

# Tutorial

up-to-date version of the Tutorial is available via Subversion repository: <http://svn.apache.org/repos/asf/thrift/trunk/tutorial/>

```
#!/usr/local/bin/thrift --gen cpp --gen java --gen py --php --gen rb --gen perl --erl --xsd -r
#
# Thrift Tutorial
# Mark Slee (mcslee@facebook.com)
#
# This file aims to teach you how to use Thrift, in a .thrift file. Neato. The
# first thing to notice is that .thrift files support standard shell comments.
# This lets you make your thrift file executable and include your Thrift build
# step on the top line. And you can place comments like this anywhere you like.
#
# Before running this file, you will need to have installed the thrift compiler
# into /usr/local/bin.

/**
 * The first thing to know about are types. The available types in Thrift are:
 *
 * bool          Boolean, one byte
 * byte          Signed byte
 * i16           Signed 16-bit integer
 * i32           Signed 32-bit integer
 * i64           Signed 64-bit integer
 * double        64-bit floating point value
 * string        String
 * map<t1,t2>    Map from one type to another
 * list<t1>     Ordered list of one type
 * set<t1>      Set of unique elements of one type
 *
 * Did you also notice that Thrift supports C style comments?
 */

// Just in case you were wondering... yes. We support simple C comments too.

/**
 * Thrift files can reference other Thrift files to include common struct
 * and service definitions. These are found using the current path, or by
 * searching relative to any paths specified with the -I compiler flag.
 *
 * Included objects are accessed using the name of the .thrift file as a
 * prefix. i.e. shared.SharedObject
 */
include "shared.thrift"

/**
 * Thrift files can namespace, package, or prefix their output in various
 * target languages.
 */
namespace cpp tutorial
namespace java tutorial
php_namespace tutorial
namespace perl tutorial
namespace smalltalk.category Thrift.Tutorial

/**
 * Thrift lets you do typedefs to get pretty names for your types. Standard
 * C style here.
 */
typedef i32 MyInteger

/**
 * Thrift also lets you define constants for use across languages. Complex
 * types and structs are specified using JSON notation.
 */
const i32 INT32CONSTANT = 9853
const map<string,string> MAPCONSTANT = {'hello':'world', 'goodnight':'moon'}
```

```

* You can define enums, which are just 32 bit integers. Values are optional
* and start at 1 if not supplied, C style again.
*   ^ ThriftIDL page says "If no constant value is supplied,
*   the value is either 0 for the first element, or one greater than the
*   preceding value for any subsequent element" so I'm guessing that's a bug.
*   PS: http://enel.ucalgary.ca/People/Norman/enel315\_winter1997/enum\_types/ states that if values are not
supplied, they start at 0 and not 1.
*/
enum Operation {
    ADD = 1,
    SUBTRACT = 2,
    MULTIPLY = 3,
    DIVIDE = 4
}

/**
* Structs are the basic complex data structures. They are comprised of fields
* which each have an integer identifier, a type, a symbolic name, and an
* optional default value.
*
* Fields can be declared "optional", which ensures they will not be included
* in the serialized output if they aren't set. Note that this requires some
* manual management in some languages.
*/
struct Work {
    1: i32 num1 = 0,
    2: i32 num2,
    3: Operation op,
    4: optional string comment,
}

/**
* Structs can also be exceptions, if they are nasty.
*/
exception InvalidOperation {
    1: i32 what,
    2: string why
}

/**
* Ahh, now onto the cool part, defining a service. Services just need a name
* and can optionally inherit from another service using the extends keyword.
*/
service Calculator extends shared.SharedService {

    /**
    * A method definition looks like C code. It has a return type, arguments,
    * and optionally a list of exceptions that it may throw. Note that argument
    * lists and exception lists are specified using the exact same syntax as
    * field lists in struct or exception definitions. NOTE: Overloading of
    * methods is not supported; each method requires a unique name.
    */

    void ping(),

    i32 add(1:i32 num1, 2:i32 num2),

    i32 calculate(1:i32 logid, 2:Work w) throws (1:InvalidOperation ouch),

    /**
    * This method has an oneway modifier. That means the client only makes
    * a request and does not listen for any response at all. Oneway methods
    * must be void.
    *
    * The server may execute async invocations of the same client in parallel/
    * out of order.
    */
    oneway void zip(),
}

/**

```

```
* It's possible to declare more than one service per Thrift file.
*/
service CalculatorExtreme extends shared.SharedService {
    void pingExtreme(),
}

/**
 * That just about covers the basics. Take a look in the test/ folder for more
 * detailed examples. After you run this file, your generated code shows up
 * in folders with names gen-<language>. The generated code isn't too scary
 * to look at. It even has pretty indentation.
 */
```