syntax = "proto3";

package tensorflow;

import "tensorflow/core/framework/attr\_value.proto";

import "tensorflow/core/framework/node\_def.proto";

import "tensorflow/core/framework/op\_def.proto";

option cc\_enable\_arenas = true;

option java\_outer\_classname = "FunctionProtos";

option java\_multiple\_files = true;

option java\_package = "org.tensorflow.framework";

option go\_package = "github.com/tensorflow/tensorflow/tensorflow/go/core/framework/function\_go\_proto";

// A library is a set of named functions.

message FunctionDefLibrary {

repeated FunctionDef function = 1;

repeated GradientDef gradient = 2;

repeated RegisteredGradient registered\_gradients = 3;

}

// A function can be instantiated when the runtime can bind every attr

// with a value. When a GraphDef has a call to a function, it must

// have binding for every attr defined in the signature.

//

// TODO(zhifengc):

// \* device spec, etc.

message FunctionDef {

// The definition of the function's name, arguments, return values,

// attrs etc.

OpDef signature = 1;

// Attributes specific to this function definition.

map<string, AttrValue> attr = 5;

// Attributes for function arguments. These attributes are the same set of

// valid attributes as to \_Arg nodes.

message ArgAttrs {

map<string, AttrValue> attr = 1;

}

map<uint32, ArgAttrs> arg\_attr = 7;

// Unique IDs for each resource argument, used to track aliasing resources. If

// Argument A and Argument B alias each other, then

// resource\_arg\_unique\_ids[A.index] == resource\_arg\_unique\_ids[B.index].

//

// If this field is empty, none of the arguments could alias; otherwise, every

// resource argument should have an entry in this field.

//

// When instantiated, the unique IDs will be attached to the \_Arg nodes'

// "\_resource\_arg\_unique\_id" attribute.

map<uint32, uint32> resource\_arg\_unique\_id = 8;

// NOTE: field id 2 deleted on Jan 11, 2017, GraphDef version 21.

reserved 2;

// In both of the following fields, there is the need to specify an

// output that is used as either the input to another node (in

// `node\_def`) or as a return value of the function (in `ret`).

// Unlike the NodeDefs in GraphDef, we need to be able to specify a

// list in some cases (instead of just single outputs). Also, we

// need to be able to deal with lists of unknown length (so the

// output index may not be known at function definition time). So

// we use the following format instead:

// \* "fun\_in" where "fun\_in" is the name of a function input arg in

// the `signature` field above. This represents that input, whether

// it is a single tensor or a list.

// \* "fun\_in:0" gives the first element of a function input arg (a

// non-list input is considered a list of length 1 for these

// purposes).

// \* "node:out" where "node" is the name of a node in `node\_def` and

// "out" is the name one of its op's output arguments (the name

// comes from the OpDef of the node's op). This represents that

// node's output, whether it is a single tensor or a list.

// Note: We enforce that an op's output arguments are never

// renamed in the backwards-compatibility test.

// \* "node:out:0" gives the first element of a node output arg (a

// non-list output is considered a list of length 1 for these

// purposes).

//

// NOT CURRENTLY SUPPORTED (but may be in the future):

// \* "node:out:-1" gives last element in a node output list

// \* "node:out:1:" gives a list with all but the first element in a

// node output list

// \* "node:out::-1" gives a list with all but the last element in a

// node output list

// The body of the function. Unlike the NodeDefs in a GraphDef, attrs

// may have values of type `placeholder` and the `input` field uses

// the "output" format above.

// By convention, "op" in node\_def is resolved by consulting with a

// user-defined library first. If not resolved, "func" is assumed to

// be a builtin op.

repeated NodeDef node\_def = 3;

// A mapping from the output arg names from `signature` to the

// outputs from `node\_def` that should be returned by the function.

map<string, string> ret = 4;

// A mapping from control output names from `signature` to node names in

// `node\_def` which should be control outputs of this function.

map<string, string> control\_ret = 6;

}

// GradientDef defines the gradient function of a function defined in

// a function library.

//

// A gradient function g (specified by gradient\_func) for a function f

// (specified by function\_name) must follow the following:

//

// The function 'f' must be a numerical function which takes N inputs

// and produces M outputs. Its gradient function 'g', which is a

// function taking N + M inputs and produces N outputs.

//

// I.e. if we have

// (y1, y2, ..., y\_M) = f(x1, x2, ..., x\_N),

// then, g is

// (dL/dx1, dL/dx2, ..., dL/dx\_N) = g(x1, x2, ..., x\_N,

// dL/dy1, dL/dy2, ..., dL/dy\_M),

// where L is a scalar-value function of (x1, x2, ..., xN) (e.g., the

// loss function). dL/dx\_i is the partial derivative of L with respect

// to x\_i.

message GradientDef {

string function\_name = 1; // The function name.

string gradient\_func = 2; // The gradient function's name.

}

// RegisteredGradient stores a gradient function that is registered in the

// gradients library and used in the ops of a function in the function library.

// Unlike GradientDef, these gradients are identified by op type, and not

// directly linked to any function.

message RegisteredGradient {

string gradient\_func = 1; // The gradient function's name.

string registered\_op\_type = 2; // The gradient function's registered op type.

}