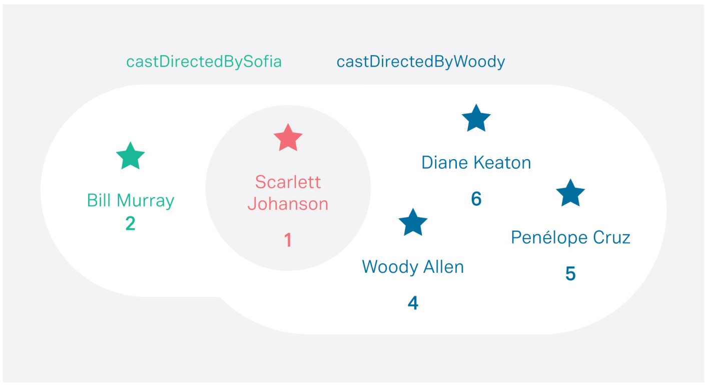


Figure 11: Link between Woody Allen and Sofia Coppola

Java

```
// Say hello to the people found
ObjectsIterator it = castFromBoth.iterator();
while (it.hasNext())
{
    long peopleOid = it.next();
    g.getAttribute(peopleOid, peopleNameType, value);
    System.out.println("Hello " + value.getString());
}
// The ObjectsIterator must be closed
it.close();

// The Objects must be closed
castFromBoth.close();
```

[C#]

```
// Say hello to the people found
ObjectsIterator it = castFromBoth.Iterator();
while (it.HasNext())
{
    long peopleOid = it.Next();
    g.GetAttribute(peopleOid, peopleNameType, value);
    System.Console.WriteLine("Hello " + value.GetString());
}
// The ObjectsIterator must be closed
it.Close();

// The Objects must be closed
castFromBoth.Close();
```

C++

```
// Say hello to the people found
ObjectsIterator *it = castFromBoth->Iterator();
while (it->HasNext())
{
    oid_t peopleOid = it->Next();
    g->GetAttribute(peopleOid, peopleNameType, *value);
    std::wcout << L"Hello " << value->GetString() << std::endl;
}
// The ObjectsIterator must be deleted
delete it;

// The Objects must be deleted
delete castFromBoth;
```

Python

```
# Say hello to the people found
for people_oid in cast_from_both:
    graph.get_attribute(people_oid, people_name_type, value)
    print "Hello ", value.get_string()

# The Objects must be closed
    cast_from_both.close()
```

Objective-C

```
// Say hello to the people found
STSObjectsIterator *it = [castFromBoth iterator];
while ([it hasNext])
{
    long long peopleOid = [it next];
    [g getAttributeInValue: peopleOid attr: peopleNameType value: value];
    NSLog(@"Hello %@\n", [value getString]);
}
// The ObjectsIterator must be closed
[it close];

// The Objects must be closed
[castFromBoth close];
```

Again we have reused the Value class instance to get the value of the **NAME** attribute.

Note that the ObjectsIterator should also be closed before closing the Objects collection.

Closing the database

This guide is almost finished. You have now performed all the tasks necessary to create a graph with its schema, add data and query this data afterwards. There is only one inevitable and very important final step: the proper closing of the database.

You must take into account the fact that all the collections and iterators should be closed first. You can close them now, or an even better programming practice is to close them as soon as they are no longer needed.

To close Sparksee, once collections and iterators have been closed, you must first close the Session (or sessions) which will free all the temporary information stored in it, then close the Database to proceed to the closure of the Sparksee instance.

Java

```
sess.close();
db.close();
sparksee.close();
```

[C#]

```
sess.Close();
db.Close();
sparksee.Close();
```

C++

```
delete sess;
delete db;
delete sparksee;
```

Python

```
sess.close()
db.close()
sparks.close()
```

Objective-C

```
[sess close];
[db close];
[sparksee close];
// If you are not using Objective-C Automatic Reference Counting, you
// may want to release the sparksee here, when it's closed.
//[sparksee release];
```

Compile and run

To compile and run your Sparksee application you must take into account your development framework.

Java

In the Installation chapter we have seen how to download and unpack the java package to get the **sparkseejava.jar** file. You need to include that jar in you development environment project.

If you don't use an IDE, you just need to add the **sparkseejava.jar** file to the classpath. So, you can compile and run the HelloSparksee application with these simple commands:

```
$ javac -cp sparkseejava.jar HelloSparksee.java
$ java -cp sparkseejava.jar;. HelloSparksee
```

With Maven

If you use Apache Maven, then it is even easier. Sparksee is in the maven central repository, adding the dependency to the correct Sparksee version into your "pom.xml" file should be enough:

```
<dependency>
  <groupId>com.sparsity</groupId>
  <artifactId>sparkseejava</artifactId>
  <version>5.0.0</version>
</dependency>
```

With Android

The procedure to use Sparkseejava in Android is the same for Eclipse and Android Studio, but We have separated some steps to better explain the procedure in each environment.

- Copy all the content of the **lib/** directory to the **libs/** directory of your android project.
- If you want different ".apk" files for the different target architectures instead of a single application file that supports all platforms or you only want to support certain architectures, you just need to exclude from the previous step the subdirectories of the platforms that you don't want.
- Refresh the project explorer if you can't see the files that you just copied.

Using Eclipse

- Right click on the **libs/sparkseejava.jar** file and select Build Path > Add to Build Path.
- Set in your "AndroidManifest.xml" a minimum sdk version greater or equal to 9.

Android Studio

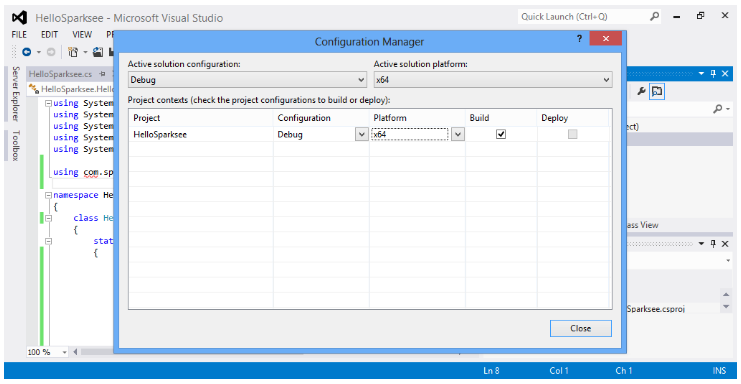
- Right click on the **libs/sparkseejava.jar** file and select Add as ...library.
- Add this exact text "compile files'(libs/sparkseejava.'jar)" to the **build.gradle** file.
- Set a minimum sdk version >= 9 in the **build.gradle** file.

.NET

.NET developers have the following two different options to build a .NET application.

MS Visual Studio users

If you are a MS Visual Studio IDE developer, you need to add the reference to the Sparksee .NET library (**sparkseenet.dll**) in your project and set build platform to the appropiate specific platfrom (x64 or x86). Please, check that in the "Configuration Manager" from the "BUILD" menu of Visual Studio, the "Platform" selected for the build is NOT "Any CPU". If it's "Any CPU", you must select "New" to set a "x86" or "x64" build target.


Configuration Manager

Active solution configuration: Debug

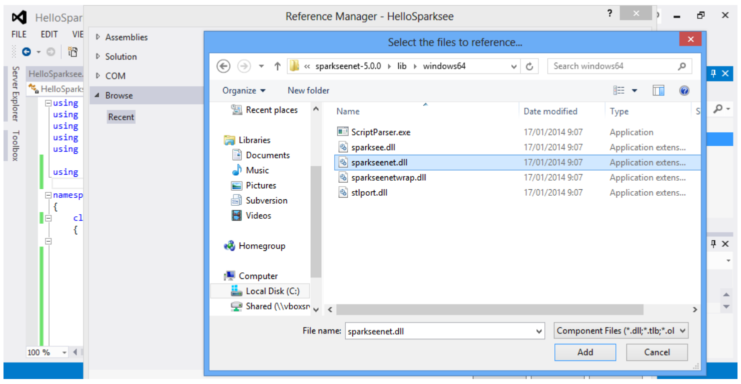
Active solution platform: x64

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build	Deploy
HelloSparksee	Debug	x64	✓	

Figure 12: .Net compilation - setting the platform

Since all the other libraries included in the package are native libraries that will be loaded at runtime, they must be available too. And the MS Visual C runtime must be installed on all the computers running your application.



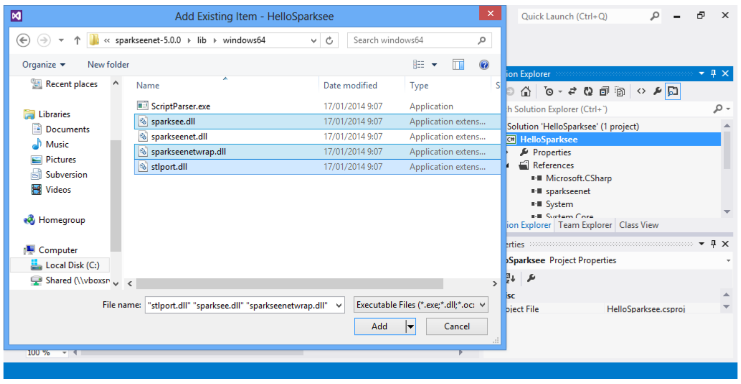
Name	Date modified	Type
ScriptParser.exe	17/01/2014 9:07	Application
sparksee.dll	17/01/2014 9:07	Application extens...
sparkseenet.dll	17/01/2014 9:07	Application extens...
sparkseenetwrap.dll	17/01/2014 9:07	Application extens...
stlport.dll	17/01/2014 9:07	Application extens...

File name: sparkseenet.dll
Component Files (*.dll;*.tlb;*.ol

Figure 13: .Net compilation - adding reference

The best option is to copy all the other ".dll" files into the same directory where your application executable file will be.

Using the development environment, this can be done using the option Add existing Item, choosing to see Executable files and selecting the other three native libraries (sparksee.dll, sparkseenetwarp.dll and stlport.dll).



Name	Date modified	Type
ScriptParser.exe	17/01/2014 9:07	Application
sparksee.dll	17/01/2014 9:07	Application extens...
sparkseenet.dll	17/01/2014 9:07	Application extens...
sparkseenetwrap.dll	17/01/2014 9:07	Application extens...
stlport.dll	17/01/2014 9:07	Application extens...

File name: "stlport.dll" "sparksee.dll" "sparkseenetwrap.dll"
Executable Files (*.exe;*.dll;*.oc:

Figure 14: .Net compilation - adding existing item

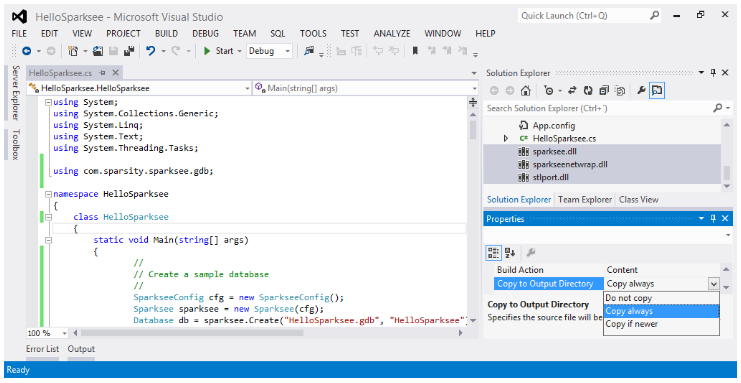
Next you must select the three libraries in the Solution Explorer window and set the property Copy to Output as Copy Always for all three native libraries.

Instead of copying the native libraries to the application target directory, an alternative would be to put all the native ".dll" files into your Windows system folder (System32 or SysWOW64 depending on your Windows version).

Now you are ready to build and run the application like any Visual Studio project.

Command-line users

If you just want to quickly test the HelloSparksee sample application, you can use the command line. First setup your compiler environment with the vsvars32



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using com.sparsity.sparksee.gdb;

namespace HelloSparksee
{
    class HelloSparksee
    {
        static void Main(string[] args)
        {
                //
                // Create a sample database
                //
                SparkseeConfig cfg = new SparkseeConfig();
                Sparksee sparksee = new Sparksee(cfg);
                Database db = sparksee.Create("HelloSparksee.gdb", "HelloSparksee")
```

Build Action: Content
Copy to Output Directory: Copy always

Do not copy
Copy always
Copy if newer

Copy to Output Directory
Specifies the source file will be

Figure 15: .Net compilation - copy to output

.bat file (or vcvarsall.bat) if you are using a 32 bit MS Visual Studio.

```
> call "C:\Program Files\Microsoft Visual Studio 11.0\Common7\Tools\vsvars32.bat"
```

or with the vcvarsall.bat file with the appropiate argument if you are using a 64 bit MS Visual Studio.

```
> call "C:\Program Files (x86)\Microsoft Visual Studio 11.0\VC\vcvarsall.bat"
    amd64
```

Then compile and run the application (assuming all the libraries have already been copied to the same directory):

```
> csc /out:HelloSparksee.exe /r:sparkseenet.dll HelloSparksee.cs
> HelloSparksee.exe
```

C++

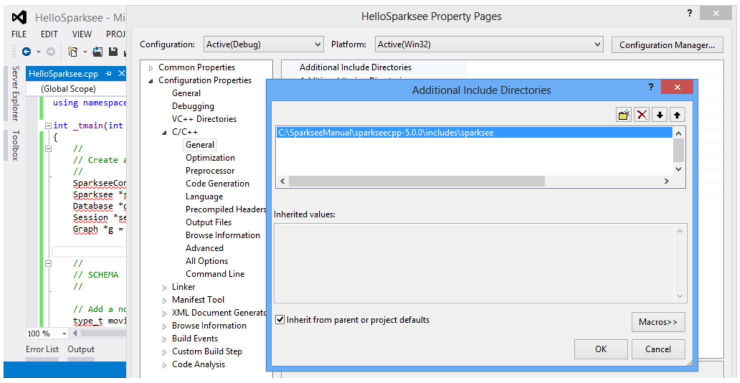
The Sparksee C++ interface contains include files and dynamic libraries in order to compile and run an application using Sparksee. The general procedure is to first add the include directories to your project, then link with the supplied libraries corresponding to your operating system and finally copy them to any place where they can be loaded at runtime (common places are the same folder as the target executable file or your system libraries folder).

Let's have a look at a more detailed description of this procedure in the most common environments.

Remember that the package should already be unpacked in a known directory (see chapter 2).

Windows

If your development environment is Microsoft Visual Studio, your first step should be to add to the Additional include directories, C++ general property of your project the **sparksee** subdirectory of the includes-folder from the Sparksee package.


HelloSparksee Property Pages

Configuration: Active(Debug) Platform: Active(Win32)

Configuration Properties > C/C++ > General > Additional Include Directories

Additional Include Directories

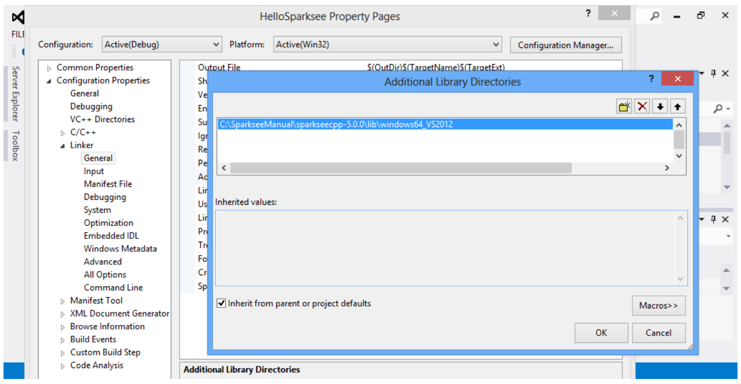
C:\SparkseeManual\sparkseecpp-5.0.0\includes\sparksee

Inherited values:

☑ Inherit from parent or project defaults

Figure 16: C++ compilation - include directories

This must also be done with the library directory, so the Additonal library directories linker general property must be edited to add the correct subdirectory of the Sparksee **lib** folder for your operating system.


HelloSparksee Property Pages

Configuration: Active(Debug) Platform: Active(Win32)

Configuration Properties > Linker > General > Additional Library Directories

Output File $(OutDir)$(TargetName)$(TargetExt)

Additional Library Directories

C:\SparkseeManual\sparkseecpp-5.0.0\lib\windows64_VS2012

Inherited values:

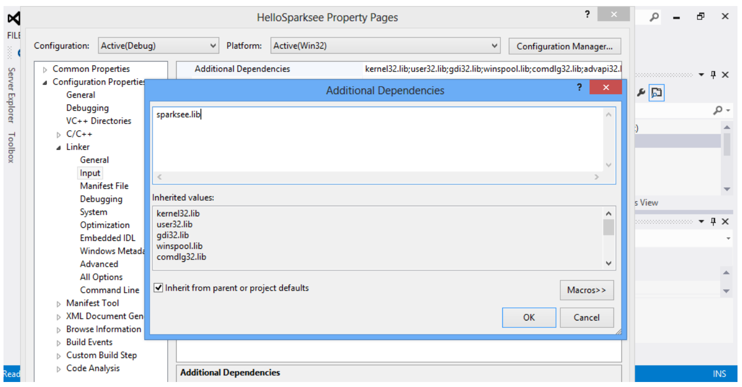
☑ Inherit from parent or project defaults

Figure 17: C++ compilation - add library directories

After this, you should add the **sparksee** “.lib” library to the Additional Dependencies linker input property.

Finally make sure that the dll files can be found at run time. An easy way to do this is to add a post-process in your project to copy the dll files to the same output folder where your application executable will be built.

An alternative would be to simply put all the native “.dll” files into your Windows system folder (System32 or SysWOW64 depending on your Windows version).


HelloSparksee Property Pages

Configuration: Active(Debug) Platform: Active(Win32)

Additional Dependencies: kernel32.lib;user32.lib;gdi32.lib;winspool.lib;comdlg32.lib;advapi32.l

Additional Dependencies

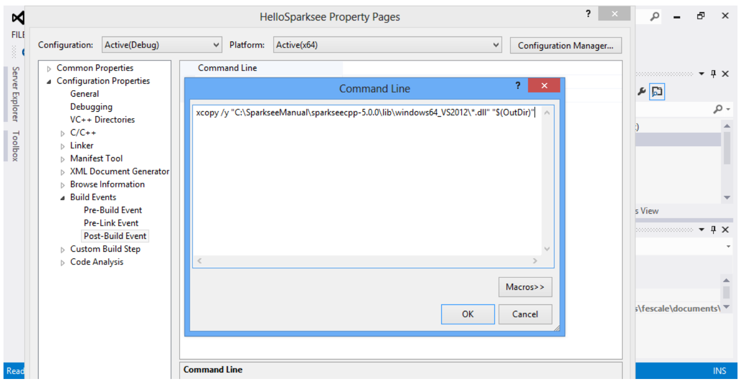
```
sparksee.lib
```

Inherited values:

```
kernel32.lib
user32.lib
gdi32.lib
winspool.lib
comdlg32.lib
```

☑ Inherit from parent or project defaults

Figure 18: C++ compilation - add dependencies


HelloSparksee Property Pages

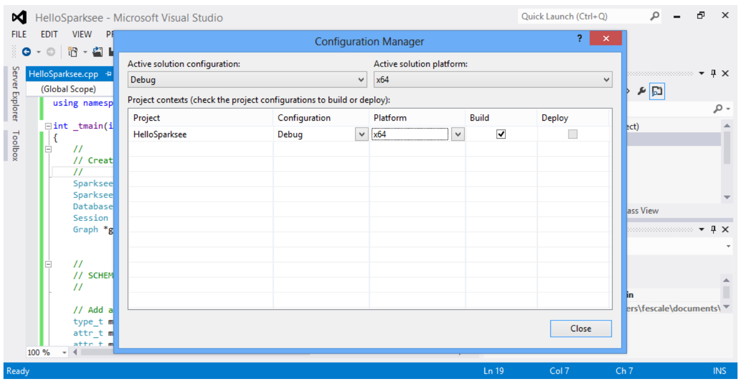
Configuration: Active(Debug) Platform: Active(x64)

Command Line

```
xcopy /y "C:\SparkseeManual\sparkseecpp-5.0.0\lib\windows64_VS2012\*.dll" "$(OutDir)"
```

Figure 19: C++ compilation - post build event

It's important to check that the build platform selected matches the libraries that you are using.


Configuration Manager

Active solution configuration: Debug

Active solution platform: x64

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build	Deploy
HelloSparksee	Debug	x64	✓	☐

Figure 20: C++ compilation - platform

Now you are ready to build and run the application like any Visual Studio project.

Finally, if you just want to quickly test the HelloSparksee sample application, you can use the command line. First setup your compiler environment with the vsvars32.bat file (or vcvarsall.bat) if you are using a 32-bit MS Visual Studio.

```
> call "C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common7\Tools\vsvars32
    .bat"
```

or with the vcvarsall.bat file with the appropiate argument if you are using a 64-bit MS Visual Studio.

```
> call "C:\Program Files (x86)\Microsoft Visual Studio 11.0\VC\vcvarsall.bat"
    amd64
```

Then compile and run the application (example on a 32-bit Windows):

```
> cl /I"path_to_the_unpacked_sparksee\includes\sparksee" /D "WIN32" /D "_UNICODE"
    /D "UNICODE" /EHsc /MD /c HelloSparksee.cpp

> link /OUT:"HelloSparksee.exe" HelloSparksee.obj /LIBPATH:"
    path_to_the_unpacked_sparksee\lib\windows32" "sparksee.lib"

> xcopy /y "path_to_the_unpacked_sparksee\lib\windows32\*.dll" .

> HelloSparksee.exe
```

Linux/MacOS

We are not going to focus on any specific integrated development environment for Linux or Mac OS because it is beyond the scope of the guide. Instead we will give a list of procedures which you can adapt for the specifics of your development environment.

- In the **includes** directory, there is the subdirectory **sparksee** that must be added as include search directory in your project.
- The **lib** directory contains a subdirectory for each operating system available. You should add the correct directory for your computer as a link search directory in your project.
- To link your application, the **sparksee** and your **pthread** libraries must be used in this order.
- Finally you may need to add the directory where the libraries can be found to the **LD_LIBRARY_PATH**, or **DYLD_LIBRARY_PATH** in MacOS, environment variable to be sure that they will be found at run-time.

Finally, if you just want to quickly test the HelloSparksee sample application, you can use the command line.

```
$ g++ -I/path_to_the_unpacked_sparksee/includes/sparksee -o HelloSparksee.o -c
    HelloSparksee.cpp

$ g++ HelloSparksee.o -o HelloSparksee -L../lib/linux64 -lsparksee -lpthread

$ export LD_LIBRARY_PATH=/path_to_the_unpacked_sparksee/lib/linux64/

$ ./HelloSparksee
```

Android

The Android usage is not very different from the Linux usage.

- You also have to add the **includes/sparksee** directory to the includes search path.
- The sparksee dynamic library and the provided stlport_shared library must be included in the applicacion. There are 4 versions of each library in subdirectories from the **lib/** directory. You must use the appropiate libraries for the processor target of your project.
- The **stlport_shared** is the NDK library but must be included in the application instead of just being linked.
- The **z** and **dl** libraries from the Android NDK must be linked too.

iOS

Once you have extracted the **Sparksee.framework** directory from the distribution ".dmg" file, the basic steps to use Sparksee in your Xcode application project are the following:

- Add the Sparksee include files to the search path in your application project: The path to Sparksee.framework/Resources/sparksee/ must be added as non-recursive to the User Header Search Paths option on the build settings of your xcode application project. This is required because Sparksee include files use a hierarchy of directories not usual in an xcode framework. Therefore, they can't be included in the regular Headers directory of the framework.
- Add the **Sparksee.framework** to the Link Binary With Libraries build phase of your application project. You can just drag it there.
- Choose the appropiate library: libstdc++ (GNU C++ standard library) or libc++ (LLVM C++ standard library with C++11 support) in the C++ Standard Library option in the build settings of the compiler. The option choosen must match the downloaded version of the Sparkseecpp for iOS.
- Remember that all the source files using C++ should have the extension ".mm" instead of ".m".
- Take into account that, after all these changes, a Clean of your Project may be needed.
- Setting an explicit memory limit to the Sparksee cache (using the SparkseeConfig class) is highly recommended. For more information about SPARKSEE cache and the SPARKSEEConfig class check the full User Manual and the reference guides.

Python

Sparksee Python doesn't need to be compiled, which is a difference regarding the other Sparksee APIs.

In order to run a Python script using Sparksee make sure that the Sparksee module script (called sparksee.py) and the wrapper library named "_sparksee" are in the Python module search path.

The other available library included in Sparksee's Python distribution is the dynamic native library. This must be located in a specific system directory defined by the following:

- DYLD_LIBRARY_PATH for MacoOSX systems.
- LD_LIBRARY_PATH for Linux systems.
- PATH environment variable for Windows systems.

Once the installation is completed you can run the script normally. The following example assumes that the python2.7 executable is available in your path, otherwise you should write the full path to your Python executable.

```
$ python ./HelloSparksee.py
```

Objective-C

Although both the Mac OS and iOS versions can be included in you application project in the same way and could be used equally from the source code, the installation of the framework and deployment of your application is slightly different. Let's take a look at both versions in the following sections.

MacOS

In the Installation chapter we have seen how to download, unpack the dmg file and install the Sparksee package to get the **Sparksee.framework** installed in the /Library/Frameworks/ directory. The Mac OS version of the framework is a standard framework containing a dynamic library, so it's installed in a fixed standard location.

To use the Sparksee Objective-C framework in your application, you have to add the **Sparksee.framework** from /Library/Frameworks/Sparksee.framework to the Link Binary With Libraries build phase of your application project.

Then you can import the header in your source code.

```
#import <Sparksee/Sparksee.h>
```

Take into account that your application will depend on the Sparksee dynamic library, therefore the Sparksee framework must be installed on the target computers either manually or redistributed with your own installer.

Alternatively, you could include the framework as a private framework inside your application, but then a modification of the framework library location on the @executable_path instead of the standard /Library/Frameworks/ would be required

iOS

In the Installation chapter we have seen how to download and unpack the dmg file. For iOS, you don't need to install the framework because it would be already there. You can copy it directly to wherever you want because there is not any established standard dynamic library framework.

To use the Sparksee Objective-C framework in your application, you have to add the **Sparksee.framework** to the Link Binary With Libraries build phase of your application project. You can just drag it there.

Then you can import the header in your source code.

```
#import <Sparksee/Sparksee.h>
```

Your iOS application will contain the Sparksee library embedded (it's a static library), so your application deployment should be exactly the same as any other iOS application.

Download examples

Here you can download the *HelloSparksee* sources including all the examples which have been explained in this starting guide.

If you have followed all the steps up to this point you should have created a graph database which looks exactly like **Figure 7**.

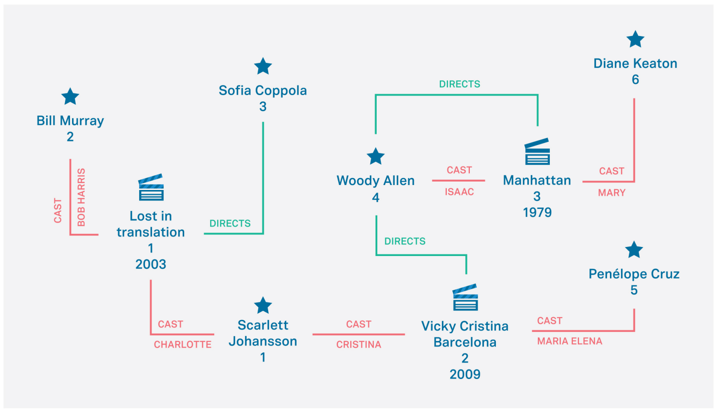


Figure 7: HelloSparksee complete graph

You can also directly download *HelloSparksee* sources which, once you run them, will construct the same graph.

HelloSparksee first creates a Sparksee graph database (see Chapter 3-section 2), then creates the schema (see Chapter3-section 4), adds data creating nodes and edges and their attributes (see Chapter3-section 5) and finally queries this data (see Chapter3-section 6).

Queries included in the example retrieve neighbors from some nodes. For instance *all the movies directed by Woody Allen*, which will be the neighbors of Woody Allen through the DIRECTS edge. Or other more complex examples include retrieving *all the people who acted both in movies directed by Woody Allen and in movies directed by Sofia Coppola.*

Choose your *HelloSparksee* download language:

- Java: Hello Sparksee in Java
- .Net: Hello Sparksee for .Net
- C++: Hello Sparksee in C++
- Python2.7: Hello Sparksee in Python2.7
- Objective-C: Hello Sparksee in Objective-C

Support

This is the final section of the Sparksee starting guide. We have guided you through the entire process of creating your first graph with Sparksee. Moreover, you have added data to your graph and finally queried it.

We encourage you to learn more about the advanced features of Sparksee, practice with the rest of available queries & functionalities, and build your own application.

While developing do not forget to consult the Sparksee **reference manuals**. Reference manuals are included in the doc directory of the Sparksee package. You can also directly consult the information in the documentation section of our website choosing your preferred programming language.

Also in the documentation section of the Sparsity Technologies website you can find **tutorials** that will give you further details about Sparksee.

One of our main support channels is the Sparksee Technical Area **forum** where code examples are displayed and questions resolved. Here you will find Sparksee advanced programmers who will answer to your questions. Do not hesitate to share any doubts you may have, however small.

Finally do not forget to **follow us** on twitter, facebook and linkedin. You can share your doubts and thoughts there too!