syntax = "proto3";

package tensorflow;

import "google/protobuf/any.proto";

import "tensorflow/core/framework/graph.proto";

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

import "tensorflow/core/framework/tensor\_shape.proto";

import "tensorflow/core/framework/types.proto";

import "tensorflow/core/protobuf/saved\_object\_graph.proto";

import "tensorflow/core/protobuf/saver.proto";

import "tensorflow/core/protobuf/struct.proto";

option cc\_enable\_arenas = true;

option java\_outer\_classname = "MetaGraphProtos";

option java\_multiple\_files = true;

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

option go\_package = "github.com/tensorflow/tensorflow/tensorflow/go/core/protobuf/for\_core\_protos\_go\_proto";

// NOTE: This protocol buffer is evolving, and will go through revisions in the

// coming months.

//

// Protocol buffer containing the following which are necessary to restart

// training, run inference. It can be used to serialize/de-serialize memory

// objects necessary for running computation in a graph when crossing the

// process boundary. It can be used for long term storage of graphs,

// cross-language execution of graphs, etc.

// MetaInfoDef

// GraphDef

// SaverDef

// CollectionDef

// TensorInfo

// SignatureDef

message MetaGraphDef {

// Meta information regarding the graph to be exported. To be used by users

// of this protocol buffer to encode information regarding their meta graph.

message MetaInfoDef {

// User specified Version string. Can be the name of the model and revision,

// steps this model has been trained to, etc.

string meta\_graph\_version = 1;

// A copy of the OpDefs used by the producer of this graph\_def.

// Descriptions and Ops not used in graph\_def are stripped out.

OpList stripped\_op\_list = 2;

// A serialized protobuf. Can be the time this meta graph is created, or

// modified, or name of the model.

google.protobuf.Any any\_info = 3;

// User supplied tag(s) on the meta\_graph and included graph\_def.

//

// MetaGraphDefs should be tagged with their capabilities or use-cases.

// Examples: "train", "serve", "gpu", "tpu", etc.

// These tags enable loaders to access the MetaGraph(s) appropriate for a

// specific use-case or runtime environment.

repeated string tags = 4;

// The \_\_version\_\_ string of the tensorflow build used to write this graph.

// This will be populated by the framework, which will overwrite any user

// supplied value.

string tensorflow\_version = 5;

// The \_\_git\_version\_\_ string of the tensorflow build used to write this

// graph. This will be populated by the framework, which will overwrite any

// user supplied value.

string tensorflow\_git\_version = 6;

// A flag to denote whether default-valued attrs have been stripped from

// the nodes in this graph\_def.

bool stripped\_default\_attrs = 7;

// FunctionDef name to aliases mapping.

map<string, string> function\_aliases = 8;

}

MetaInfoDef meta\_info\_def = 1;

// GraphDef.

GraphDef graph\_def = 2;

// SaverDef.

SaverDef saver\_def = 3;

// collection\_def: Map from collection name to collections.

// See CollectionDef section for details.

map<string, CollectionDef> collection\_def = 4;

// signature\_def: Map from user supplied key for a signature to a single

// SignatureDef.

map<string, SignatureDef> signature\_def = 5;

// Asset file def to be used with the defined graph.

repeated AssetFileDef asset\_file\_def = 6;

// Extra information about the structure of functions and stateful objects.

SavedObjectGraph object\_graph\_def = 7;

}

// CollectionDef should cover most collections.

// To add a user-defined collection, do one of the following:

// 1. For simple data types, such as string, int, float:

// tf.add\_to\_collection("your\_collection\_name", your\_simple\_value)

// strings will be stored as bytes\_list.

//

// 2. For Protobuf types, there are three ways to add them:

// 1) tf.add\_to\_collection("your\_collection\_name",

// your\_proto.SerializeToString())

//

// collection\_def {

// key: "user\_defined\_bytes\_collection"

// value {

// bytes\_list {

// value: "queue\_name: \"test\_queue\"\n"

// }

// }

// }

//

// or

//

// 2) tf.add\_to\_collection("your\_collection\_name", str(your\_proto))

//

// collection\_def {

// key: "user\_defined\_string\_collection"

// value {

// bytes\_list {

// value: "\n\ntest\_queue"

// }

// }

// }

//

// or

//

// 3) any\_buf = any\_pb2.Any()

// tf.add\_to\_collection("your\_collection\_name",

// any\_buf.Pack(your\_proto))

//

// collection\_def {

// key: "user\_defined\_any\_collection"

// value {

// any\_list {

// value {

// type\_url: "type.googleapis.com/tensorflow.QueueRunnerDef"

// value: "\n\ntest\_queue"

// }

// }

// }

// }

//

// 3. For Python objects, implement to\_proto() and from\_proto(), and register

// them in the following manner:

// ops.register\_proto\_function("your\_collection\_name",

// proto\_type,

// to\_proto=YourPythonObject.to\_proto,

// from\_proto=YourPythonObject.from\_proto)

// These functions will be invoked to serialize and de-serialize the

// collection. For example,

// ops.register\_proto\_function(ops.GraphKeys.GLOBAL\_VARIABLES,

// proto\_type=variable\_pb2.VariableDef,

// to\_proto=Variable.to\_proto,

// from\_proto=Variable.from\_proto)

message CollectionDef {

// NodeList is used for collecting nodes in graph. For example

// collection\_def {

// key: "summaries"

// value {

// node\_list {

// value: "input\_producer/ScalarSummary:0"

// value: "shuffle\_batch/ScalarSummary:0"

// value: "ImageSummary:0"

// }

// }

message NodeList {

repeated string value = 1;

}

// BytesList is used for collecting strings and serialized protobufs. For

// example:

// collection\_def {

// key: "trainable\_variables"

// value {

// bytes\_list {

// value: "\n\017conv1/weights:0\022\024conv1/weights/Assign

// \032\024conv1/weights/read:0"

// value: "\n\016conv1/biases:0\022\023conv1/biases/Assign\032

// \023conv1/biases/read:0"

// }

// }

// }

message BytesList {

repeated bytes value = 1;

}

// Int64List is used for collecting int, int64 and long values.

message Int64List {

repeated int64 value = 1 [packed = true];

}

// FloatList is used for collecting float values.

message FloatList {

repeated float value = 1 [packed = true];

}

// AnyList is used for collecting Any protos.

message AnyList {

repeated google.protobuf.Any value = 1;

}

oneof kind {

NodeList node\_list = 1;

BytesList bytes\_list = 2;

Int64List int64\_list = 3;

FloatList float\_list = 4;

AnyList any\_list = 5;

}

}

// Information about a Tensor necessary for feeding or retrieval.

message TensorInfo {

// For sparse tensors, The COO encoding stores a triple of values, indices,

// and shape.

message CooSparse {

// The shape of the values Tensor is [?]. Its dtype must be the dtype of

// the SparseTensor as a whole, given in the enclosing TensorInfo.

string values\_tensor\_name = 1;

// The indices Tensor must have dtype int64 and shape [?, ?].

string indices\_tensor\_name = 2;

// The dynamic logical shape represented by the SparseTensor is recorded in

// the Tensor referenced here. It must have dtype int64 and shape [?].

string dense\_shape\_tensor\_name = 3;

}

// Generic encoding for composite tensors.

message CompositeTensor {

// The serialized TypeSpec for the composite tensor.

TypeSpecProto type\_spec = 1;

// A TensorInfo for each flattened component tensor.

repeated TensorInfo components = 2;

}

oneof encoding {

// For dense `Tensor`s, the name of the tensor in the graph.

string name = 1;

// There are many possible encodings of sparse matrices

// (https://en.wikipedia.org/wiki/Sparse\_matrix). Currently, TensorFlow

// uses only the COO encoding. This is supported and documented in the

// SparseTensor Python class.

CooSparse coo\_sparse = 4;

// Generic encoding for CompositeTensors.

CompositeTensor composite\_tensor = 5;

}

DataType dtype = 2;

// The static shape should be recorded here, to the extent that it can

// be known in advance. In the case of a SparseTensor, this field describes

// the logical shape of the represented tensor (aka dense\_shape).

TensorShapeProto tensor\_shape = 3;

}

// SignatureDef defines the signature of a computation supported by a TensorFlow

// graph.

//

// For example, a model with two loss computations, sharing a single input,

// might have the following signature\_def map, in a MetaGraphDef message.

//

// Note that across the two SignatureDefs "loss\_A" and "loss\_B", the input key,

// output key, and method\_name are identical, and will be used by system(s) that

// implement or rely upon this particular loss method. The output tensor names

// differ, demonstrating how different outputs can exist for the same method.

//

// signature\_def {

// key: "loss\_A"

// value {

// inputs {

// key: "input"

// value {

// name: "input:0"

// dtype: DT\_STRING

// tensor\_shape: ...

// }

// }

// outputs {

// key: "loss\_output"

// value {

// name: "loss\_output\_A:0"

// dtype: DT\_FLOAT

// tensor\_shape: ...

// }

// }

// method\_name: "some/package/compute\_loss"

// }

// ...

// }

// signature\_def {

// key: "loss\_B"

// value {

// inputs {

// key: "input"

// value {

// name: "input:0"

// dtype: DT\_STRING

// tensor\_shape: ...

// }

// }

// outputs {

// key: "loss\_output"

// value {

// name: "loss\_output\_B:0"

// dtype: DT\_FLOAT

// tensor\_shape: ...

// }

// }

// method\_name: "some/package/compute\_loss"

// }

// ...

// }

message SignatureDef {

// Named input parameters.

map<string, TensorInfo> inputs = 1;

// Named output parameters.

map<string, TensorInfo> outputs = 2;

// Extensible method\_name information enabling third-party users to mark a

// SignatureDef as supporting a particular method. This enables producers and

// consumers of SignatureDefs, e.g. a model definition library and a serving

// library to have a clear hand-off regarding the semantics of a computation.

//

// Note that multiple SignatureDefs in a single MetaGraphDef may have the same

// method\_name. This is commonly used to support multi-headed computation,

// where a single graph computation may return multiple results.

string method\_name = 3;

}

// An asset file def for a single file or a set of sharded files with the same

// name.

message AssetFileDef {

// The tensor to bind the asset filename to.

TensorInfo tensor\_info = 1;

// The filename within an assets directory. Note: does not include the path

// prefix, i.e. directories. For an asset at /tmp/path/vocab.txt, the filename

// would be "vocab.txt".

string filename = 2;

}