



Points of interest

1 Contents

2	Use-cases	2
3	General points of interest	2
4	TPEG	2
5	Specific points of interest	3
6	Weather	4
7	Security and privacy considerations	4
8	Access to location information	4
9	Attack surface	4
10	Other requirements	5
11	Open questions	5
12	Recommendations	5
13	TPEG	6
14	General POI providers	8
15	Specific POI providers	9
16	Weather	9
17	Dealing with multiple categories	10

18 Use-cases

19 General points of interest

20 Third-party applications (the “provider”) might be aware of general “points of
21 interest” for mapping. For example, an accommodation booking app-bundle
22 might be aware of the locations of hotels, either from a particular chain if the
23 app-bundle is for that chain, or for all hotels if the app-bundle is for a general
24 service like Trip Advisor.

- 25 • It must be possible for a third-party app-bundle to provide these “points
26 of interest” to a navigation app.
- 27 • It must be possible for the third-party app-bundle to contain a non-GUI
28 service (an “agent”) which will act as the POI provider.
- 29 • The navigation app should not be flooded with irrelevant points of inter-
30 est, for example a hotel in Cambridge while driving from Hannover to
31 Hildesheim
- 32 • For privacy, if an application bundle advertises that it is a points-of-
33 interest provider but its manifest indicates that it is not intended to have
34 access to location data, the navigation app must not reveal the current
35 location to it.
- 36 • If the POI provider is implemented in terms of an Internet service that
37 can provide a filter/search-based stream of points of interest, it must be
38 possible for the POI provider to avoid being flooded with irrelevant points
39 of interest by the remote server.
- 40 • This pattern could potentially be generalized to replace the navigation
41 app with any other consumer.
- 42 • We will assume that there is no particular preferred encoding that will be

43 used by a majority of applications, so re-encoding into a common format
44 will usually be necessary.

45 TPEG

46 One specific instance of **General points of interest** is receiving **TPEG**¹ broad-
47 casts. These are typically carried on digital radio (DAB) and can encode time-
48 and location-bound events such as traffic congestion, road closures and weather.
49 They can also be transferred via the Internet.

50 When TPEG is broadcast over DAB, this is done with a “**carousel**”² approach
51 in which the transmitter repeatedly cycles through a list of currently-valid mes-
52 sages. It is not currently clear to us whether TPEG feeds over the Internet
53 follow the same “carousel” design.

54 It is not currently clear to us whether TPEG feeds over the Internet filter for
55 relevant events at the client- or server-side.

- 56 • It must be possible for an app-bundle to contain an agent that receives
57 TPEG broadcasts and provides events to a navigation app.
- 58 • The navigation app should not be flooded with irrelevant points of interest,
59 for example traffic congestion that is not close to either the current location
60 or the projected route.
 - 61 – If filtering is performed at the server side, the TPEG provider agent
62 should be able to give the server a suitable filter.
 - 63 – If filtering is performed at the client side, it could take place in the
64 agent rather than the navigation app, to minimize IPC traffic between
65 them.
 - 66 – If filtering is performed in the navigation app itself, the channel be-
67 tween the TPEG provider agent and the navigation app must be one
68 that is suitable for high-throughput events, and should be able to
69 feed back that events are being transferred faster than the naviga-
70 tion app can read them, so that the TPEG provider agent can “back
71 off” by omitting some events.
- 72 • More than one TPEG provider can be installed at the same time. If they
73 are, they are all used in parallel.
- 74 • **A brochure from the Institut für Rundfunktechnik**³ uses 64 kbit/s = 56
75 messages/s as an indicative figure for TPEG over DAB. This implies that
76 an average binary message should be in the range 100-200 bytes.
- 77 • **The European Broadcasting Union’s document “TPEG - What is it all
78 about?”**⁴ suggests that TPEG-over-Internet clients are expected to poll
79 servers, using a model in which the client downloads a comparatively large

¹<http://tisa.org/technologies/tpg/>

²https://en.wikipedia.org/wiki/Data_and_object_carousel

³https://www.easyway-its.eu/sites/default/files/EW-Highlight_Bavaria_Multimodal_TPEG.pdf

⁴<https://tech.ebu.ch/docs/other/TPEG-what-is-it.pdf>

80 “initial state” at startup, and subsequently receives “deltas” from that
81 initial state.

82 **Specific points of interest**

83 Third-party applications (the “provider”) might be aware of “points of interest”
84 that are relevant to the user with a high probability. For example, an accommoda-
85 tion or travel booking app-bundle might be aware that the user has booked
86 a stay at a particular hotel, or a flight from a particular airport; or a PIM ap-
87 plication might be aware of the location of the user’s home or normal place of
88 work, or an upcoming appointment.

- 89 • It must be possible for a third-party app-bundle to provide these “points
90 of interest” to a navigation app.
- 91 • Unlike **General points of interest**, being flooded with irrelevant points of inter-
92 est is unlikely to be a problem here, because the “provider” application
93 knows that these points of interest are highly likely to be relevant.
- 94 • Time-based filtering could be useful here: appointments beyond a defined
95 date range might not be considered relevant.
- 96 • This pattern could potentially be generalized to replace the navigation
97 app with a non-specific “sink”.

98 **Weather**

99 Weather information might be queried from an Internet service or decoded from
100 **TPEG** broadcasts by an agent in a built-in, preinstalled or third-party app-
101 bundle.

- 102 • The requirements closely resemble **General points of interest**: we require
103 the weather at one or a few weather stations closest to our location or
104 projected route, and we are not interested in distant weather reports.
- 105 • Date-based queries (e.g. projected weather for the next week while plan-
106 ning a long drive) might also be useful here.
- 107 • We anticipate that the data rate here will be considerably lower than for
108 **TPEG** in general.

109 **Security and privacy considerations**

110 **Access to location information**

111 As noted above, if an application bundle advertises that it is a points-of-interest
112 provider but its manifest indicates that it is not intended to have access to
113 location data, the navigation app must not reveal the current location to it.

114 **Attack surface**

115 **TPEG** messages come from DAB broadcasts or the Internet.

116 Regardless of how well we might mitigate attacks, if an attacker is able to send
117 crafted TPEG messages to the Apertis system in a way that is indistinguishable
118 from legitimate data, then that attacker can cause the navigation app to display
119 points-of-interest of their choice. This is unavoidable, and is mentioned here only
120 for completeness.

121 Beyond that, if attacker can cause the Apertis system to receive crafted TPEG
122 messages, they might be able to exploit implementation errors in a TPEG parser.
123 We consider three threat-models here:

- 124 • assume that the parser is robust and will not crash or misbehave with
125 malicious input;
- 126 • alternatively, assume that the parser has a denial-of-service vulnerability
127 which causes it to crash or otherwise cease to process information, but
128 does not allow arbitrary code execution;
- 129 • alternatively, assume that the parser has an arbitrary code execution vul-
130 nerability which causes it to execute attacker-chosen code

131 The second and third threat models should be considered to be vulnerabilities
132 (security-sensitive bugs) in the TPEG-parsing component; and can be made less
133 likely by using techniques such as [fuzz testing](#)⁵ to verify the robustness of the
134 parser. However, it might be considered valuable to mitigate any vulnerabilities
135 in those classes that remain and are discovered by an attacker.

136 Similarly, a non-TPEG-based points-of-interest or weather information provider
137 is likely to receive data in some non-TPEG format from the Internet, and equiv-
138 alent security considerations apply to that data. We recommend that crypto-
139 graphically protected channels such as HTTPS are used where possible.

140 Other requirements

141 It is undesirable to increase coupling between consumers and providers by hav-
142 ing consumers provide any location/route information to providers. Instead,
143 providers should retrieve that information from the platform.

144 Similarly, it would be possible for the consumer to give the provider an indication
145 of what is required (in terms of level of detail, search radius around the location
146 and so on). This would allow the provider to optimize its trade-off between
147 resource consumption and information provided, by providing what is needed
148 but no more. However, this causes relatively tight coupling between consumers
149 and providers, which is undesired in an app-centric model. This should not
150 be implemented; instead, the provider should be responsible for determining a
151 reasonable policy.

⁵https://en.wikipedia.org/wiki/Fuzz_testing

152 Open questions

- 153 • Does TPEG already have a defined encoding for points-of-interest similar
154 to those discussed in [General points of interest](#) and [Specific points of
155 interest](#), or does it only have encodings for traffic, weather information,
156 and a few specific classes of point-of-interest such as parking?
- 157 • Are the data rates discussed above (64 kbit/sec, 56 messages/sec) repre-
158 sentative?
- 159 • How frequently do we anticipate that a consumer would wish to consume
160 some families of location-sensitive data (“applications” in TPEG jargon -
161 traffic information, weather, points of interest) but not others?
- 162 • Do we expect weather information to come from TPEG, or from a
163 query/lookup-based web service with requests like “tell me the weather
164 in Hildesheim tomorrow”, or a combination of the two?

165 Recommendations

166 See [Data sharing](#)⁶ for background information on various possible communica-
167 tion between apps. The *consumer* here is the navigation app, and the *providers*
168 are the POI, TPEG and weather providers.

169 We anticipate that the navigation app (or other consumer) might either be a
170 HMI that is started via user action or a platform component that is started
171 automatically on boot, whereas the various providers will be agents provided
172 by platform components and/or store applications, started either on boot or on-
173 demand. Because the consumer might be a HMI, we recommend that it should
174 act as the *initiator* as described in [Data sharing](#)⁷.

175 We recommend that the navigation app should locate suitable POI, weather
176 and/or TPEG providers by performing [interface discovery](#)⁸.

177 To bypass concerns about [Access to location information](#) and ensure that con-
178 sumers and providers remain “loosely coupled”, we recommend that the con-
179 sumer does not inform providers about the current location or route. Instead,
180 each provider that requires this information should retrieve it from a platform
181 service via a D-Bus API, following the conventional publish/subscribe model.
182 The platform should only allow this access for providers whose app-bundle man-
183 ifests authorize it.

184 TPEG

185 For TPEG, there is a choice between several solutions. The answers to the [Open](#)
186 [questions](#) above influence the choice between these options.

⁶https://em.pages.apertis.org/apertis-website/architecture/data_sharing/

⁷https://em.pages.apertis.org/apertis-website/architecture/data_sharing/

⁸https://em.pages.apertis.org/apertis-website/concepts/interface_discovery/

187 Our provisional recommendation is to take the **TPEG stream** design, and mini-
188 mize exploitable bugs in TPEG parsing by subjecting the parser to fuzz testing
189 and code auditing.

190 Where TPEG is received with a **carousel**⁹ model, we recommend that the TPEG
191 provider is responsible for keeping a cache of items received during previous
192 sessions, forgetting those items when they are no longer valid or relevant, and
193 replaying them when each new consumer connects.

194 **TPEG stream**

195 One option is to use **data sharing#Consumer-initiated push via a stream**¹⁰. In
196 this model, the method that is called by the provider would start a stream of
197 TPEG messages, in either the binary or XML encoding, with a suitable framing
198 protocol (which will need to be specified) if one is required.

199 This solution has the advantage that it allows a TPEG provider to treat TPEG
200 as opaque: for example, a DAB radio receiver that is primarily designed to play
201 audio, but receives TPEG via DAB as a side-effect, could pass TPEG messages
202 through to navigation software without parsing or understanding them. This
203 avoids requiring TPEG parsing in DAB applications or agents, leading to high
204 throughput and low resource consumption by these processes (but potentially
205 a correspondingly higher resource consumption for the consumer, which does
206 need to parse the TPEG).

207 The disadvantage of this solution is that if a provider takes this TPEG pass-
208 through approach, the consumer is directly exposed to potentially malicious data
209 received from the network; for example, a malformed TPEG message might be
210 designed to exploit bugs in the consumer's parser.

211 **Stream of some other format**

212 Another option is to use **data sharing#Consumer-initiated push via a stream**¹¹,
213 but require the messages carried by the stream to be in some other format
214 instead of TPEG, for example **KML**¹². This would force the provider to parse
215 TPEG and re-encode it in the other format, which comes with an efficiency
216 cost. It also requires the choice and implementation of the other format; if
217 a new format is designed, requirements-gathering and design will be needed,
218 which could be viewed as a waste of effort that could be avoided by reusing an
219 existing format.

220 All the usual trade-offs between data formats - size efficiency, time efficiency,
221 expressiveness, scope for debugging, and so on - apply to the choice of the other

⁹https://en.wikipedia.org/wiki/Data_and_object_carousel

¹⁰https://em.pages.apertis.org/apertis-website/architecture/data_sharing/#consumer-initiated-push-via-a-stream

¹¹https://em.pages.apertis.org/apertis-website/architecture/data_sharing/#consumer-initiated-push-via-a-stream

¹²https://en.wikipedia.org/wiki/Keyhole_Markup_Language

222 format. GVariant, JSON or XML might make a reasonable basis for a data
223 format, but each of those would require a suitable data representation (schema)
224 to be designed and specified, similar to the way KML is layered above XML.

225 If the TPEG parser is assumed to be robust, this approach has no advantage
226 over the TPEG stream. The advantage of this option over a TPEG stream is
227 that if the TPEG parser is not considered to be robust, this option somewhat
228 reduces the **attack surface** available to potentially-malicious TPEG messages.

229 If the attacker is only assumed to be able to cause a crash via a malformed
230 message, this is entirely mitigated by performing TPEG parsing in the provider:
231 the consumer would stop receiving messages from that provider, but continue
232 to run. Optionally, it could detect the unexpected end-of-stream and re-initiate
233 communication with a new instance of the provider.

234 However, if the attacker is assumed to be able to cause arbitrary code execu-
235 tion in the TPEG parser, this design does not necessarily provide any mitiga-
236 tion: having subverted the TPEG provider, the attacker could send arbitrary
237 messages to the consumer in the other format, potentially triggering denial-of-
238 service or code-execution vulnerabilities in the parser for that other format.
239 This could be mitigated by requiring that the parser for the other format has a
240 robust and well-understood implementation provided by the platform.

241 **Publish/subscribe via D-Bus**

242 TPEG providers could use a **publish/subscribe approach via D-Bus**¹³, with each
243 D-Bus message either carrying a binary or XML TPEG message, or a message
244 translated into a non-TPEG encoding using D-Bus data structures. However,
245 our current understanding of the expected message rate is that it is at the high
246 end of the rates for which D-Bus was designed, so we do not recommend this.

247 In addition, if the TPEG provider does not cache currently-valid TPEG mes-
248 sages in memory but merely passes them through as they are received, then
249 this would be a poor fit for conventional D-Bus design patterns, since a GetCur-
250 rentState() method call would not make sense (the provider would not have any
251 current state to get).

252 **General POI providers**

253 For POI providers that are implemented in terms of access to a web service,
254 either the POI provider can perform a series of search operations on the remote
255 server and get relevant POIs in response (a “pull” model), or the remote server
256 can send a stream of POIs to the POI provider while periodically polling the
257 current locations of interest (a “push” model).

258 For the transport between the POI provider and the POI consumer, multiple
259 approaches are possible, depending on the answers to the **Open questions**.

¹³https://em.pages.apertis.org/apertis-website/architecture/data_sharing/#publish.2fsubscribe-via-d-bus

260 Our provisional recommendation is that points of interest are encoded as TPEG
261 and transferred to consumers via the same mechanisms we would use for a TPEG
262 stream.

263 **TPEG stream**

264 If TPEG has an encoding for points of interest, then the same approach can
265 be taken as for TPEG: [data sharing#Consumer-initiated push via a stream](#)¹⁴,
266 with a stream of binary or XML TPEG messages. This has the advantage that
267 we do not need to define our own encoding for points of interest.

268 Another advantage of this approach (assuming that TPEG providers also pro-
269 vide a TPEG stream) is that the consumer only needs to consume TPEG, rather
270 than consuming both TPEG (from DAB broadcasts) and some other format
271 (from general POI providers). If TPEG does not already have an encoding for
272 points of interest, but it is feasible to add one, it might still be worthwhile to
273 do so in order to receive the second advantage.

274 The disadvantage is that each provider needs to encode whatever information it
275 provides into TPEG format. This could introduce a loss of information if TPEG
276 is not sufficiently expressive to describe everything the provider requires.

277 If the corresponding solution is not chosen for TPEG providers, this approach
278 should not be chosen either.

279 **Stream of some other format**

280 As with TPEG, we could define a non-TPEG encoding for points of interest and
281 stream them to the consumer using [Data sharing#Consumer-initiated push via](#)
282 [a stream](#)¹⁵. Similar considerations apply to the design of the new format: we
283 could select an existing format such as [KML](#)¹⁶, or design a new one, perhaps
284 using GVariant or JSON as a base.

285 If the non-TPEG encoding is chosen well, it should be possible to encode all
286 aspects of a point of interest without information loss. However, if the non-
287 TPEG encoding is not sufficiently expressive, information might still be lost in
288 translation.

289 **Publish/subscribe via D-Bus**

290 As with TPEG, POI providers could use a [publish/subscribe approach via D-](#)
291 [Bus](#)¹⁷, with each D-Bus message carrying one or more points of interest, either

¹⁴https://em.pages.apertis.org/apertis-website/architecture/data_sharing/#consumer-initiated-push-via-a-stream

¹⁵https://em.pages.apertis.org/apertis-website/architecture/data_sharing/#consumer-initiated-push-via-a-stream

¹⁶https://en.wikipedia.org/wiki/Keyhole_Markup_Language

¹⁷https://em.pages.apertis.org/apertis-website/architecture/data_sharing/#publish.2fsubscribe-via-d-bus

292 described using D-Bus data structures or as an opaque byte-array in some known
293 format. We anticipate that POI providers would normally have rather lower
294 message rates than TPEG, so they might be within the range for which D-Bus
295 is designed, even if TPEG is not.

296 **Specific POI providers**

297 The implementation options for specific POI providers are essentially the same
298 as general POI providers, although their requirements are less stringent: the
299 message rate is anticipated to be lower, and location information is not neces-
300 sarily needed.

301 We recommend that the same interface as for general points of interest is used:
302 we do not see any reason why we need to distinguish between the two.

303 **Weather**

304 One possible implementation is to use the same TPEG-stream-based design as
305 the other use cases. This is appropriate if we anticipate that the majority of
306 providers of weather information will be implemented using TPEG or similar,
307 but it is a poor fit for a more query-based provider such as one that accesses
308 a web service: the provider would have to perform speculative queries covering
309 the area around the location and/or route, and emit their results at intervals as
310 TPEG or TPEG-like messages.

311 Another possible implementation is to use [Data_sharing#Query-based access
312 via D-Bus](https://em.pages.apertis.org/apertis-website/architecture/data_sharing/#query-based-access-via-d-bus)¹⁸, which is a good fit for implementations that query a web service,
313 but a poor fit for implementations that receive weather from TPEG (which
314 would have to cache all available weather information, and use their cache to be
315 able to answer queries). This could be mitigated by treating weather queries as
316 a platform service, so that the cache is maintained by the platform.

317 A third possibility is to have two separate weather APIs - one stream-based
318 design for “ambient weather information”, and one query-based design for “cur-
319 rent and forecasted weather queries” - and require consumers to choose whether
320 to use one or both depending on their particular requirements. This design has
321 the advantage that each of those APIs represents the underlying weather service
322 in the most natural way, but has the disadvantage that consumers are expected
323 to use two parallel APIs.

324 We do not currently have a specific recommendation here, and would appreciate
325 feedback on the relative priorities of those designs’ strengths and weaknesses.

¹⁸https://em.pages.apertis.org/apertis-website/architecture/data_sharing/#query-based-access-via-d-bus

326 **Dealing with multiple categories**

327 This recommendation is conditional on the answers to the [Open questions](#),
328 above.

329 We recommend using a name such as “categories”, “types” or “classes” for
330 TPEG’s jargon term “applications”, to avoid confusion with the already over-
331 loaded term “application”. Documentation could mention the TPEG jargon for
332 clarification, for example:

```
333 /** * ... * Return a list of categories of location information that are sup-  
334 ported * by this provider. These categories are the same concept as * "applica-  
335 tions" in TPEG terminology. * ... */ gchar **..._list_categories (... *self);
```

336 If we anticipate that all categories of data will typically all have the same con-
337 sumers, we recommend having a single shared interface for [interface discovery](#)¹⁹,
338 perhaps `org.apertis.LocationInfoProvider` OR `org.apertis.TPEGProvider`. Clients
339 not requiring all of the available data for that interface could receive and discard
340 it.

341 If we anticipate that categories will often have different consumers, we will need
342 a list of categories, and an interface for [interface discovery](#)²⁰ per category, for
343 example `PointsOfInterestProvider`, `TrafficProvider` and `WeatherProvider`.

344 One approach to these categories would be to define a separate D-Bus interface
345 per interface-discovery interface, with intentionally similar D-Bus APIs to set
346 up a stream. Each provider that could provide more than one category would
347 be required to demultiplex the messages that it receives and write them into
348 the appropriate streams.

349 Another approach to these categories would be to define a separate D-Bus in-
350 terface per interface-discovery interface, with intentionally similar D-Bus APIs
351 to set up a stream. Each provider that could provide more than one category
352 would be required to demultiplex the messages that it receives and write them
353 into the appropriate streams.

¹⁹https://em.pages.apertis.org/apertis-website/concepts/interface_discovery/

²⁰https://em.pages.apertis.org/apertis-website/concepts/interface_discovery/