syntax = "proto3";

package tensorflow;

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

import "tensorflow/core/protobuf/graph\_debug\_info.proto";

option cc\_enable\_arenas = true;

option java\_outer\_classname = "DebugEventProtos";

option java\_multiple\_files = true;

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

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

// Available modes for extracting debugging information from a Tensor.

// TODO(cais): Document the detailed column names and semantics in a separate

// markdown file once the implementation settles.

enum TensorDebugMode {

UNSPECIFIED = 0;

// Only records what tensors are computed, eagerly or in graphs.

// No information regarding the value of the tensor is available.

NO\_TENSOR = 1;

// A minimalist health summary for float-type tensors.

// Contains information only about the presence/absence of pathological

// values including Infinity and NaN.

// Applicable only to float dtypes.

CURT\_HEALTH = 2;

// A concise health summary for float-type tensors.

// Contains more information that CURT\_HEALTH.

// Infinity and NaN are treated differently.

// Applicable only to float and integer dtypes.

CONCISE\_HEALTH = 3;

// A detailed health summary.

// Contains further detailed information than `CONCISE\_HEALTH`.

// Information about device, dtype and shape are included.

// Counts for various types of values (Infinity, NaN, negative, zero,

// positive) are included.

// Applicable to float, integer and boolean dtypes.

FULL\_HEALTH = 4;

// Provides full runtime shape information, up to a maximum rank, beyond

// which the dimension sizes are truncated.

SHAPE = 5;

// Full numeric summary.

// Including device, dtype, shape, counts of various types of values

// (Infinity, NaN, negative, zero, positive), and summary statistics

// (minimum, maximum, mean and variance).

// Applicable to float, integer and boolean dtypes.

FULL\_NUMERICS = 6;

// Full tensor value.

FULL\_TENSOR = 7;

// Reduce the elements of a tensor to a rank-1 tensor of shape [3], in which

// - the 1st element is -inf if any element of the tensor is -inf,

// or zero otherwise.

// - the 2nd element is +inf if any element of the tensor is +inf,

// or zero otherwise.

// - the 3rd element is nan if any element of the tensor is nan, or zero

// otherwise.

REDUCE\_INF\_NAN\_THREE\_SLOTS = 8;

}

// An Event related to the debugging of a TensorFlow program.

message DebugEvent {

// Timestamp in seconds (with microsecond precision).

double wall\_time = 1;

// Step of training (if available).

int64 step = 2;

oneof what {

// Metadata related to this debugging data.

DebugMetadata debug\_metadata = 3;

// The content of a source file.

SourceFile source\_file = 4;

// A stack frame (filename, line number and column number, function name and

// code string) with ID.

StackFrameWithId stack\_frame\_with\_id = 6;

// The creation of an op within a graph (e.g., a FuncGraph compiled from

// a Python function).

GraphOpCreation graph\_op\_creation = 7;

// Information about a debugged graph.

DebuggedGraph debugged\_graph = 8;

// Execution of an op or a Graph (e.g., a tf.function).

Execution execution = 9;

// A graph execution trace: Contains information about the intermediate

// tensors computed during the graph execution.

GraphExecutionTrace graph\_execution\_trace = 10;

// The ID of the graph (i.e., FuncGraph) executed here: applicable only

// to the execution of a FuncGraph.

string graph\_id = 11;

// A device on which debugger-instrumented ops and/or tensors reside.

DebuggedDevice debugged\_device = 12;

}

}

// Metadata about the debugger and the debugged TensorFlow program.

message DebugMetadata {

// Version of TensorFlow.

string tensorflow\_version = 1;

// Version of the DebugEvent file format.

// Has a format of "debug.Event:<number>", e.g., "debug.Event:1".

string file\_version = 2;

// A unique ID for the current run of tfdbg.

// A run of tfdbg is defined as a TensorFlow job instrumented by tfdbg.

// Multiple hosts in a distributed TensorFlow job instrumented by tfdbg

// have the same ID.

string tfdbg\_run\_id = 3;

}

// Content of a source file involved in the execution of the debugged TensorFlow

// program.

message SourceFile {

// Path to the file.

string file\_path = 1;

// Name of the host on which the file is located.

string host\_name = 2;

// Line-by-line content of the file.

repeated string lines = 3;

}

// A stack frame with ID.

message StackFrameWithId {

// A unique ID for the stack frame: A UUID-like string.

string id = 1;

// Stack frame, i.e., a frame of a stack trace, containing information

// regarding the file name, line number, function name, code content

// of the line, and column number (if available).

GraphDebugInfo.FileLineCol file\_line\_col = 2;

}

// Code location information: A stack trace with host-name information.

// Instead of encoding the detailed stack trace, this proto refers to IDs of

// stack frames stored as `StackFrameWithId` protos.

message CodeLocation {

// Host name on which the source files are located.

string host\_name = 1;

// ID to a stack frame, each of which is pointed to

// by a unique ID. The ordering of the frames is consistent with Python's

// `traceback.extract\_tb()`.

repeated string stack\_frame\_ids = 2;

}

// The creation of an op in a TensorFlow Graph (e.g., FuncGraph in TF2).

message GraphOpCreation {

// Type of the op (e.g., "MatMul").

string op\_type = 1;

// Name of the op (e.g., "Dense/MatMul\_1").

string op\_name = 2;

// Name of the graph that the op is a part of (if available).

string graph\_name = 3;

// Unique ID of the graph (generated by debugger).

// This is the ID of the immediately-enclosing graph.

string graph\_id = 4;

// Name of the device that the op is assigned to (if available).

string device\_name = 5;

// Names of the input tensors to the op.

repeated string input\_names = 6;

// Number of output tensors emitted by the op.

int32 num\_outputs = 7;

// The unique ID for code location (stack trace) of the op's creation.

CodeLocation code\_location = 8;

// Unique IDs for the output tensors of this op.

repeated int32 output\_tensor\_ids = 9;

}

// A debugger-instrumented graph.

message DebuggedGraph {

// An ID for the graph.

// This can be used up to look up graph names. Generated by the debugger.

string graph\_id = 1;

// Name of the graph (if available).

string graph\_name = 2;

// Names of the instrumented ops. This can be used to look up op name

// based on the numeric-summary tensors (2nd column).

repeated string instrumented\_ops = 3;

// Original (uninstrumented) GraphDef (if available).

bytes original\_graph\_def = 4;

// An encoded version of a GraphDef.

// This graph may include the debugger-inserted ops.

bytes instrumented\_graph\_def = 5;

// IDs of the immediate enclosing context (graph), if any.

string outer\_context\_id = 6;

}

// A device on which ops and/or tensors are instrumented by the debugger.

message DebuggedDevice {

// Name of the device.

string device\_name = 1;

// A debugger-generated ID for the device. Guaranteed to be unique within

// the scope of the debugged TensorFlow program, including single-host and

// multi-host settings.

// TODO(cais): Test the uniqueness guarantee in multi-host settings.

int32 device\_id = 2;

}

// Data relating to the eager execution of an op or a Graph.

// For a op that generates N output tensors (N >= 0), only one

// Execution proto will be used to describe the execution event.

message Execution {

// Op type (e.g., "MatMul").

// In the case of a Graph, this is the name of the Graph.

string op\_type = 1;

// Number of output tensors.

int32 num\_outputs = 2;

// The graph that's executed: applicable only to the eager

// execution of a FuncGraph.

string graph\_id = 3;

// IDs of the input tensors (if available).

repeated int64 input\_tensor\_ids = 4;

// IDs of the output tensors (if availbable).

// If specified, must have the same length as tensor\_protos.

repeated int64 output\_tensor\_ids = 5;

// Type of the tensor value encapsulated in this proto.

TensorDebugMode tensor\_debug\_mode = 6;

// Output Tensor values in the type described by `tensor\_value\_type`.

// The length of this should match `num\_outputs`.

repeated TensorProto tensor\_protos = 7;

// Stack trace of the eager execution.

CodeLocation code\_location = 8;

// Debugged-generated IDs of the devices on which the output tensors reside.

// To look up details about the device (e.g., name), cross-reference this

// field with the DebuggedDevice messages.

repeated int32 output\_tensor\_device\_ids = 9;

// TODO(cais): When backporting to V1 Session.run() support, add more fields

// such as fetches and feeds.

}

// Data relating to an execution of a Graph (e.g., an eager execution of a

// FuncGraph).

// The values of the intermediate tensors computed in the graph are recorded

// in this proto. A graph execution may correspond to one or more pieces of

// `GraphExecutionTrace`, depending on whether the instrumented tensor values

// are summarized in an aggregated or separate fashion.

message GraphExecutionTrace {

// Unique ID of the context that the executed op(s) belong to (e.g., a

// compiled concrete tf.function).

string tfdbg\_context\_id = 1;

// Name of the op (applicable only in the case of the `FULL\_TENSOR` trace

// level).

string op\_name = 2;

// Output slot of the tensor (applicable only in the case of the `FULL\_TENSOR`

// trace level).

int32 output\_slot = 3;

// Type of the tensor value encapsulated in this proto.

TensorDebugMode tensor\_debug\_mode = 4;

// Tensor value in the type described by `tensor\_value\_type`.

// This tensor may summarize the value of a single intermediate op of the

// graph, or those of multiple intermediate tensors.

TensorProto tensor\_proto = 5;

// Name of the device that the op belongs to.

string device\_name = 6;

}