/**
 * Starts BLE broadcasts and scanning based on the defined protocol.
 * If not previously used, this shows a user dialog for consent to start exposure
detection and get permission. Matching options can optionally be provided to
tune the matching algorithm (for example, setting a signal strength attenuation
or duration threshold).
 * Calls back when data is to be pushed or pulled from the client, see
 * {@link ExposureNotificationCallback}.
 */
Task<Void> start(
    PendingIntent exposureNotificationCallback,
    @Nullable MatchingOptions matchingOptions);

@IntDef({...})
@interface Status {
    int SUCCESS = 0;
    int FAILED_REJECTED_OPT_IN = 1;
    int FAILED_SERVICE_DISABLED = 2;
    int FAILED_BLUETOOTH_SCANNING_DISABLED = 3;
    int FAILED_TEMPORARILY_DISABLED = 4;
    int FAILED_INSUFFICIENT_STORAGE = 5;
    int FAILED_INTERNAL = 6;
}

/**
 * Optional matching options that can be provided when initializing the service.
 */
class MatchingOptions {
/**
 * The signal strength attenuation value that must be reached within the exposure
duration before the match is returned to the client. Attenuation is defined
as the advertiser’s TX power minus the scanner’s RSSI.
 * This value must have range 0-255.
 */
int attenuationValueThreshold;

/**
 * The time that 2 devices must be within range of each other before
 * the match is returned to the client.
 *
* If set to 0, then all matches within the
  * {@link #getAttenuationValueThreshold} will be considered.
  */
  int durationMinutesThreshold;
}

/**
 * Handles an intent which was invoked via the exposureNotificationCallback and
 * calls the corresponding {@link ExposureNotificationCallback} methods.
 */
Task<Void> handleIntent(
  Intent intentCallback, ExposureNotificationCallback callback);

/** Callback used to notify client apps when they need to perform an action. */
interface ExposureNotificationCallback {
  /** Notifies the client that the user has been exposed to a diagnosis key. */
  void onExposure();

  /** Requests the client to provide a list of all newly uploaded diagnosis keys
   * from the server. This should be done by invoking
   * {@link #provideDiagnosisKeys}. */
  void onDiagnosisKeysRequested();
}

/** A key generated for advertising over a window of time. */
class TemporaryExposureKey {
  /** The randomly generated Temporary Exposure Key information. */
  byte[] keyData;

  /**
   * A number describing when a key starts.
   * It is equal to startTimeOfKeySinceEpochInSecs / (60 * 10).
   */
  long rollingStartNumber;
}

/**
 * Disables advertising and scanning. Contents of the database and keys will
 * remain.
 * If the client app has been uninstalled by the user, this will be automatically
 * invoked and the database and keys will be wiped from the device.
 */
Task<Void> stop();
/**
 * Indicates whether exposure notifications are currently running for the
 * requesting app.
 */
 Task<Boolean> isEnabled();

/**
 * Gets {@link TemporaryExposureKey} history to be stored on the server.
 * This should only be done after proper verification is performed on the
 * client side that the user is diagnosed positive.
 * The keys provided here will only be from previous days; keys will not be
 * released until after they are no longer an active exposure key.
 * This shows a user permission dialog for sharing and uploading data to the
 * server.
 */
 Task<List<TemporaryExposureKey>> getTemporaryExposureKeyHistory();

/**
 * Provides a list of diagnosis keys for exposure checking. The keys are to be
 * provided by a centralized service (e.g. synced from the server). Diagnosis
 * keys older than the relevant period will be ignored.
 * When invoked after the {@link ExposureNotificationCallback#onDiagnosisKeysRequested} callback, this triggers a
 * recalculation of exposure status which can be obtained via {@link #getExposureSummary} after the calculation has finished.
 * Should be called with a maximum of {@link getMaxDiagnosisKeys()} keys at a
 * time, waiting until the Task finishes before providing the next batch.
 */
 Task<Void> provideDiagnosisKeys(List<TemporaryExposureKey> keys);

/**
 * The maximum number of keys to pass into {@link #provideDiagnosisKeys} at any
 * given time.
 */
 Task<Integer> getMaxDiagnosisKeys();

/**
 * Gets a summary of the latest exposure calculation. The calculation happens
asynchronously, the most recent check’s result will be returned immediately.

```
Task<ExposureSummary> getExposureSummary();
```

```
**
* Summary information about recent exposures.
* The client can get this information via \{@link #getExposureSummary\}. *
*/

```

```
class ExposureSummary {
/**
 * Days since last match to a diagnosis key from the server. 0 is today, 1 is
 * yesterday, etc. Only valid if \{@link #getMatchedKeysCount\} > 0.
 */

    int daysSinceLastExposure;

/** Number of matched diagnosis keys. */
    int matchedKeyCount;
}
```

```
**
* Gets detailed information about exposures that have occurred. The calculation
* happens asynchronously, the most recent check’s result will be returned
* immediately.
*
* When multiple \{@link ExposureInformation\} objects are returned, they can
* be:
* <ul>
*   <li>Multiple encounters with a single diagnosis key.
*   <li>Multiple encounters with the same device across key rotation boundaries.
*   <li>Encounters with multiple devices.
* </ul>
*
* Plans to ensure user transparency in the use of this function are currently
* being evaluated.
*/

Task<List<ExposureInformation>> getExposureInformation();
```

```
**
* Information about an exposure, meaning a single diagnosis key
* over a contiguous period of time specified by durationMinutes.
*
* The client can get the exposure information via \{@link #getExposureInformation\}. *
*/

```

```
class ExposureInformation {
/** Day level resolution that the exposure occurred. */

    Date date;
```
/** Length of exposure in 5 minute increments, with a 30 minute maximum. */
int durationMinutes;

/**
 * Signal strength attenuation, representing the closest the two devices were
 * within the duration of the exposure. This value is the advertiser's TX power
 * minus the receiver's maximum RSSI.
 *
 * Note: This value may be misleading, higher attenuation does not necessarily
 * mean farther away. Phone in pocket vs hand can greatly affect this value,
 * along with other situations that can block the signal.
 *
 * This value will be in the range 0-255.
 */
int attenuationValue;

/**
 * Delete all stored data associated with the user including exposure keys,
 * bluetooth scan history, and previously detected exposures.
 */
Task<Void> resetAllData();

/** Reset the current exposure key, leaving past exposure history in place. */
Task<Void> resetTemporaryExposureKey();
Revision History

v1.1 - April 21, 2020

- Renamed from Contact Tracing -> Exposure Notification
- Update start()
  - Allow sending in matching options, including attenuationValueThreshold and durationMinutesThreshold
  - Update documentation
- Update ExposureNotificationCallback javadocs and method naming to better conform to Android conventions
- Renamed DailyTracingKey to TemporaryExposureKey
  - date to rollingStartNumber per crypto spec
- Renamed startSharingTemporaryTracingKeys to getTemporaryExposureKeyHistory and return the keys directly
- Renamed hasContact to getExposureSummary and added ExposureSummary
- Changed ContactInfo to ExposureInformation
  - Added attenuationValue
  - Documented maximum durationMinutes to 30.
- Updated return types for all methods to conform with Google Play services style conventions
- Updated documentation on provideDiagnosisKeys that older keys will be ignored
- Added methods for a client to resetAllData or resetTemporaryExposureKey
- General doc cleanup

v1.0 - April 10, 2020

- Initial draft