syntax = "proto3";

package tensorflow;

option cc\_enable\_arenas = true;

option java\_outer\_classname = "FullTypeProtos";

option java\_multiple\_files = true;

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

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

// Experimental. Represents the complete type information of a TensorFlow value.

enum FullTypeId {

// The default represents an uninitialized values.

TFT\_UNSET = 0;

// Type symbols. Used to construct more complex type expressions like

// algebraic data types.

// Type variables may serve as placeholder for any other type ID in type

// templates.

//

// Examples:

// TFT\_DATASET[TFT\_VAR["T"]] is a Dataset returning a type indicated by "T".

// TFT\_TENSOR[TFT\_VAR["T"]] is a Tensor of n element type indicated by "T".

// TFT\_TENSOR[TFT\_VAR["T"]], TFT\_TENSOR[TFT\_VAR["T"]] are two tensors of

// identical element types.

// TFT\_TENSOR[TFT\_VAR["P"]], TFT\_TENSOR[TFT\_VAR["Q"]] are two tensors of

// independent element types.

//

TFT\_VAR = 1;

// Wildcard type. Describes a parameter of unknown type. In TensorFlow, that

// can mean either a "Top" type (accepts any type), or a dynamically typed

// object whose type is unknown in context.

// Important: "unknown" does not necessarily mean undeterminable!

TFT\_ANY = 2;

// The algebraic product type. This is an algebraic type that may be used just

// for logical grouping. Not to confused with TFT\_TUPLE which describes a

// concrete object of several elements.

//

// Example:

// TFT\_DATASET[TFT\_PRODUCT[TFT\_TENSOR[TFT\_INT32], TFT\_TENSOR[TFT\_FLOAT64]]]

// is a Dataset producing two tensors, an integer one and a float one.

//

TFT\_PRODUCT = 3;

// Represents a named field, with the name stored in the attribute.

//

// Parametrization:

// TFT\_NAMED[<type>]{<name>}

// \* <type> is the type of the field

// \* <name> is the field name, as string (thpugh can theoretically be an int

// as well)

//

// Example:

// TFT\_RECORD[

// TFT\_NAMED[TFT\_TENSOR[TFT\_INT32]]{'foo'},

// TFT\_NAMED[TFT\_TENSOR[TFT\_FLOAT32]]{'bar'},

// ]

// is a structure with two fields, an int tensor "foo" and a float tensor

// "bar".

TFT\_NAMED = 4;

// Template definition. Expands the variables by repeating a template as

// arguments of container.

//

// Parametrization:

// TFT\_FOR\_EACH[<container\_type>, <template>, <expansions>]

// \* <container\_type> is the type of the container that the template will be

// expanded into

// \* <template> is any type definition that potentially contains type

// variables

// \* <expansions> is a TFT\_VAR and may include more types in the future

//

// Example:

// TFT\_FOR\_EACH[

// TFT\_PRODUCT,

// TFT\_TENSOR[TFT\_VAR["t"]],

// TFT\_VAR["t"]

// ]

// will substitute a T = TFT\_INT32 to TFT\_PRODUCT[TFT\_TENSOR[TFT\_INT32]]

// and a T = (TFT\_INT32, TFT\_INT64) to

// TFT\_PRODUCT[TFT\_TENSOR[TFT\_INT32], TFT\_TENSOR[TFT\_INT64]].

TFT\_FOR\_EACH = 20;

// Callable types describe functions and ops.

//

// Parametrization:

// TFT\_CALLABLE[<arg type>, <return type>]

// \* <arg type> is the type of the arguments; TFT\_PRODUCT represents

// multiple

// arguments.

// \* <return type> is the return type; TFT\_PRODUCT represents multiple

// return values (that means that callables returning multiple things

// don't necessarily return a single tuple).

//

// Example:

// TFT\_CALLABLE[

// TFT\_ANY,

// TFT\_PRODUCT[TFT\_TENSOR[TFT\_INT32], TFT\_TENSOR[TFT\_FLOAT64]],

// ]

// is a callable with unspecified (for now) input arguments, and

// two return values of type tensor.

//

TFT\_CALLABLE = 100;

// Concrete type IDs, representing "proper" data types that can describe

// runtime TensorFlow objects.

// The usual Tensor. This is a parametric type.

//

// Parametrization:

// TFT\_TENSOR[<element type>, <shape type>]

// \* <element type> is currently limited to one of the element types

// defined below.

// \* <shape type> is not yet defined, and may only be TFT\_UNKNOWN for now.

//

// A TFT\_SHAPE type will be defined in the future.

//

// Example:

// TFT\_TENSOR[TFT\_INT32, TFT\_UNKNOWN]

// is a Tensor of int32 element type and unknown shape.

//

// TODO(mdan): Define TFT\_SHAPE and add more examples.

TFT\_TENSOR = 1000;

// Array (or tensorflow::TensorList in the variant type registry).

// Note: this is not to be confused with the deprecated `TensorArray\*` ops

// which are not supported by FullType.

// This type represents a random-access list whose elements can be

// described by a single type. Although immutable, Array is expected to

// support efficient mutation semantics (i.e. element update) in the

// user-facing API.

// The element type may be generic or even TFT\_ANY for a heterogenous list.

//

// Parametrization:

// TFT\_ARRAY[<element type>]

// \* <element type> may be any concrete type.

//

// Examples:

// TFT\_ARRAY[TFT\_TENSOR[TFT\_INT32]] is a TensorArray holding int32 Tensors

// of any shape.

// TFT\_ARRAY[TFT\_TENSOR[TFT\_UNKNOWN]] is a TensorArray holding Tensors of

// mixed element types.

// TFT\_ARRAY[TFT\_UNKNOWN] is a TensorArray holding any element type.

// TFT\_ARRAY[] is equivalent to TFT\_ARRAY[TFT\_UNKNOWN].

// TFT\_ARRAY[TFT\_ARRAY[]] is an array or arrays (of unknown types).

TFT\_ARRAY = 1001;

// Optional (or tensorflow::OptionalVariant in the variant type registry).

// This type represents a value that may either hold an element of a single

// specified type, or nothing at all.

//

// Parametrization:

// TFT\_OPTIONAL[<element type>]

// \* <element type> may be any concrete type.

//

// Examples:

// TFT\_OPTIONAL[TFT\_TENSOR[TFT\_INT32]] is an Optional holding an int32

// Tensor of any shape.

TFT\_OPTIONAL = 1002;

// Literal types describe compile-time constant values.

// Literal types may also participate in dependent types.

//

// Parametrization:

// TFT\_LITERAL[<value type>]{<value>}

// \* <value type> may be any concrete type compatible that can hold <value>

// \* <value> is the type's attribute, and holds the actual literal value

//

// Examples:

// TFT\_LITERAL[TFT\_INT32]{1} is the compile-time constant 1.

TFT\_LITERAL = 1003;

// Type attributes. These always appear in the parametrization of a type,

// never alone. For example, there is no such thing as a "bool" TensorFlow

// object (for now).

// The bool element type.

// TODO(mdan): Quantized types, legacy representations (e.g. ref)

TFT\_BOOL = 200;

// Integer element types.

TFT\_UINT8 = 201;

TFT\_UINT16 = 202;

TFT\_UINT32 = 203;

TFT\_UINT64 = 204;

TFT\_INT8 = 205;

TFT\_INT16 = 206;

TFT\_INT32 = 207;

TFT\_INT64 = 208;

// Floating-point element types.

TFT\_HALF = 209;

TFT\_FLOAT = 210;

TFT\_DOUBLE = 211;

TFT\_BFLOAT16 = 215;

// Complex element types.

// TODO(mdan): Represent as TFT\_COMPLEX[TFT\_DOUBLE] instead?

TFT\_COMPLEX64 = 212;

TFT\_COMPLEX128 = 213;

// The string element type.

TFT\_STRING = 214;

// Other types that we don't know yet whether they will become part of the

// core type system or be consisdered third-party (and consequently moved to

// user-defined type mechanisms). Presently, they are effectively in the core

// type system, because key compilation passes like Placer account for their

// existence.

// Datasets created by tf.data ops and APIs. Datasets have generator/iterable

// semantics, that is, one can construct an iterator from them. Like

// Array, they are considered to return elements that can be described

// by a single type. Unlike Array, they do not support random access or

// mutation, and can potentially produce an infinite number of elements.

// A datasets can produce logical structures (e.g. multiple elements). This

// is expressed using TFT\_PRODUCT.

//

//

// Parametrization: TFT\_ARRAY[<element type>].

// \* <element type> may be a concrete type or a type symbol. It represents

// the data type of the elements produced by the dataset.

//

// Examples:

// TFT\_DATSET[TFT\_TENSOR[TFT\_INT32]] is a Dataset producing single int32

// Tensors of unknown shape.

// TFT\_DATSET[TFT\_PRODUCT[TFT\_TENSOR[TFT\_INT32], TFT\_TENSOR[TFT\_FLOAT32]] is

// a Dataset producing pairs of Tensors, one integer and one float.

// Note: The high ID number is to prepare for the eventuality that Datasets

// will be supported by user types in the future.

TFT\_DATASET = 10102;

// A ragged tensor created by tf.ragged ops and APIs.

//

// Parametrization: TFT\_RAGGED[<element\_type>].

TFT\_RAGGED = 10103;

// A mutex lock tensor, produced by tf.raw\_ops.MutexLock.

// Unlike strict execution models, where ownership of a lock is denoted by

// "running after the lock has been acquired", in non-strict mode, lock

// ownership is in the true sense: "the op argument representing the lock is

// available".

// Mutex locks are the dynamic counterpart of control dependencies.

// TODO(mdan): Properly document this thing.

//

// Parametrization: TFT\_MUTEX\_LOCK[].

TFT\_MUTEX\_LOCK = 10202;

// The equivalent of a Tensor with DT\_VARIANT dtype, kept here to simplify

// translation. This type should not normally appear after type inference.

// Note that LEGACY\_VARIANT != ANY: TENSOR[INT32] is a subtype of ANY, but is

// not a subtype of LEGACY\_VARIANT.

TFT\_LEGACY\_VARIANT = 10203;

}

// Highly experimental and very likely to change.

// This encoding uses tags instead of dedicated messages for regularity. In

// particular the encoding imposes no restrictions on what the parameters of any

// type should be, which in particular needs to be true for type symbols.

message FullTypeDef {

// The principal type represented by this object. This may be a concrete type

// (Tensor, Dataset) a type variable (used for dependent types) a type

// symbol (Any, Union). See FullTypeId for details.

FullTypeId type\_id = 1;

repeated FullTypeDef args = 2;

// Literal values of this type object, if the the type admits one.

// For example, a type variable admits a string attribute - its name.

// Shape-related types may admit int attributes - their static shape values.

// Fields for more data types to be added as needed.

oneof attr {

string s = 3;

int64 i = 4;

// TODO(mdan): list/tensor, map? Need to reconcile with TFT\_RECORD, etc.

}

}