Notification API™ Reference

ShotSpotter®
Notification
Engine
version 2.6

SST™

Table of Contents

1 Technical Background

1.1 Wide-Area Acoustic Surveillance

In 1995, SST, Inc. (previously ShotSpotter, Inc.) pioneered the concept of **Wide-Area Acoustic surveillance for gunfire** alert and analysis and today has deployed its technologies in over 70 cities worldwide, including cities such as Washington, D.C.; San Francisco, CA; Boston, MA; Chicago, IL; Rio de Janeiro, Brazil; Panama City, Panama; and many others.

Wide-area acoustic surveillance for gunshot detection involves the deployment of multiple, collaborative **acoustic sensors** throughout a coverage area to create a robust, redundant coverage array stretching from a single square mile up to 20 or more square miles. The sensors are paired with analysis **software** which identifies the unique signature of gunshots and other loud explosive sounds in real time.

When a gun is fired, the sound it makes (the muzzle blast) radiates outward at a known velocity (the speed of sound), and arrives at different ShotSpotter sensors at slightly different times, based on each sensor's distance from the origin of the sound. Using these slight differences in the time of arrival of the sound, the sensors trigger on each **impulsive noise** they hear, mark the precise time, and forward that information to the ShotSpotter server software. The server software pinpoints the precise location of each round fired by algorithms primarily relying on a combination of **multilateration** and **triangulation**. In contrast to point-protection gunshot detection systems, which rely on individual sensors to operate within a small radius (generally 100-200 meters around a single sensor) and which must have a clear line-of-sight the moment a gun is fired in order to provide useful information, SST's wide-area acoustic surveillance technology detects and precisely locates incidents that occur anywhere within a large outdoor coverage area extending many square miles in size, regardless of visibility and at distances as far as a mile or more distant from individual sensors. Thus a single gunfire incident is often heard by many sensors, some of them at the "right" time based on direct path transit of the sound and some of them at "wrong" times due to echoes, multipath, or other delays.

Subsequent to sound detection by the sensors, the technical superiority of the ShotSpotter approach becomes evident as the server software disambiguates the location from the many "true" (and some "not true," or echo/multipath) data points provided by the sensor network. Among many possible mathematical solutions, it finds the optimal one, eliminates errors, calculates the total number of rounds fired even when one or more rounds cannot be heard on several sensors (and the speed and direction of travel of a moving source, if applicable) and reports these data. Single-sensor, point solutions do not benefit either from the wide area, multi-sensor redundancy or from the opportunity to compare data on multiple sensors to filter out echoes, delays, or multipath. After the ShotSpotter server performs the **initial location** calculation, it turns to the individual pulses (shots) within the sequence of shots and optimizes its location to produce a **final location result**. Then it uses certain acoustic pulse characteristics (metadata) to make an **initial machine classification** of the incident, and optionally requests one or more sensors to begin to **download an audio recording** of the few seconds containing the incident back to the server. Note that, prior to this download request, the entire sequence of events (detection, location, optimization, and alert) is performed *without* access to the underlying raw audio stream at the sensor.

These initial alert data have not yet been reviewed and are therefore referred to as **raw alerts** or sometimes more technically as **unreviewed alerts**. Such alerts commonly include some cases of gunfire as well as cases of fireworks and other loud **impulsive noises** such as explosions. Generally speaking, these raw alerts are useful in the sense that they indicate that something loud and impulsive in acoustic nature or character originated at a particular geographic location at a particular time. In cases where targeting other sensors has relatively low cost, as for example nearby, pan-tilt-zoom- (PTZ-) controllable video cameras, *it is often desirable to trigger or **notify** external systems on the basis of these raw alerts in order to get "eyes on target" immediately and thus further the investigation*. Such notifications can be (but do not have to be) performed immediately and before gunfire is confirmed. Thereafter (or in parallel) a process of **incident review** begins in order to establish whether the incident in question is, indeed, gunfire.

1.2 Incident Review and Alert

The next step is thus incident **review**. For customers of SST's **ShotSpotter Flex**SM subscription based service, raw incident data are reported directly to the SST **Incident Review Center**TM (IRC) where our **Reviewed Alerts Service**SM provides immediate assessment and qualification of gunshot alerts by a highly trained team of SST gunfire and acoustics experts, 24x7x365. For customers of SST's **ShotSpotter OnSite** (capital equipment) products, the Reviewed Alert Service is available as an option, or customers can choose to review all the raw incidents themselves. Over time, most SST customers have chosen to avoid the expense and effort of training their own 24x7x365 staff and now rely on SST's Reviewed Alerts Service. SST incident reviewers have quite literally heard more gunfire incidents, from a wider variety of distances and acoustic environments, than anyone else in the world. When their work is complete, the incident may or may not have been **reclassified**—i.e., determined to be of a different type (gunfire, fireworks, etc.) than the initial machine classification; this final state is referred to as the **final classification** or less commonly the **reviewed classification**.

Once incidents have been reviewed, the data—now called **reviewed gunfire alert** data—are ready for presentation to end-users either by SST's own ShotSpotter user interface software or by third-party software products which are **notified**. Our own user interface software which displays these data exists in several generations including the **PSC**TM ("Public Safety Console") generation for ShotSpotter OnSite systems and the cloud-based ShotSpotter **Alert Console**TM and **Incident & Reports Portal**TM software for subscription-based ShotSpotter Flex customers. Gunfire incidents are also permanently archived, along with forensic data and actual incident audio recordings, for subsequent analysis, review during investigation, presentation during criminal legal proceedings, and long-term crime trend analysis.

1.3 Reviewed versus Unreviewed ("Raw") Alerts

After review and subsequent customer alert, many customers seek to have ShotSpotter data automatically transmitted (**notified**) to external systems which had not been previously notified of the raw (unreviewed) alert. Thus there are two types of notifications:

1) Instantaneous notification of **raw (unreviewed) alerts** is appropriate for "low cost to respond" assets, such as PTZ cameras, for which the cost of turning in the direction of a sound is negligible.

2) By contrast, "high cost to respond" assets, such as police officers, or the creation of a CAD (Computer-Aided Dispatch) record, or the triggering of a Common Operating Picture (COP) system, are typically notified only of **reviewed alerts**.

There are several other differences between the two classes of notifications:

	Unreviewed ("Raw")	***Reviewed***
Classification	Machine only	Machine + Reviewer
Other Situational Context	None	Provided when available
Timing	< 1 second after location	SLA 90% within 60 seconds of receipt of data; average < 20 seconds
Notification Appropriate for	**Relatively low** cost to respond, fast response (< 10 seconds) actions: • PTZ Cameras • DVR timestamp/tagging • UAV look-at-target (camera slew) • Other Sensor triggering	**Relatively high** cost to respond, slower response (> 10 seconds) actions: • Officer or first responder dispatch • CAD record creation • Air (helicopter) asset mission • UAV fly-to-target
Typical Daily Volumes	Hundreds	Tens
Peak Volumes (holidays)	Thousands	Hundreds
Updates Sent by ShotSpotter	Review	None
Supported in Notification Engine version	2.0+	2.6+

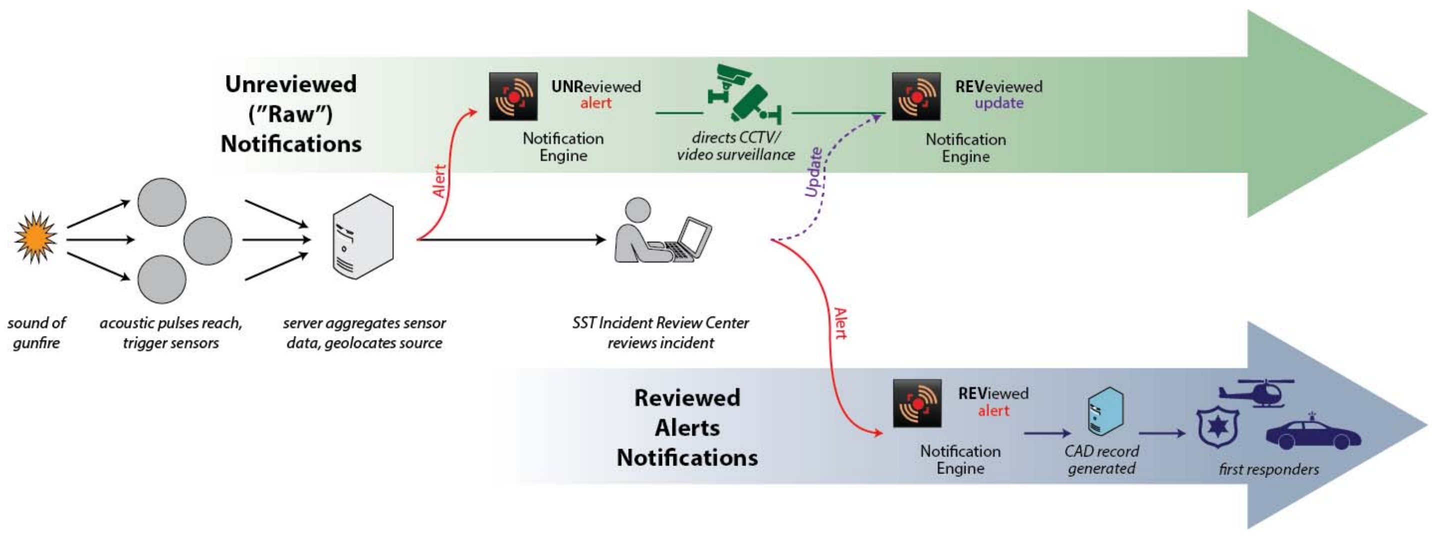
Recipients of unreviewed alerts can optionally also subscribe to **update** notifications which provide information about the subsequent review of an incident previously notified in its unreviewed state.

1.4 ShotSpotter Notification API

For the purposes of notification to external systems, it is important to remember that the wide area, distributed nature of a ShotSpotter gunfire alert and analysis solution means that more than one sensor will hear a given gunfire incident, but none of the sensors has sufficient information on its own to locate the incident. For a notification to be useful, therefore, it must come from the analysis of multiple sensors' data by the **ShotSpotter *server***, not from an individual sensor. Accordingly, the ShotSpotter **Notification API** is designed to notify external systems when the *server* reports a new incident. This is consistent with the way SST's own ShotSpotter user interface software works, but it can cause some confusion for those familiar with other sensor-based surveillance systems which send signals from the edges of the network.

The ShotSpotter Notification API™ can report new incidents or updated information about incidents previously reported. As its name suggests, the design of the Notification API focuses heavily on *notification* of new (or recently updated) incidents and not on making historical data available for analysis (see *Historical Data* below for more information about how to access historical ShotSpotter incident data).

Reviewed and unreviewed (raw) alerts follow slightly different notification workflows, as depicted in the following diagram.


Unreviewed ("Raw") Notifications
UNReviewed alert
Notification Engine
directs CCTV/ video surveillance
REViewed update
Notification Engine
Alert
Update
sound of gunfire
acoustic pulses reach, trigger sensors
server aggregates sensor data, geolocates source
SST Incident Review Center reviews incident
Alert
Reviewed Alerts Notifications
REViewed alert
Notification Engine
CAD record generated
first responders

1.5 About Notification Engine

The Notification Engine is fully documented in the *Notification Engine User's Manual*, a companion document to this *Reference*. Please refer to the *User's Manual* for detailed information.

Incident-by-incident notification is performed by a software component called the **ShotSpotter Notification Engine**. The Notification Engine runs on most modern Microsoft Windows operating systems and:

1. **polls** for new ShotSpotter incidents
2. **filters** new incidents by type (gunfire, firework, etc.) and workflow stage (unreviewed, reviewed)
3. **notifies** multiple subscribers of multiple types by sending one or more **message packets**
4. **receives** confirmation of receipt if desired and **retries** if configured to do so

The current version of the Notification Engine is 2.6. It delivers substantial upgrades over previous versions. In the table below, capabilities new since prior versions are highlighted in blue:

Incident Workflow Status	Unreviewed Alerts, Reviewed Alerts, Reviewed Updates to Unreviewed Alerts
Incident Types	Single Gunfire, Multiple Gunfire, Possible Gunfire, Firecracker/Fireworks, Backfires, Others
Packet Types	9 Generation I Packets 5 Generation II (Advanced) Packets
*Geospatial Boundary Awareness (**"geofences"**)*	Two-levels of customizable geographic boundaries reported for each incident
Street Address Lookup (Reverse Geocoding)	Parcel map, address point, or Bing maps web service

Operating System Support	32-bit Windows XP, Windows 7, Windows Server 2003, 2008 64-bit Windows 7, Windows Server 2008, Windows Server 2008 R2, Windows Server 2012 (beta)
ShotSpotter Product Support	ShotSpotter OnSite ShotSpotter Flex ShotSpotter SpecialOps ShotSpotter CIKR
Transport Channels	HTTP POST (XML payload) HTTP GET (query string) Socket Google Earth .KML Plugin (`.dll`)
Payload Encoding	ASCII stream HTTP query string (key=value) XML

1.6 Accessing Historical ShotSpotter Data

The ShotSpotter Notification API, and its implementation in the ShotSpotter Notification Engine, provides real-time incident alerts to multiple endpoints. They specifically *are not* designed to provide access to long-term (aggregated) data or statistics. Although the endpoints are free to keep their own historical records of incidents (*e.g.* as a CAD or COP system might), SST, Inc. also offers access to its historical data to customers via both its user interface software and via database APIs. The ShotSpotter Flex Incident & Reports Portal, for example, can export data meeting any of various search criteria to Microsoft Excel® or .CSV format. For additional information, please refer to *Application Note 101: Accessing ShotSpotter Data*, available from SST, Inc.

2 Packet Formats: Two Generations

With Version 2.6 of the ShotSpotter Notification API and Notification Engine, SST has introduced a new family of message packets designed for both the ShotSpotter OnSite (capital equipment) and ShotSpotter Flex (subscription) products and business models. We refer to this new family of message packets as **Generation II message packets**. Message packets supported by earlier API versions are referred to as **Generation I message packets**.

2.1 Important: Use Generation II Packets for All New Projects

Although Version 2.6 of the Notification API continues to support the Generation I packets originally supported by earlier versions (to the extent technically possible), changes to both the core ShotSpotter product and improvements to the depth and content of the API will *not* be mirrored in the Generation I packets, and future support for Generation I packets is not guaranteed. All new development should therefore be performed exclusively with the new, Generation II packets. The Generation II packets have intentionally been designed to be backwards-compatible with the Generation I packets, and thus any revisions to existing projects should make the effort to switch from Generation I to Generation II. The change should require minimal effort, because other than the content of packet prefix opcode itself, there exists a Generation II packet with all the same fields as (and sometimes more than) their Generation I counterparts. Properly designed interfaces should have minimal difficulty changing from one to the other after making a simple prefix opcode change.

2.2 Inventory of Packet Types

2.2.1 Notification & Response

	Generation II (Fully Supported)	*Generation I (Deprecated)*
Simple Alert	`IALRT01`	`INCAUPD`
Extended Alert	`IALRT02`	`INCAUPE` `INDAUPE`
Extended Alert with Multimedia (audio) URLs	`IALRT03`	n/a
Incident Update	`IUPDT02` `IUPDT03`	`INCIUPD`
Response Packet	*scheduled for future release*	`INCARSP`

2.2.2 Telemetry

Telemetry functionality has not been updated for Generation II. Due to the 24x7x365 monitoring and support capabilities provided by SST to its customers, API-based export of sensor telemetry is no longer a requirement and is not scheduled for support or maintenance in the future.

	Generation II (Fully Supported)	*Generation I (Deprecated)*
Sensor Status		`WGSIUPD`
Sensor Update	*no longer supported*	`WGSAUPD`
Sensor Pair		`WGSPAIR`

3 Generation II (Fully-Supported) Packet Formats

The Generation II packet formats provide incident **notification** (**alert**) via the `IALRT0`*x* group of packets, three of which are introduced with **API Version 2.6**. To accommodate Incident Review Center (or customer-premises) incident review workflow, a new `IUPDT0`*x* group of packets is introduced to permit subsequent update to incident information after initial (raw) alert. Developers have often asked for API-level access to ShotSpotter sensor audio (*e.g.* an .mp3 recording of the gunfire incident as heard at each sensor, plus one to two seconds of audio before and after). The new Generation II packet `IALRT03` and `IUPDT03` provide audio URLs to all available sensor audio and are structured to provide support in the future for other multimedia types, including for example aerial imagery of the incident location showing a "dot on the map." The packet types available are:

Purpose	*Message Packet Prefix (Opcode)*	*Generation I Equivalent*	*Purpose*
Simple Alert	`IALRT01`	`INCAUPD`	"Basic" alert with minimal information to pan-tilt-zoom devices, can accept a geo-referenced endpoint and will calculate relative range/bearing/elevation.
Extended Alert	`IALRT02`	`INCAUPE` `INDAUPE`	"Detailed" alert packet with detailed information regarding the incident, can accept a geo-referenced endpoint and will calculate relative range/bearing/elevation.
Extended Alert with Multimedia (audio) URLs	`IALRT03`	n/a	Extended "detailed" alert packet with detailed information regarding the incident and provides a variable number of associated URLs for incident audio, etc.
Incident Update	`IUPDT02`	`INCIUPD`	Update packet supporting Reviewed Alerts and other after-detection updates (classification change, etc.). Issued to subscribers of `IALRT02` messages.
	`IUPDT03`	n/a	Update packet supporting Reviewed Alerts and other after-detection updates (classification change, etc.). Issued to subscribers of `IALRT03` messages.

In the sections that follow, each packet, its opcode, and its fields and values are provided in detail. In most cases, the Generation II packet is designed as a superset of a Generation I packet. For convenience, sections which are new or modified from Generation I to Generation II are highlighted in blue. Note that incident IDs have been standardized in Generation II to 7 characters permitting 10 million incident IDs per system; they were previously variable length. (It is not uncommon to see 500,000 or more incidents on busy systems.) Distances in meters and headings in degrees are now standardized as 5 places before the decimal place and 2 places after.

3.1 `IALRT01` Packet

The `IALRT01` packet is intended for simple endpoints which cannot manage much data. It supports the same field types as the Generation I `INCAUPD` packet but now provides Incident Type explicitly as well as Incident Review Workflow Status.

IALRT01 Packet Format

Field	HTML/XML Field Name	Sample	Length/Notes
Packet Prefix[1]	root node `<IALRT01>`	`$IALRT01`	see note [1] below
Incident ID	`id`	`0001234`	7 characters
Incident Distance[2, 3]	`distance`	`00120.00`	8 characters, meters
Incident Heading[2, 3]	`heading`	`00315.68`	8 characters, decimal degrees
Incident Type	`incident-type`	`01`	see §3.6 Incident Type Codes
Incident Worflow Status	`incident-workflow-status`	`UNR`	see §3.7 Incident Review Workflow Status
Checksum indicator[4]	n/a	`*`	
Checksum	`checksum`	`0E`	see **Error! Not a valid result for table.**
Message ending	n/a	`\r\n`	

Notes:

1. The packet prefix begins with a `$` character in ASCII stream format (the default for the socket channel). In other channels, there is no `$` character and the prefix is simply the 7-character prefix.
2. These fields contain values relative to the latitude and longitude properties (i.e., the *position of the endpoint)* assigned to individual end-points using subscription properties in Notification Engine user interface provided for the camera or endpoint, as described in *§6.2 Subscription Properties – INCAUPE, INCAUPD, IALRT01, and IALRT02.* If no such position is provided, then the values of these fields are undefined.
3. Incident distance and heading are calculated based on the position of the endpoint as described above using two-dimensional trigonometric calculations. If a 3D (pan/azimuth angle, range, heading/elevation angle) solution is required, use the `IALRT02` packet.
4. The checksum indicator will appear only in the ASCII stream format (the default for the socket channel). See Appendix II: Checksum Calculation Algorithm.

3.2 `IALRT02` Packet

The `IALRT02` packet is intended for more capable endpoints which want full incident data but do not intend to reference multimedia resources available by URL request. It supports the same field types as the Generation I `INCAUPE` and `INDAUPE` packet, but now provides Incident Type explicitly, Incident Review Workflow Status, and information regarding the location of the event with regard to geographic boundary(ies) specified for the two different geographic boundary (geofence) layers on the ShotSpotter server, commonly referred to as "beat" layer and "jurisdiction" layer (but which can be used for any similar geofencing purpose).

Like the Generation I packets `INCAUPE` and `INDAUPE`, the `IALRT02` packet supports a heading "offset" and an elevation "offset." These offsets are useful when cameras are not installed with their internal zero-degree direction pointing precisely North, or when they are not positioned such that zero-degree elevation is precisely parallel to the ground. These and other settings are configured in the subscription-specific settings for a given end-point (see *§6.2 Subscription Properties – INCAUPE, INCAUPD, IALRT01, and IALRT02* and *§6.3Subscription Properties – INCAUPE and IALRT02 only*).

IALRT02 Packet Format			
Field	*HTML/XML Field Name*	*Sample*	*Length/Notes*
Packet Prefix[1]	*root node* `<IALRT02>`	`$IALRT02`	*see note [1] below*
Incident ID	`id`	`0001234`	7 characters
Latitude	`latitude`	`+01.123456`	*see note [2] below*
Longitude	`longitude`	`+012.123456`	
Incident [Street] Address	`address`	`303 Second Street`	reverse geocoded address[3]
Additional Description	`description`	`7 rounds, shooter moving SE at 5mph`	*see note [4] below*
Fine-grained geofence	`beat`	`111AAA`	*see note [5] below*
Coarse-grained geofence	`district`	`222BBB`	
Source Name	`source`	`ABCity`	the name of the ShotSpotter Database which corresponds to the coverage area
Incident Distance[6,7]	`distance`	`00120.00`	8 characters, meters
Incident Heading[6,7]	`heading`	`00315.68`	8 characters, decimal degrees
Heading Adjusted[6,7]	`heading-adjusted`	`00300.68`	8 characters, decimal degrees
Elevation[6,7]	`elevation`	`015.0`	5 characters, decimal degrees
Elevation Adjusted[6,7]	`elevation-adjusted`	`010.0`	5 characters, decimal degrees
Zoom	`zoom`	`01.00`	4 characters, reserved for future use
Incident Time	`time`	`2012-10-16 00:00:27`	Local database time (NOT UTC)
GMT (UTC) Offset	`gmt-offset`	`-08`	3 characters, time zone offset from GMT
Incident Type	`incident-type`	`01`	see §*3.6 Incident Type Codes*
Incident Worflow Status	`incident-workflow-status`	`UNR`	see §*3.7 Incident Review Workflow Status*
Checksum indicator	*n/a*	`*`	
Checksum	`checksum`	`0E`	see ***Error! Not a valid result for table.***
Message ending	*n/a*	`\r\n`	

Notes:

1. The packet prefix begins with a `$` character in ASCII stream format (the default for the socket channel). In other channels, there is no `$` character and the field contains just the 7-character prefix.
2. Please remember that the latitude/longitude (and other geographic coordinates) provided represent *the location of the incident itself*, and explicitly *not* the coordinates of the sensors reporting the incident. This has been the source of confusion for some integrators more accustomed to sensor-focused surveillance mechanisms.
3. For ShotSpotter OnSite systems, reverse geocoding is performed against a customer-provided parcel map or address point map. For ShotSpotter Flex systems, reverse geocoding is attempted first against the customer-provided parcel map or address point map, if it exists and provides a valid answer, and then subsequently as a fallback mechanism against the Microsoft Bing Maps reverse geocoding web service. If the customer parcel or address point map exists but a suitable value is not found during the lookup, the Flex server *will* attempt to reverse geocode the coordinates using Microsoft Bing Maps.
4. For ShotSpotter Flex customers, the Additional Description field may contain additional information and situational context provided by the SST Incident Review Center. Such additional information commonly includes number of rounds fired (in cases of multiple gunfire) and the speed and direction of travel, if the server was able to detect movement among multiple gunshots.
5. Fine-grained and coarse-grained **geofences**, often used for "beats" and "districts" respectively, can be set up on the ShotSpotter server to permit customers to resolve incidents within fine-grained or coarse-grained geographic boundaries within their coverage area. The terms "beat" and "district" are arbitrary, but provide some insight into the uses to which these two levels of boundaries (geofences) are commonly put.
6. These fields contain values relative to the latitude, longitude and height properties (i.e., the *position of the endpoint)* assigned to individual end-points using subscription properties in Notification Engine user interface provided for the camera or endpoint, as described in *§6.2 Subscription Properties – INCAUPE, INCAUPD, IALRT01, and IALRT02* and in *§6.3 Subscription Properties – INCAUPE and IALRT02 only*.
   If no such position is provided, then the values of these fields are undefined.
7. Incident distance, heading, and elevation angle are calculated using three-dimensional trigonometric calculations and based on the physical position of the endpoint. If your application requires only a 2D (pan/azimuth angle, range, without elevation angle) solution, you should subscribe to the `IALRT01` packet instead.

3.3 IUPDT02 Packet

The IUPDT02 packet exists to provide incident review workflow updates for subscribers of IALRT02 packets. It duplicates most of the fields in the IALRT02 packet and introduces a new Final Incident Type field, identifying the **final classification** of an incident (i.e., the classification after the SST Incident Review Center or customer has reviewed the incident audio). It also provides an updated workflow status.

IUPDT02 Packet Format			
Field	*HTML/XML Field Name*	*Sample*	*Length/Notes*
Packet Prefix	root node <IUPDT02>	$IUPDT02	see note [1] for IALRT02 above
Incident ID Latitude Longitude Incident [Street] Address Additional Description Fine-grained geofence Coarse-grained geofence Source Name Incident Distance Incident Heading Heading Adjusted Elevation Elevation Adjusted Zoom	see IALRT02 above		
Original Incident Type	original-incident-type	01	see §3.6 Incident Type Codes
Final [Reviewed] Incident Type	final-incident-type	03	
Incident Worflow Status	incident-workflow-status	REV	see §3.7 Incident Review Workflow Status
Checksum indicator Checksum Message ending	see IALRT02 above		

3.4 IALRT03 Packet

The IALRT03 packet marks a departure from Generation I packets and from their IALRT01 and IALRT02 Generation II counterparts. The IALRT03 packet delivers nearly all the incident data available for a given ShotSpotter incident and would be appropriate for use in **Common Operating Picture (COP)** or Command and **Control (C2)** systems which seek to aggregate sensor data from multiple platforms. In addition to the data provided within the packet, it will contain one or more **URI**s (**uniform resource identifiers**[1]) pointing to additional multimedia content available relating to the incident. In version 2.6 of the Notification Engine, URLs for all sensor audio automatically downloaded are provided. Future versions of the ShotSpotter platform may provide additional resources and add additional URIs. Each URI includes **metadata** describing the media available at the URI (for example, `Sensor 10, 130 meters`) and the **MIME content type** (for example, `audio/mpeg`).

Because of its intended use primarily as a system-to-system connector packet, and to save room, the `IALRT03` packet does not include the relative positioning information (heading, range, elevation) which is provided in IALRT02. Relative position makes semantic sense in the context of individual end-points, such as cameras, but makes no sense in the context of other *systems* (*e.g.*: what is the "distance" from a CAD system to a gunshot?). You are not restricted, however, from subscribing to both packet types, but we can imagine few circumstances under which doing so would be appropriate.

3.4.1 Variable-Length Packet

Unlike other packets, the `IALRT03` packet has a variable number of fields. If there are *n* URIs included in the packet, there will be (14+*3n)* fields not including the checksum indicator or the checksum itself. For example, if a given incident has four (4) sensor audio files available, then the total number of fields in the `IALRT03` packet reporting it will be (14+3×4) = 26. Similarly, an incident with 10 sensor audio files available will have (14+3×10) = 44 fields.

3.4.2 Availability of Audio Files

The process of downloading audio files from the ShotSpotter sensor network is handled by the ShotSpotter server software *entirely after the incident has been located*. In order to ensure that alerts are published as quickly as possible, an unreviewed alert will be published as soon as the location has been calculated but very often *before the audio is available*. Accordingly, your application must make allowances for additional audio download time after the incident has taken place. By the time a *reviewed* alert is published, it is nearly always the case that at least one audio file is available (on which the SST Incident Review Center has, in part, based its acoustic analysis). However, in general, external applications should assume that URLs *will be valid* at some time after an incident is alerted, but 1) almost certainly *will not* be available immediately for unreviewed alerts, and 2) *may not* all be available immediately for reviewed alerts. As SST configures the web servers providing audio, they will return HTTP standard response code 404 before audio is available, and 200 (along with the audio data) when it is available. Thus a simple HTTP polling loop will suffice to deliver audio when it becomes available. We recommend you stop polling at some point after alert, as it is sometimes the case that audio is expected to be available but, for network connectivity or congestion reasons, cannot be downloaded.

[1] "URI" is the term describing a superset which includes the more familiar **URL** (**uniform resource locator**) as well as **URN** (**uniform resource name**).

3.4.3 IALRT03 Packet Format Specification

IALRT03 Packet Format

Field	HTML/XML Field Name	Sample	Length/Notes
Packet Prefix[1]	root node `<IALRT03>`	`$IALRT03`	see note [1] below
Incident ID	`id`	`0001234`	7 characters
Latitude	`latitude`	`+01.123456`	see note [2] below
Longitude	`longitude`	`+012.123456`	
Incident [Street] Address	`address`	`303 Second Street`	reverse geocoded address[3]
Additional Description	`description`	`7 rounds, shooter moving SE at 5mph`	see note [4] below
Fine-grained geofence	`beat`	`111AAA`	see note [5] below
Coarse-grained geofence	`district`	`222BBB`	
Source Name	`source`	`ABCity`	the name of the ShotSpotter Database which corresponds to the coverage area
Incident Time	`time`	`2012-10-16 00:00:27`	Local database time (NOT UTC)
GMT (UTC) Offset	`gmt-offset`	`-08`	3 characters, time zone offset from GMT
Incident Type	`incident-type`	`01`	see §3.6 Incident Type Codes
Incident Worflow Status	`incident-workflow-status`	`UNR`	see §3.7 Incident Review Workflow Status
URI Count	`uri-count`	`05`	2 digits, total number of URIs (URLs) following
	`<uris>` `<uri-detail>`		
URI #1 MIME Content-Type	`mime`	`audio/mpeg`	the IANA MIME media type available at this URI
URL #1	`url`	`https://us1.shotspotter.net/City/CityAudio/2012-08/2012-08-05/City0312_5_(2540-City___5).mp3`	see note [5] below
URI #1 Metadata	`description`	`Sensor 5, 50 meters`	
…		…	…
URI #n MIME Content-Type	`mime`	`audio/mpeg`	
URL #n	`url`	`https://us1.shotspotter.net/City/CityAudio/2012-08/2012-08-05/City0312_7_(2540-City___7).mp3`	see note [6] below
URI #n Metadata	`description`	`Sensor 27, 250 meters`	
	`</uri-detail>` `</uris>`		
Checksum indicator	n/a	`*`	
Checksum	`checksum`	`0E`	see Error! Not a valid result for table.
Message ending	n/a	`\r\n`	

Notes:

1. The packet prefix begins with a `$` character in ASCII stream format (the default for the socket channel). In other channels, there is no `$` character and the field contains just the 7-character prefix.
2. Please remember that the latitude/longitude (and other geographic coordinates) provided represent *the location of the incident itself*, and explicitly *not* the coordinates of the sensors reporting the incident. This has been the source of confusion for some integrators more accustomed to sensor-focused surveillance mechanisms.
3. For ShotSpotter OnSite systems, reverse geocoding is performed against a customer-provided parcel map or address point map. For ShotSpotter Flex systems, reverse geocoding is performed first against the customer-provided parcel map or address point map, if it exists and provides a valid answer, and then subsequently as a fallback mechanism against the Microsoft Bing Maps reverse geocoding web service. If the customer parcel or address point map exists but a suitable value is not found during the lookup, the Flex server *will* attempt to reverse geocode the coordinates using Microsoft Bing Maps.
4. For ShotSpotter Flex customers, the Additional Description field may contain additional information and situational context provided by the SST Incident Review Center. Such additional information commonly includes number of rounds fired (in cases of multiple gunfire) and the speed and direction of travel, if the server was able to detect movement among multiple gunshots.
5. Fine-grained and coarse-grained **geofences**, often used for "beats" and "districts" respectively, can be set up on the ShotSpotter server to permit customers to resolve incidents within fine-grained or coarse-grained geographic boundaries within their coverage area. The terms "beat" and "district" are arbitrary, but provide some insight into the uses to which these two levels of boundaries (geofences) are commonly put.
6. Internally, the audio URL is constructed from a base URL component (which includes the protocol indicator, FQDN, and root path) and a variable URL component which changes on an incident-by-incident basis. The variable URL component contains a date hierarchy as well as a sensor and incident-specific name. The fixed and variable components of the URL examples above are shown here:

Base URL component	*Variable component*
`https://us1.shotspotter.net/City/CityAudio/`	`2012-08/2012-08-05/City0312_5_(2540-City___5).mp3`
	`2012-08/2012-08-05/City0312_27_(2540-City___27).mp3`

For flexibility, the Notification Engine allows you to configure a different base URL than is used internally by ShotSpotter software. The base URL is configured as a subscription-specific property. See Subscription Properties – IALRT03 only.

3.5 `IUPDT03` Packet

Like `IALRT02`, `IALRT03` has a companion update packet, `IUPDT03`, which exists to provide incident review workflow updates for subscribers of `IALRT03` packets. It duplicates the fields in the `IALRT03` packet and introduces a new Final Incident Type field, identifying the **final**

classification of an incident (i.e., the classification after the SST Incident Review Center or customer has reviewed the incident audio). It also provides an updated workflow status.

`IUPDT03` **Packet Format**

Field	*HTML/XML Field Name*	*Sample*	*Length/Notes*
Packet Prefix	*root node* `<IUPDT03>`	`$IUPDT03`	
Incident ID	*see IALRT03 Packet above*		
Latitude			
Longitude			
Incident [Street] Address			
Additional Description			
Fine-grained geofence			
Coarse-grained geofence			
Source Name			
Incident Time			
GMT (UTC) Offset			
Original Incident Type	`original-incident-type`	`01`	see §*3.6 Incident Type Codes*
Final [Reviewed] Incident Type	`final-incident-type`	`03`	
Incident Workflow Status	`incident-workflow-status`	`REV`	see §*3.7 Incident Review Workflow Status*
URI Count	*see IALRT03 Packet above*		
URI #1 MIME Content-Type			
URI #1			
URI #1 Metadata			
…			
URI #n MIME Content-Type			
URI #n			
URI #n Metadata			
Checksum indicator			
Checksum			
Message ending			

3.6 Incident Type Codes

Two-digit incident type codes are now provided for all Generation II packets. Incidents may be gunfire, fireworks (firecrackers), or other loud, impulsive noise. Depending on the workflow status, incidents type may be a result of machine classification or human review. The following list of incident types defines the entire "type space" of incidents, but it is important to note that neither automatic machine classification nor human review will necessarily produce all of these incident types, nor does SST offer its product for the detection and classification of acoustic events other than gunfire. The incident types are provided for data analysis and ease of traceability only. Do *not* rely on incident types other than gunfire being consistently produced.

Incident Type Code	*Description*	*Gunfire?*	*Canonical "Dot" Color in ShotSpotter User Interface Software*
0	Unclassified		blue
1	Single Gunshot	Yes	red
2	Multiple Gunshots	Yes	red
3	Firecracker		yellow *or* white
4	Bottle Rocket		yellow *or* white
5	Aircraft		purple
6	Other		lime
7	System Test		pale green
8	Backfire		orange
9	Helicopter		purple
10	Motorcycle		purple
11	Construction		blue
12	Sonic Boom		blue
13	Transformer		lime
14	Explosion		lime
15	Thunder		fuchsia
16	Rain		fuchsia
17	Firing Test	Yes	pale green
18	Simulation		pale green
19	Possible Gunshot	Yes	half red/half yellow *or* half red/half white
20	Anticipated Gunshot	Yes	orange
21	Friendly Force Gunshot	Yes	blue

3.7 Incident Review Workflow Status

Incident alerts and incident updates now provide status information regarding the stage (progress along) the review workflow. Referring back to the figure in *§1.5 ShotSpotter Notification API*, the status codes are three letter values defining whether an incident is **UNR**eview, **REV**iewed, or has been **P**ublished automatically due to a **T**ime-**O**ut. Additional values are reserved for future use in the API but are not currently output by the Notification Engine.

Review Workflow Status	*Description*
UNR	Unreviewed
REV	Reviewed
PTO	Published due to timeout (rare)
SPT	
CHT	*reserved for future use*
CMT	

REP
RCL
RLT
FRP
OTH

4 Generation I (Legacy) Packet Formats [Deprecated]

Version 2.6 of the ShotSpotter Notification API continues support for the Generation I (Legacy) packet formats introduced in the API versions which shipped with Notification Engine 1.0 and 2.*x* versions prior to 2.6.

4.1 Important: Use Generation II Packets for All New Projects; Transition Away from Generation I on Existing Projects in Your Next Rebuild

Although Version 2.6 of the Notification API continues to support the Generation I packets originally supported by earlier versions (to the extent technically possible), changes to both the core ShotSpotter product and improvements to the depth and content of the API will *not* be mirrored in the Generation I packets, and future support for Generation I packets is not guaranteed. **For new projects, use only Generation II. For existing projects, you should switch to Generation II packets as soon as possible—ideally in your next project rebuild.** Several Generation II packets have intentionally been designed to be backwards-compatible with the Generation I packets, and thus any revisions to existing projects should make the minimal effort to switch from Generation I to Generation II.

4.2 `INCAUPD` Packet [Deprecated]

This is a legacy Generation I packet type. Its use is supported but deprecated in Notification API versions 2.6 and later. The Generation II packet `IALRT01` can be used as a replacement for this packet with minimal required code changes.

Component	***HTML/XML Field Name***	***Sample***
Packet Prefix	`prefix`	`$INCAUPD`
Incident ID	`id`	`1234`
Incident Distance (in meters)	`distance`	`120.00`
Incident Heading (in degrees)	`heading`	`315.68`
Checksum indicator		`*`
Checksum	`checksum`	`0E`
Message ending		`\r\n`

4.3 `INCAUPE` Packet [Deprecated]

This is a legacy Generation I packet type. Its use is supported but deprecated in Notification API versions 2.6 and later. The Generation II packet `IALRT02` can be used as a replacement for this packet with minimal required code changes.

Component	*HTML/XML Field Name*	*Sample*
Packet Prefix	`prefix`	`$INCAUPE`
Incident ID	`id`	`1234`
Incident Distance (in meters)	`distance`	`120.00`
Incident Heading (in degrees)	`heading`	`315.68`
Heading Adjusted	`heading-adjusted`	`300.68`
Elevation (in degrees)	`elevation`	`15.0`
Elevation Adjusted	`elevation-adjusted`	`10.0`
Zoom[(1)]	`zoom`	`1`
Incident Time	`time`	`2012-10-16 12:00:27`
Checksum indicator		`*`
Checksum	`checksum`	`0E`
Message ending		`\r\n`

Notes:
1. Zoom is reserved for future use

4.4 `INCIUPD` Packet [Deprecated]

This is a legacy Generation I packet type. Its use is supported but deprecated in Notification API versions 2.6 and later. The Generation II packet `IUPDT02` can be used as a replacement for this packet with minimal required code changes.

Component	*HTML/XML Field Name*	*Sample*
Packet Prefix	`prefix`	`$INCIUPD`
Incident ID	`id`	`1234`
Incident Latitude	`latitude`	`37.1234`
Incident Longitude	`longitude`	`120.1234`
Incident Type(1)	`type`	`1`
Incident Address	`address`	`303 Second Street`
Incident Location X(2)	`location-x`	`2234556`
Incident Location Y(2)	`location-y`	`4559696`
Incident Time	`time`	`2012-10-16 12:00:27`
Checksum indicator		`*`
Checksum	`checksum`	`0E`
Message ending		`\r\n`

Notes:
1. *see §3.6 Incident Type Codes*
2. (X, Y) coordinates represents the incident location in the local UTM geographic coordinate system. UTM zone must be inferred from system location. A convenient UTM lookup tool is provided by APSalin at www.apsalin.com/utm-zone-finder.aspx.

4.5 `WGSAUPD` Packet [Deprecated]

This is a legacy Generation I packet type. Its use is supported but deprecated in Notification API versions 2.6 and later. See §2.2.2 Telemetry for information regarding future support of telemetry information.

Component	*HTML/XML Field Name*	*Sample*
Packet Prefix	`prefix`	`$WGSAUPD`
Sensor ID	`id`	`1234`
Sensor Distance (in meters)	`distance`	`120.00`
Sensor Heading (in degrees)	`heading`	`315.68`
Heading Adjusted	`heading-adjusted`	`300.68`
Elevation (in degrees)	`elevation`	`15.0`
Elevation Adjusted	`elevation-adjusted`	`10.0`
Checksum indicator		`*`
Checksum	`checksum`	`0E`
Message ending		`\r\n`

Notes:
1. Zoom is reserved for future use

4.6 **`WGSIUPD` packet [Deprecated]**

This is a legacy Generation I packet type. Its use is supported but deprecated in Notification API versions 2.6 and later. See *§2.2.2 Telemetry* for information regarding future support of telemetry information.

Component	***HTML/XML Field Name***	***Sample***
Packet Prefix	`prefix`	`$WGSIUPD`
Sensor ID	`id`	`1234`
Sensor Friendly Number	`friendly-number`	`1234`
Sensor Friendly Name	`friendly-name`	`Sensor A`
Sensor Type	`type`	`1`
Sensor Latitude	`latitude`	`33.7890`
Sensor Longitude	`longitude`	`120.2424`
Sensor Status(1)	`snr`	`1`
Checksum indicator		`*`
Checksum	`checksum`	`0E`
Message ending		`\r\n`

Notes:
1. Values for the Status field:

`0`	OK
`1`	Offline
`2`	Down
`3`	Unknown

4.7 **`WGSPAIR` packet [Deprecated]**

This is a legacy Generation I packet type. Its use is supported but deprecated in Notification API versions 2.6 and later. See *§2.2.2 Telemetry* for information regarding future support of telemetry information.

Component	***HTML/XML Field Name***	***Sample***
Packet Prefix	`prefix`	`$WGSPAIR`
Sensor ID	`id`	`1234`
Sensor Latitude	`latitude`	`33.7890`
Sensor Longitude	`longitude`	`120.2424`
Sensor Battery Level	`battery`	`20.123`
Sensor PDOP	`pdop`	`0.123`
Sensor Bearing	`bearing`	`20.123`
Sensor SNR(1)	`snr`	`0`
Checksum indicator	`checksum`	`*`
Checksum	`prefix`	`0E`
Message ending	`id`	`\r\n`

Notes:

1. Values for the SNR field will be reported as zero (`0`) in this version.

4.8 **Response packet: `INCARSP` [Deprecated]**

In response to `INCAUPD` and `INCAUPE` packets, you may issue an `INCARSP` packet, described below, back to the server. (For details on configuring inbound receipt of packets, please see the Users Manual.)

Component	***HTML/XML Field Name***	***Sample***
Packet Prefix	`prefix`	`$INCARSP`
Incident ID	`id`	`1234`
Acceptance flag (A – Accepted, R – Rejected)	`flag`	`A`
Checksum indicator	*n/a*	`*`
Checksum	`checksum`	`45`
Message ending	*n/a*	`\r\n`

5 Delivery Channels and Formats

The Notification Engine supports various **channels** through which packets can be transmitted. The packet contents do not change, but the formatting and delimiting of the packet fields does change slightly.

5.1 HTTP POST (XML payload) Channel

The preferred delivery format for sending packets is XML, as it permits full fidelity and is easy to inspect visually. XML is the default payload format for the HTTP POST channel, which transmits the data packet in the format of a regular HTTP POST request method. Here is a sample of an `INCIUPD` packet formatted as an XML payload:

```
<INCIUPD>
   <id>1053</id>
   <latitude>38.85962</latitude>
   <longitude>-76.98861</longitude>
   <type>1</type>
   <address>303 Second Street </address>
   <location-x>0</location-x>
   <location-y>0</location-y>
   <time>2012-10-18 14:00:36</time>
</INCIUPD>
```

Note that, unlike the ASCII stream formatting for socket channel packets, there are no start and checksum delimiting characters (`$` and `*`, respectively).

5.2 HTTP GET (query string) Channel

In the HTTP GET (query string) channel, the data packet is in the format of a regular HTTP *request* (GET) string, with the field values provided as query string parameters. All field names are in lowercases. Here is a sample of an `INCAUPD` packet formatted as a query string:

```
?prefix=INCAUPD&id=1234&distance=120.00&checksum=0E
```

The above packet might be sent to a listening URL as a standard HTTP request, for example:

```
http://192.168.0.1/somepage.jsp?prefix=INCAUPD&id=1234&dista
nc=120.00&heading=315.68&checksum=0E
```

Note that, unlike the ASCII stream formatting for socket channel packets, there are no start and checksum delimiting characters (`$` and `*`, respectively). Instead, the commonly-used `key=value` query string format is relied upon to delimit contents.

5.3 Socket (TCP) Channel

In the socket channel, packet data is sent as ASCII streams. Each data packet starts with a `$` (dollar sign) character and the 7-character packet prefix, and ends with two characters `\r\n` (carriage return followed by line feed, ASCII values 13 and 10 respectively). A packet body will usually contain multiple fields, separated by comma (`,`). The last three characters of a packet body is an asterisk (`*`) sign and a two-character checksum. The following is a sample of a Generation I `INCAUPD` packet as it would be transmitted via the socket channel:

```
$INCAUPD,1234,120.00,315.68*0E\r\n
```

and here is a sample Generation II `IALRT02` packet:

```
$IALRT01,0001234, +37.538867,-122.064656,303 Second Street,7
rounds shooter moving SE at 5mph,111AAA,222BBB,ABCity,
00120.00, 315.68,300.68,015.0,010.0,01.00,2012-10-16
12:00:27,-08,01,UNR*0E\r\n
```

5.4 Google Earth

Using the **Google Earth** device type and "GoogleEarth" protocol permits users to generate (and update) a Google Earth overlay file (.kml) so that incidents can be displayed in Google Earth environment when they are available. A specific `GoogleEarth_INCIUPD` pre-defined format has been created for this purpose. For additional information see the Google Earth predefined section below and the *Notification Engine User's Manual*.

5.5 Plug-In Channel

The packets are kept in their original XML format when the **plug-in** channel is used. Each packet has its packet type as its root element and support fields as direct children of the root. For example, `INCIUPD`, in its original format, looks like:

```
<INCIUPD>
   <id>1053</id>
   <latitude>38.85962</latitude>
   <longitude>-76.98861</longitude>
   <type>1</type>
   <address>303 Second Street </address>
   <location-x>0</location-x>
   <location-y>0</location-y>
   <time>2012-10-18 14:00:36</time>
</INCIUPD>
```

For information regarding how to develop Notification Engine plug-in `.DLL`s, see Appendix III: Plug-In Architecture.

5.6 Custom formats

Notification Engine provides a default format for each of the message types and channels it supports. However, you can override the default format by authoring a custom XSLT style sheet. The original incident data from Notification Engine is in XML format. Notification use pre-defined XSLT style sheets to translate the XML document to specific format for the delivery channel. If a custom XSLT style sheet is defined, Notification Engine will use customized style sheet instead of the default style sheet.

The following XML is an example of default packet format. Each packet is an XML segment with the packet prefix opcode as root node, and all supported fields as direct children of the root node.

Default `INCAUPE` *message - in original XML format*

```
<INCAUPE>
```

```
        <id>Incident Id (integer)</id>
        <distance>Incident distance from device (number)</distance>
        <heading>Calcuated device heading (degrees from North)</heading>
        <heading-adjusted>Adjusted device heading (heading - device default heading)</heading-adjusted>
        <elevation>Calcuated device elevation (degrees from Horizon)</elevation>
        <elevation-adjusted>Adjusted device elevation (elevation - device default elevation)</elevation-
adjusted>
        <zoom>NOT SUPPORTED</zoom>
        <time>Incident time (yyyy-MM-dd HH:mm:ss)</time>
</INCAUPE>
```

The following XML shows how the packet looks like after XSLT transformation. The transformed message is also an XML segment with `<notifications>` as root element and two children `<Packet>` and `<Check>`. During delivery, only the text (or XML chunk) under `<Packet>` is delivered to destination devices. The `<Check>` element is required by system for checksum calculation.

So, when authoring your own custom XSLT, you should ensure the transformed message is in the same format as shown below.

Default `INCAUPE` *message - in default HTTP format*

```
<notifications>
        <Packet>?prefix=INCAUPE&id=Incident Id (integer)&distance=Incident distance from device
(number)&heading=Calcuated device heading (degrees from North)&heading-adjusted=Adjusted device heading
(heading - device default heading)&elevation=Calcuated device elevation (degrees from Horizon)&elevation-
adjusted=Adjusted device elevation (elevation - device default elevation)&zoom=NOT SUPPORTED&time=Incident
time (yyyy-MM-dd HH:mm:ss)&checksum=check sum of the message</Packet>
        <Check>all fields values concated as one single string</Check>
</notifications>
```

5.6.1 Sample of default XSLT – HTTP format

As you can see in above sections, Notification Engine requires that the default `INCAUPE` message be transformed into following format:

```
<notifications>
        <Packet>message text to be sent to devices.</Packet>
        <Check>string used to calculate checksum. The string should be composed by concating all values of all
fields (without names, equal signs, and delimiters)</Check>
</notifications>
```

To achieve this transformation, Notification uses the following style sheet (for its HTTP channel):

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:template match="/">
                <notifications>
                        <Packet>
                                <xsl:call-template name="Packet" />
                        </Packet>
                        <Check>
                                <xsl:call-template name="Check" />
                        </Check>
                </notifications>
        </xsl:template>
        <xsl:template match="INCAUPE" name="Packet">
                ?prefix=INCAUPE&id=
                <xsl:apply-templates select="INCAUPE/id" />
                &distance=
                <xsl:apply-templates select="INCAUPE/distance" />
                &heading=
                <xsl:apply-templates select="INCAUPE/heading" />
                &heading-adjusted=
                <xsl:apply-templates select="INCAUPE/heading-adjusted" /> &elevation=
                <xsl:apply-templates select="INCAUPE/elevation" />
                &elevation-adjusted=
                <xsl:apply-templates select="INCAUPE/elevation-adjusted" />
                &zoom=
                <xsl:apply-templates select="INCAUPE/zoom" />
                &time=
                <xsl:apply-templates select="INCAUPE/time" />
                &checksum=[CHK]
        </xsl:template>
        <xsl:template match="INCAUPE" name="Check">
                <xsl:apply-templates />
        </xsl:template>
</xsl:stylesheet>
```

Note that 1) the "Check" template is used to compose the checksum string. This template can be copied to your style sheet without any change. 2) The string`[CHK]` is a place holder for check sum. When the message is delivered, `[CHK]` will be replaced with the properly-calculated checksum.

5.6.2 Composing Your Own XSLT

You can use the above example as template to compose your own style sheet. If you only want to eliminate some fields to trim down the message, you can simply modify the "Packet" template and remove unwanted fields. However, in some other cases, advanced XSLT authoring skills are required. For instance, some devices only accept integer values, while some others only accept hex number formats. The following is an example of a custom XSLT style sheet. It translates the `INCAUPE` message to a 6-letter-long string, with the first 3 digits representing heading, a space, and the last 2 digits representing elevation. (This happens to be the XSLT used to generate the pre-defined `RMS_INCAUPE` format)

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:template match="/">
                <notifications>
                        <Packet>
                                <xsl:call-template name="Packet" />
                        </Packet>
                        <Check />
                </notifications>
        </xsl:template>
        <xsl:template match="INCAUPE" name="Packet">
                <xsl:apply-templates select="INCAUPE/heading" />
                &#160;
                <xsl:apply-templates select="INCAUPE/elevation" /> </xsl:template>
        <xsl:template match="INCAUPE/heading">
                <xsl:value-of select="format-number(.,"000")" />
        </xsl:template>
        <xsl:template match="INCAUPE/elevation">
                <xsl:value-of select="format-number(.,"00")" />
        </xsl:template>
</xsl:stylesheet>
```

Note that `&#160;` is the other (the other one is `[CHK]`) special token in Notification Engine. Notification Engine will ensure it gets converted to a valid space (ASCII 32) when the messages are sent.

5.6.3 An extended example

XSLT style sheets may get very complex and may require serious debugging. Try your style sheet outside Notification Engine first before you plug it into Notification Engine. The following example is the style sheet used to generate SONY_INCAUPE format:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:template match="/">
                <notifications>
                        <Packet>
                                <xsl:call-template name="Packet" />
                        </Packet>
                        <Check>
                                <xsl:call-template name="Check" />
                        </Check>
                </notifications>
        </xsl:template>
        <xsl:template match="INCAUPE" name="Packet">
                ptzf.cgi?AbsolutePanTilt=
                <xsl:call-template name="sony_from_360">
                        <xsl:with-param name="sony_value" select="INCAUPE/pan" />
                </xsl:call-template>
                ,
                <xsl:call-template name="sony_from_360">
                        <xsl:with-param name="sony_value" select="360-INCAUPE/tilt" />
                </xsl:call-template>
                ,24
        </xsl:template>
        <xsl:template match="INCAUPE" name="Check">
                <xsl:apply-templates />
        </xsl:template>
        <xsl:variable name="hex_digits" select="'0123456789ABCDEFX'" />
        <xsl:template name="sony_from_360">
                <xsl:param name="sony_value" />
                <xsl:choose>
                        <xsl:when test="$sony_value >=0 and $sony_value <=170">
                                <xsl:call-template name="dec_to_hex">
                                        <xsl:with-param name="value" select="floor($sony_value *
                                2267 div 170)" />
                                </xsl:call-template>
                        </xsl:when>
                        <xsl:when test="$sony_value >=170 and $sony_value <=180">
                                <xsl:call-template name="dec_to_hex">
                                        <xsl:with-param name="value" select="2267" />
                                </xsl:call-template>
                        </xsl:when>
                        <xsl:when test="$sony_value >180 and $sony_value <=190">
                                <xsl:call-template name="dec_to_hex">
                                        <xsl:with-param name="value" select="63269" />
                                </xsl:call-template>
                        </xsl:when>
                        <xsl:when test="$sony_value >190 and $sony_value <=360">
                                <xsl:call-template name="dec_to_hex">
                                        <xsl:with-param name="value" select="floor(($sony_value -
                                190) * 2267 div 170) + 63270" />
                                </xsl:call-template>
                        </xsl:when>
                        <xsl:otherwise>
                                <xsl:call-template name="dec_to_hex">
                                        <xsl:with-param name="value" select="0" />
                                </xsl:call-template>
                        </xsl:otherwise>
                </xsl:choose>
        </xsl:template>
        <xsl:template name="dec_to_hex">
                <xsl:param name="value" />
                        <xsl:if test="$value > 15">
                                <xsl:call-template name="dec_to_hex">
```

```
                                            <xsl:with-param name="value" select="floor($value div 16)"
                                    />
                                    </xsl:call-template>
                            </xsl:if>
                            <xsl:value-of select="substring($hex_digits, ($value mod 16)+1, 1)" />
            </xsl:template>
    </xsl:stylesheet>
```

5.7 Additional Pre-Defined Formats

In additional to the default formats, some additional formats are predefined in Notification Engine 2.6. To use these formats, enter the corresponding **Format Key** into the **Custom Format XSLT** field of your subscription. *(Note: make sure subscribed packets match the selected pre-defined format!* For instance, the `RMS_INCAUPE` format only applies to `INCAUPE` packet, as the format key makes clear.)

5.7.1 RMS_INCAUPE

This is a very short format with only 6 characters. The first 3 characters represent the heading angle (in whole degrees, from 000 to 359). The fourth character is a space (ASCII code 32). The last two characters represent elevation angle (in whole degrees, from 00 to 90).

Example: `312 46`

5.7.2 SONY_INCAUPE

This format is designed specifically for SONY SNC-RZ25N/P cameras. It generates a Pan/Tilt/Zoom/Focus command according to SONY's specification and sends it to the designated CGI program. The three values passed in the command are pan, tilt, and speed (`24` being the maximum speed).

Example: `ptzf.cgi?AbsolutePanTilt=0,0,24`

Thus, the address of your subscription should be the URL where the `ptzf.cgi` is located (with trailing \).

5.7.3 PDA_INCIUPD

This is a simplified `INCAUPD` format with only incident id, incident type, incident latitude and incident longitude included. It doesn't include a checksum.

Example: `$INCIUPD,1027,1,38.85962,-76.98861`

6 Subscription parameters for different packets

Different packets require different sets of parameters. These parameters need to be set correctly in your subscriptions in order for Notification Engine to populate the fields of the output packets correctly. Some protocols may support only one or several of all possible packets, and some specific hardware/software only works with specific device type/subscription combinations.

You can change parameters of subscriptions at any time. A subscription can be individually enabled or disabled as well. However, if the device is disabled, all its subscriptions are automatically disabled as well.

6.1 Subscription Properties - Common

Property Name	*Comment*
Enabled	Indicates if the subscription is enabled. If a subscription (or its associated device) is disabled, no message is sent.
Channel	Delivery protocol used for this delivery. HTTP channel means the messages will be sent as HTTP GET request string, Telnet channel means the message will be send as ASCII stream over a socket connection.
Address	Delivery address. For HTTP channels, this address should be the web page that accepts the request string. The final request string generated by Notification Engine will be of the format [Address]`?`[Message string]. The message string consists a series of name-value pairs, in the format of [Name]=[Value], separated by &. For example, in the case of `INCAUPE`, assuming the subscription is `http://192.168.0.2/index.htm`, the final request string looks like: `http://192.168.0.2/index.htm?prefix =INCAUPE&id=701&distance=3747239.58885&heading =73.39422&heading-adjusted=73.39422&elevation= 0.00005&elevation-adjusted=0.00005&zoom=1& time=2012-09-13 14:27:43&checksum=06`. For telnet, this address should be [IP address]:[Port number].
Max Retry	This value decides how many times the Notification Engine tries to re-deliver the message upon previous falure. Also, you can set up the "Max Delay" property, which decides the maximum time length Notification Engine keeps re-delivering. The message will be discarded upon expiration of either value
Retry Interval	The time delay (in seconds) between delivery attempts
Max Delay	The maximum time length Notification Engine should try to re-deliver the message. See "Max Retry"
Require Response	Indicate whether the Notification Engine should expect a response from the device. The response message has to be in the format of `INCARSP`, which is prefix=`INCARSP&id=`[Incident ID]`&flat=`[Response Flag]. The [Incident ID] is the incident id contained in the delivered message, and [Response Flag] can be A, I, or R. If the response flag is `"A"` (accepted) or `"I"` (Ignored), Notification Engine will consider the message delivered. If the response flag is `"R"` (rejected), Notification Engine will try to re-deliver the message till either Max Retry or Max Delay expires.
Custom Format XSLT	File name of the custom XSLT style sheet. The path of custom XSLT style sheets is defined in "Custom XSLT Location" field in the "Settings" tab. The complete path Notification uses to access the file is [Custom XSLT Location]\[Custom Format XSLT].

6.2 Subscription Properties – `INCAUPE`, `INCAUPD`, `IALRT01`, and `IALRT02`

Property Name	*Comment*
Device Latitude	Decimal Latitude of device. 6 digits after decimal point are required for accurate results.

Device Longitude	Decimal Longitude of device. 6 digits after decimal point are required for accurate results.
Device Working Range	Effective range of device. If the incident is detected out of specified range (in meters), the device won't get notified. Enter -1 for infinite range.
Flip Heading Angle	Indicates whether heading angle should be flipped (`0`=No, `1`=yes).
Flip Elevation Angle	Indicates whether elevation angle should be flipped (`0`=No, `1`=yes).

6.3 Subscription Properties – `INCAUPE` and `IALRT02` only

Property Name	***Comment***
Device Default Heading	The angle, in degrees, from North that the device faces at its initial idle state.
Device Default Elevation	The angle, in degrees, from horizon below that the device faces at its initial ideal state.
Device Minimum Zoom	Minimum zoom factor allowed by the device. This setting is **NOT** supported in current version.
Device Maximum Zoom	Maximum zoom factor allowed by the device. This setting is **NOT** supported in current version.
Device Mounted Height	The height of the device. In current version all incidents are considered to happen at ground level. This value is used to calculate the elevation angle of the device.

6.4 Subscription Properties – `IALRT03` only

Property Name	***Comment***
URL Root	The root URI to be **prepended** to all URIs in the packet. The URL is constructed from a base URL component (which includes the protocol indicator, fully qualified domain name (FQDN) of the host, and root path) and a variable URL component which changes on an incident-by-incident basis. This property sets the protocol base URL component and must provide the protocol indicator (`http://` or `https://`), FQDN of the host, and whatever root (invariant) path is required. For example, a base URL might be: `https://us1.shotspotter.net/City/CityAudio/`

6.5 Subscription Properties – `INDAUPE` (in addition to `INCAUPE`)

Property Name	***Comment***
Device Location Feed	This property replaces "Device Default Heading", "Device Latitude" and "Device Longitude" properties in `INCAUPE`. Instead of static values, device location and heading will be read from designated sensor specified in this property.
Send Only When Online	When this property is set to Yes (1), the packet is only sent when the sensor status is Online.

Send Only When Not Down	When this property is set to Yes (1), the packet is only sent when the sensor status is not Down.

Appendix I: Glossary

Alert Console	Software used by ShotSpotter Flex customers to learn about new incidents in near real-time. The Alert Console is delivered as an out-of-browser (OOB) Silverlight application.
API	Application Programming Interface, a protocol intended to be used as an interface by software components to communicate with each other.
C2	Command and Control
Channel	The delivery mechanism of a notification packet. Notification Engine 2.6 supports HTTP GET (query string), HTTP POST (XML payload), Socket, Google Earth, and customer plug-in channels.
Classification	The determination of the causative nature of an acoustic event, e.g. Multiple Gunshots, Fireworks, or Transformer Explosion.
Client Software	SST's own user interface software, available in two forms: the **PSC** (Public Safety Console) and the cloud-based Flex **Alert Console** for real-time notifications and **Incident & Reports Portal** for investigation, analysis and reporting.
COP	Common Operating Picture, a system designed to integrated data from disparate sensor, surveillance, or tracking systems and represent them in a unified view, commonly graphically displayed on a map.
Database	The historical archive of ShotSpotter incidents maintained by the ShotSpotter Server.
Device	An endpoint which receives one or more packet types to which it is subscribed. Device information is stored in the ShotSpotter data, along with subscription details. Devices can represent real-world recipients such as surveillance cameras or computers, or virtual entities such as server software endpoints, aggregators, etc.
Device Type	Device Types are pre-defined groups that into which you can put one or more Devices. Device Type does not affect how the notifications are delivered; it is an organization aid only. The pre-defined Device Types are Surveillance Camera, Computer, Google Earth, and PDA.
Final Classification	The classification (type) of an incident selected after a Reviewer has reviewed an incident

Final Location	The optimized incident location calculated in real-time by the ShotSpotter server software
Formats	The data presentation used by a given subscription. Certain channels are limited to certain formats. For example, HTTP GET will only provide data in the ASCII stream format. Formats available are Query String (key=value), XML, and ASCII stream.
Generation I	The first generation of Notification API packet types, introduced in 2006, and now formally deprecated.
Generation II	The current generation of Notification API packet types, introduced with Version 2.6, and intended for future support and expansion.
Impulsive noise	An impulse noise (adj. "impulsive noise") is one which is both sharp (goes from quiet to loud almost instantaneously) and commonly wide band in nature. The general category includes explosions, clicks, pops, and other such noises.
Incident	A sequence of one or more gunfire sounds (or other loud noises) detected by either a ShotSpotter OnSite or ShotSpotter Flex sensor network. An incident is commonly associated with the shots fired from a single weapon, but occasionally involves multiple weapons fired near each other within a few seconds. At times, an incident may involve a sufficient number of rounds or continue for a long enough period of time that multiple incidents will be generated, each covering a different portion of the shooting.
Incident & Reports Portal	Software used by ShotSpotter Flex customers to search for prior incidents, investigate individual incident timelines or assess trends and patterns. Search results can be exported for analysis in third-party applications. The Incidents & Reports Portal is delivered as an out-of-browser (OOB) Silverlight application.
Incident Review	The act of listening to audio from a given incident, reviewing other situational information (e.g. geography, other recent incidents, etc.), and appending such information to the incident record for use by subsequent users of the alert
Incident Review Center	The 24x7x365 monitoring facility maintained by SST, Inc. which reviews customer incidents for all ShotSpotter Flex and some ShotSpotter OnSite customers. The facility is located in Newark, CA.

Initial (Machine) Classification	The computer-generated initial classification of an incident, delivered as part of a raw (unreviewed) alert. The Machine Classifier implements learning algorithms and subsequent reclassification may improve its initial classification accuracy.
Initial Location	The first location calculation performed by the ShotSpotter server, commonly not reported because it is replaced within 1 second by the final location result
Multilateration	A term originating in navigation, multilateration is a technique based on the measurement of the difference in distance to two or more stations at known locations that broadcast signals at known times. Unlike measurements of absolute distance or angle, measuring the difference in distance results in an infinite number of locations that satisfy the measurement. When these possible locations are plotted, they form a hyperbolic curve. To locate the exact location along that curve, a second measurement is taken to a different pair of stations to produce a second curve, which intersects with the first. When the two are compared, a small number of possible locations are revealed, producing a “fix.”
Muzzle Blast	The expulsion of gas from the end of the barrel of a gun after it has been fired which creates a high amplitude, wide-band pressure wave (commonly called the "bang" of the gun firing)
Notification	The act of raising an alert as a result of an incident detection or subsequent updates to its stats.
Notification API	The Application Programming Interface defining the characteristics of ShotSpotter incident alert notifications.
Notification Engine	Client-side software which performs actual notifications consistent with the Notification API. Notifications take the form of message packets.
[Message] Packet	The actual content of a notification, sent from a Notification Engine instance to one endpoint.
Parameters	Specific characteristics of certain endpoints which define them specifically. For example, there may be many endpoints identified as “camera,” but each will have different latitude and longitude parameters. Parameters are often used to customize a message packet for a specific endpoint (for example, to point a camera in the correct direction relative to *that specific camera’s* location).

Point Protection	The protection of a relatively small area, such as circle of 100 or 200 meters radius around a given point. Contrast Wide-Area Acoustic surveillance
Raw alert	*See* *Reviewed versus Unreviewed ("Raw") Alerts* above
Queue	For each attached Device, Notification Engine maintains a message queue to cache packets. The message queue ensures that 1) packets are delivered in the order of occurrence and 2) no packets are lost.
Reclassification	The act of changing the type of an incident, often performed by an Incident Reviewer, after listening to the audio recording or otherwise learning of an incident's true nature
Reviewed Alert	*See* *Reviewed versus Unreviewed ("Raw") Alerts* above
Reviewed Alert Service	The 24x7x365 monitoring service provided by SST, Inc. to its Flex customers and certain OnSite customers. The Reviewed Alert Service is staffed by incident review specialists who have heard thousands of gunfire and non-gunfire training incidents.
Sensor	The edge device, deployed by SST, Inc., comprising a microphone and on-board electronics to process sounds heard by the microphone. When the sensor triggers on an impulsive noise, it transmits a small datagram to the server detailing certain characteristics of the noise, including its sharpness and amplitude. In contrast to other sensor-based surveillance technologies, ShotSpotter sensors are only part of the overall solution. The ShotSpotter server coordinates the data streams from multiple sensors, aggregates the data, disambiguates and optimizes solutions, and presents a single, unified view of each individual incident. Such unified views commonly reflect the input of anywhere from two (2) to twenty (20) sensors.
Sensor Network	A network of collaborative sensors deployed over a wide area, possibly many square miles, which transmit data to a ShotSpotter server. No individual sensor "locates" a gunfire incident, but rather many sensors will often trigger in response to a single incident. The ShotSpotter server software is solely responsible for managing these multiple triggers and transforming multiple input streams into a single, unified view of the acoustic event, called a gunfire incident.
Server	ShotSpotter server software which aggregates acoustic impulse data from many sensors, correlates and

	disambiguates the signals, and produces a single unified view of individual gunfire incidents in near realtime.
ShotSpotter Flex	The subscription-based offering of ShotSpotter technology with a cloud-based infrastructure and with no up-front capital investment required from customers. *Compare ShotSpotter OnSite*.
ShotSpotter Onsite	A ShotSpotter system sold as capital equipment (not on a subscription basis). *Compare ShotSpotter Flex*.
Subscription	The binding of a specific Device (endpoint) to a specific packet (message) type. A Device may subscribe to one or more packet types, and a given packet type may be transmitted to one or more devices.
Triangulation	the process of determining the location of a point by measuring angles to it from known points at either end of a fixed baseline, rather than measuring distances to the point directly (trilateration). The point can then be fixed as the third point of a triangle with one known side and two known angles
Unreviewed (Raw) Alert	*See Reviewed versus Unreviewed ("Raw") Alerts* above
Wide-Area Acoustic surveillance	the surveillance of a large area, commonly many square miles, for specific acoustic events, e.g. gunfire or explosions

Appendix II: Checksum Calculation Algorithm

1. HTTP Protocol

When calculating checksum, field name, equal signs (=), field delimiters (&), and prefix field value are EXCLUDED in calculation. Following algorithm in C# illustrates how the algorithm works. The input variable `s` is the packet body. For example, for packet

```
	prefix=INCAUPD&id=1234&distance=120.00&heading=315.68&checksum=0E
```

the string used for checksum calculation equals

```
	1234120.00315.68
```

2. Socket Protocol

When calculating checksum, command prefix, command ending, commas, and checksum indicator (the * sign) are EXCLUDED in calculation. Following algorithm in C# illustrates how the algorithm works. The input variable `s` is the packet body. For example, for packet

```
	$INCAUPD,1234,120.00,315.68*0E\r\n
```

the string used for checksum calculation equals

```
	1234120.00315.68
```

Exclusion of delimiters and prefixes ensures that the same packet will get same checksum regardless the protocol used for transportation. However, if customized formats are used, the packet may have a different checksum due to variance in number formats and presented fields. The following code (in C#) demonstrates the checksum algorithm. We picked a simple algorithm so it can be easily implemented by customers when necessary.

```
	byte[] outputData = ASCIIEncoding.ASCII.GetBytes(s);
	byte val = outputData [0];

	for (int i = 1; i < outputData.Length; i++)
	{
		val ^= outputData [i];
	}
	string ret = commandPrefix + s
	             + "*" + String.Format("{0:X}", (val & 0xF0) >> 4)
	             + String.Format("{0:X}", val & 0x0F) + "\r\n";
```

Appendix III: Plug-In Architecture

You'll need to author a .Net assembly that implements the `IProtocolBase` interface defined in `NEPluginInterface.dll`, which is part of Notification Engine installation. The `IProtocolBase` only contains one method: `DeliverNotification()`. This method will be called whenever the Notification Engine attempts to send a packet to the device.

The method has 8 parameters, as detailed as following:

- `string address`

The address contains the path to your assembly. Your assembly is responsible to maintain actual device address.

- `string body`

Formatted packet message. This is what you send to the device.

- `bool need_response`

Indicate if the system requires a response. If this flag is true, you should send back `INCARSP` packet once the packet is accepted.

- `int max_delay`

Maximum delay, in seconds, is allowed. The packet should be discarded if it's not been sent when maximum delay expires.

- `int max_retry`

Maximum number of attempts allowed. Your code shouldn't try more than specified times to send the packet.

- `int retry_interval`

Interval between two delivery attempts.

- `ref string message`

A string message you'd like to return. This message will be saved to Notification Engine log file and displayed on Notification Engine log screen.

- `ref string exp`

Exception stack trace, if there's an exception.

Compile your assembly and enter the path to this assembly as the device address. This assembly is dynamically loaded during message dispatch and will be unloaded afterwards.