syntax = "proto3";

package tensorflow;

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

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

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

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

// `StructuredValue` represents a dynamically typed value representing various

// data structures that are inspired by Python data structures typically used in

// TensorFlow functions as inputs and outputs.

//

// For example when saving a Layer there may be a `training` argument. If the

// user passes a boolean True/False, that switches between two concrete

// TensorFlow functions. In order to switch between them in the same way after

// loading the SavedModel, we need to represent "True" and "False".

//

// A more advanced example might be a function which takes a list of

// dictionaries mapping from strings to Tensors. In order to map from

// user-specified arguments `[{"a": tf.constant(1.)}, {"q": tf.constant(3.)}]`

// after load to the right saved TensorFlow function, we need to represent the

// nested structure and the strings, recording that we have a trace for anything

// matching `[{"a": tf.TensorSpec(None, tf.float32)}, {"q": tf.TensorSpec([],

// tf.float64)}]` as an example.

//

// Likewise functions may return nested structures of Tensors, for example

// returning a dictionary mapping from strings to Tensors. In order for the

// loaded function to return the same structure we need to serialize it.

//

// This is an ergonomic aid for working with loaded SavedModels, not a promise

// to serialize all possible function signatures. For example we do not expect

// to pickle generic Python objects, and ideally we'd stay language-agnostic.

message StructuredValue {

// The kind of value.

oneof kind {

// Represents None.

NoneValue none\_value = 1;

// Represents a double-precision floating-point value (a Python `float`).

double float64\_value = 11;

// Represents a signed integer value, limited to 64 bits.

// Larger values from Python's arbitrary-precision integers are unsupported.

sint64 int64\_value = 12;

// Represents a string of Unicode characters stored in a Python `str`.

// In Python 3, this is exactly what type `str` is.

// In Python 2, this is the UTF-8 encoding of the characters.

// For strings with ASCII characters only (as often used in TensorFlow code)

// there is effectively no difference between the language versions.

// The obsolescent `unicode` type of Python 2 is not supported here.

string string\_value = 13;

// Represents a boolean value.

bool bool\_value = 14;

// Represents a TensorShape.

tensorflow.TensorShapeProto tensor\_shape\_value = 31;

// Represents an enum value for dtype.

tensorflow.DataType tensor\_dtype\_value = 32;

// Represents a value for tf.TensorSpec.

TensorSpecProto tensor\_spec\_value = 33;

// Represents a value for tf.TypeSpec.

TypeSpecProto type\_spec\_value = 34;

// Represents a value for tf.BoundedTensorSpec.

BoundedTensorSpecProto bounded\_tensor\_spec\_value = 35;

// Represents a list of `Value`.

ListValue list\_value = 51;

// Represents a tuple of `Value`.

TupleValue tuple\_value = 52;

// Represents a dict `Value`.

DictValue dict\_value = 53;

// Represents Python's namedtuple.

NamedTupleValue named\_tuple\_value = 54;

}

}

// Represents None.

message NoneValue {}

// Represents a Python list.

message ListValue {

repeated StructuredValue values = 1;

}

// Represents a Python tuple.

message TupleValue {

repeated StructuredValue values = 1;

}

// Represents a Python dict keyed by `str`.

// The comment on Unicode from Value.string\_value applies analogously.

message DictValue {

map<string, StructuredValue> fields = 1;

}

// Represents a (key, value) pair.

message PairValue {

string key = 1;

StructuredValue value = 2;

}

// Represents Python's namedtuple.

message NamedTupleValue {

string name = 1;

repeated PairValue values = 2;

}

// A protobuf to represent tf.TensorSpec.

message TensorSpecProto {

string name = 1;

tensorflow.TensorShapeProto shape = 2;

tensorflow.DataType dtype = 3;

}

// A protobuf to represent tf.BoundedTensorSpec.

message BoundedTensorSpecProto {

string name = 1;

tensorflow.TensorShapeProto shape = 2;

tensorflow.DataType dtype = 3;

tensorflow.TensorProto minimum = 4;

tensorflow.TensorProto maximum = 5;

}

// Represents a tf.TypeSpec

message TypeSpecProto {

enum TypeSpecClass {

UNKNOWN = 0;

SPARSE\_TENSOR\_SPEC = 1; // tf.SparseTensorSpec

INDEXED\_SLICES\_SPEC = 2; // tf.IndexedSlicesSpec

RAGGED\_TENSOR\_SPEC = 3; // tf.RaggedTensorSpec

TENSOR\_ARRAY\_SPEC = 4; // tf.TensorArraySpec

DATA\_DATASET\_SPEC = 5; // tf.data.DatasetSpec

DATA\_ITERATOR\_SPEC = 6; // IteratorSpec from data/ops/iterator\_ops.py

OPTIONAL\_SPEC = 7; // tf.OptionalSpec

PER\_REPLICA\_SPEC = 8; // PerReplicaSpec from distribute/values.py

VARIABLE\_SPEC = 9; // tf.VariableSpec

ROW\_PARTITION\_SPEC = 10; // RowPartitionSpec from ragged/row\_partition.py

reserved 11;

REGISTERED\_TYPE\_SPEC = 12; // The type registered as type\_spec\_class\_name.

EXTENSION\_TYPE\_SPEC = 13; // Subclasses of tf.ExtensionType

}

TypeSpecClass type\_spec\_class = 1;

// The value returned by TypeSpec.\_serialize().

StructuredValue type\_state = 2;

// The name of the TypeSpec class.

// \* If type\_spec\_class == REGISTERED\_TYPE\_SPEC, the TypeSpec class is

// the one registered under this name. For types registered outside

// core TensorFlow by an add-on library, that library must be loaded

// before this value can be deserialized by nested\_structure\_coder.

// \* If type\_spec\_class specifies a particular TypeSpec class, this field is

// redundant with the type\_spec\_class enum, and is only used for error

// reporting in older binaries that do not know the tupe\_spec\_class enum.

string type\_spec\_class\_name = 3;

// The number of flat tensor components required by this TypeSpec.

int32 num\_flat\_components = 4;

}