

2

3 **Auto-ID Object Name Service (ONS) 1.0**

4 Auto-ID Center Working Draft 12 August 2003

5 This version:

6 <http://develop.autoidcenter.org/TR/2003/WD-ons-1.0-20030930.pdf>

7 Latest version:

8 <http://develop.autoidcenter.org/TR/ons-1.0.pdf>

9 Previous versions

10 <http://develop.autoidcenter.org/TR/2003/WD-ons-1.0-20030925.pdf>

11 <http://develop.autoidcenter.org/TR/2003/WD-ons-1.0-20030718.html>

12 <http://www.autoidcenter.org/publishedresearch/MIT-AUTOID-TM-004.pdf>

13 Editors:

14 Michael Mealling (VeriSign) <mailto:michael@verisignlabs.com>

15 Copyright ©2003 [Auto-ID Center](#)[®], All Rights Reserved.

16 **Abstract**

17 This document specifies how the Domain Name System is used to locate authoritative
18 metadata and services associated with a given Electronic Product Code (EPC). Its target
19 audience is developers that will be implementing ONS resolution systems for
20 applications.

21 **Status of this document**

22 This section describes the status of this document at the time of its publication. Other
23 documents may supersede this document. The latest status of this document series is
24 maintained at the Auto-ID Center.

25 This document is the first public working draft of the ONS Working Group. It differs in
26 several important ways from the [ONS Technical Manual \(0.5\) document](#) as published
27 by the AutoID Center. Specifically it includes the addition of the use of NAPTR records
28 and the removal of the pre-resolver and the resolution of serial numbers. Comments on
29 this document should be sent to the Auto-ID ONS Working Group mailing list [sag-](#)

30 ons@develop.autoidcenter.org (archived at
31 <http://develop.autoidcenter.org/archives/sag-ons/>).

32 This is an Auto-ID Center Working Draft for review by Auto-ID Members and other
33 interested parties. It is a draft document and may be updated, replaced or made obsolete
34 by other documents at any time. It is inappropriate to use Auto-ID Working Drafts as
35 reference material or to cite them as other than "work in progress". This is work in
36 progress and does not imply endorsement by the Auto-ID membership
37

38 **Table of contents**

39	1	Background Information.....	3
40	2	EPC System Network Architecture.....	3
41	2.1	EPC Network Software Architecture Components	3
42	2.1.1	Readers.....	4
43	2.1.2	Savant.....	4
44	2.1.3	EPC Information Service	4
45	2.1.4	ONS – Object Name Service	5
46	2.1.5	ONS local cache.....	5
47	2.1.6	ONS Root.....	5
48	2.2	EPC Network Data Standards.....	5
49	2.2.1	Electronic Product Code (EPC)	5
50	2.2.2	Physical Markup Language (PML).....	6
51	2.3	EPC Network Architecture – across Enterprises	6
52	3	ONS Introduction.....	6
53	4	EPC.....	8
54	4.1	Serial Number Level Queries to the ONS	8
55	5	DNS Query Format	8
56	6	Results Format.....	8
57	7	Examples	10
58	7.1	Finding a WSDL file for a product	10
59	7.2	Finding a PML server for a product	10
60	7.3	Finding an HTML formatted web page description of a product.....	11
61	7.4	Finding an XML-RPC gateway to the Web Service interfaces	11
62	8	References	11
63	9	Appendix A -- Glossary	12

64	10	Appendix B -- DDDS Application Specification.....	14
65	10.1	Application Unique String	14
66	10.2	First Well Known Rule	14
67	10.3	Expected Output	14
68	10.4	Valid Databases	14
69	10.5	Valid Flags	15
70	10.6	Service Parameters	15
71	11	Appendix C -- Service Field Registrations	15
72	11.1	Registration Template	15
73	11.2	Service Registrations	16

74

75 **1 Background Information**

76 This document draws from the previous work at the Auto-ID Center, and we recognize
77 the contribution of the following individuals: Joe Foley (MIT), Erik Nygren (MIT),
78 Sanjay Sarma (MIT), David Brock (MIT), Sunny Siu (MIT), Laxmiprasad Putta (Oat),
79 Sridhar Ramachandran (Oat). The following papers capture the contributions of these
80 individuals:

- 81 ▪ Engels, D., Foley, J., Waldrop, J., Sarma, S. and Brock, D., "The Networked Physical
82 World: An Automated Identification Architecture" Proceedings of the IEEE/ACM
83 International Conference on Computer Aided Design (ICCAD01), 76-77, 2001.
- 84 ▪ The Object Name Service Technical Manual, Version 0.5 (Beta)
85 <http://www.autoidcenter.org/publishedresearch/MIT-AUTOID-TM-004.pdf>

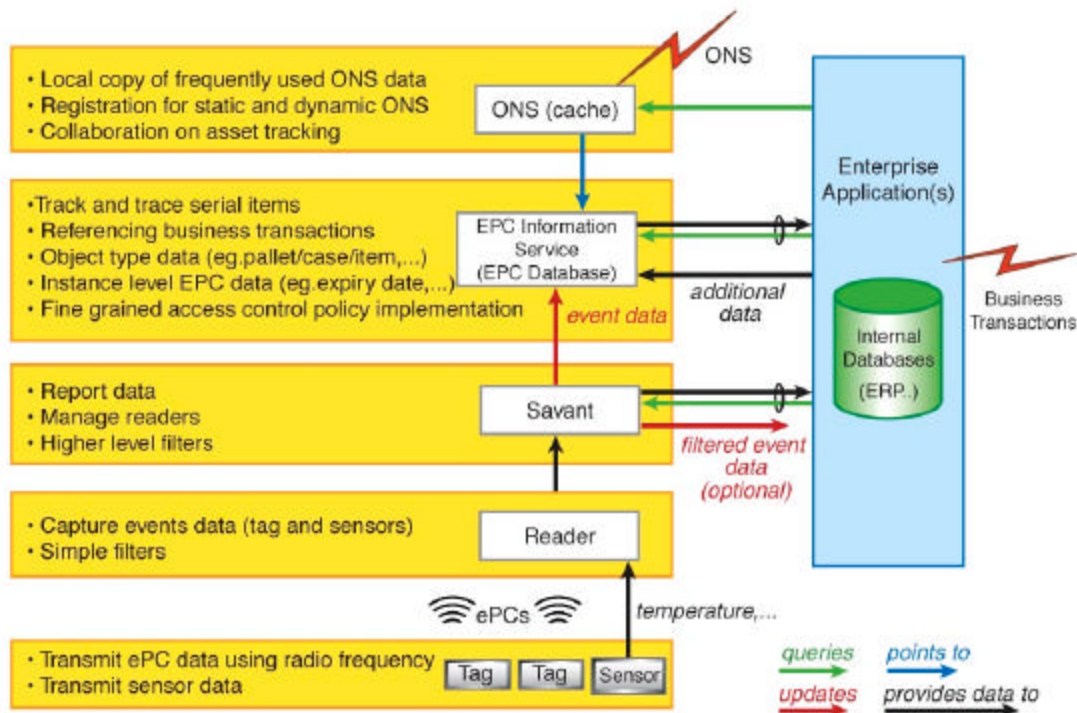
86 **2 EPC System Network Architecture**

87 Radio Frequency Identification is a technology used to identify, track and locate assets.
88 The vision that drives the developments at the Auto-ID Centre is the universal unique
89 identification of individual items. The unique number, called EPC (electronic product
90 code) will be encoded in an inexpensive Radio Frequency Identification (RFID) tag. The
91 EPC Network will also capture and make available (via Internet and for authorized
92 requests) other information that pertains to a given item to authorized requestors.

93 **2.1 EPC Network Software Architecture Components**

94 The EPC Network Architecture as in Fig. 1 shows the high-level components of the EPC
95 Network.

EPC Network Architecture-inside the Enterprise



96

97

Figure 1: EPC Network Architecture: Components and Layers

98

These functional components from the figures above are described in the sections below.

99

2.1.1 Readers

100

Readers are devices responsible for detecting when tags enter their read range. They may also be capable of interrogating other sensors coupled to tags or embedded within tags.

101

102

The Auto-ID Reader Protocol Specification 1.0 defines a standard protocol by which Readers communicate with Savants and other hosts. The Savant also has an “adapter” provision to interface to older readers that do not implement the Auto-ID Reader Protocol.

103

104

105

106

2.1.2 Savant

107

Savant is “middleware” software designed to process the streams of tag or sensor data (event data) coming from one or more reader devices. Savant performs filtering, aggregation, and counting of tag data, reducing the volume of data prior to sending to Enterprise Applications. The Auto-ID Savant Specification 1.0 defines the working of Savant, and the interface to Enterprise Applications.

108

109

110

111

112

2.1.3 EPC Information Service

113

The EPC Information Service makes EPC Network related data available in PML format to requesting services. Data available through the EPC Information Service may include tag read data collected from Savant (for example, to assist with object tracking and tracing at serial number granularity); instance-level data such as date of manufacture,

114

115

116

117 expiry date, and so on; and object class-level data such as product catalog information.
118 In responding to requests, the EPC Information Service draws upon a variety of data
119 sources that exist within an enterprise, translating that data into PML format. When the
120 EPC data is distributed across the supply chain, an industry may create an EPC Access
121 Registry that will act as a repository for EPC Information Service interface descriptions.
122 The Auto-ID EPC Information Service Specification 1.0 defines the protocol for
123 accessing the EPC Information Service.

124 **2.1.4 ONS – Object Name Service**

125 The Object Name Service provides a global lookup service to translate an EPC into one
126 or more Internet Uniform Reference Locators (URLs) where further information on the
127 object may be found. These URLs often identify an EPC Information Service, though
128 ONS may also be used to associate EPCs with web sites and other Internet resources
129 relevant to an object.

130 ONS provides both static and dynamic services. Static ONS typically provides URLs for
131 information maintained by an object's manufacturer. Dynamic ONS services record a
132 sequence of custodians as an object moves through a supply chain.

133 ONS is built using the same technology as DNS, the Domain Name Service of the
134 Internet. The Auto-ID Object Name Service Specification 1.0 defines the working of
135 ONS and its interface to applications.

136 **2.1.5 ONS local cache**

137 The local ONS cache is used to reduce the need to query the global Object Name Service
138 for each object which is seen, since frequently-asked / recently-asked values can be
139 stored in the local cache, which acts as the first port of call for ONS type queries. The
140 local cache may also manage lookup of private internal EPCs for asset tracking. Coupled
141 with the local cache will be registration functions for registering EPCs with the global
142 ONS system and with a dynamic ONS system for private tracking and collaboration
143 within the supply chain seen by each unique object.

144 **2.1.6 ONS Root**

145 The ONS root is the top-level domain name of the public EPC name space. Ultimately,
146 all global lookups start from the ONS root, but the ONS local cache serves to limit the
147 number of times the root is actually queried. At the time of this publication the value of
148 the ONS root is unknown. For the purpose of exposition, "onsroot.org" will be used in
149 examples.

150 **2.2 EPC Network Data Standards**

151 The operation of the components of the EPC Network is subject to data standards that
152 specify the syntax and semantics of data exchanged among components.

153 **2.2.1 Electronic Product Code (EPC)**

154 The Electronic Product Code is the fundamental identifier for a physical object. The
155 Auto-ID Electronic Product Code Data Specification 1.0 defines the abstract content of

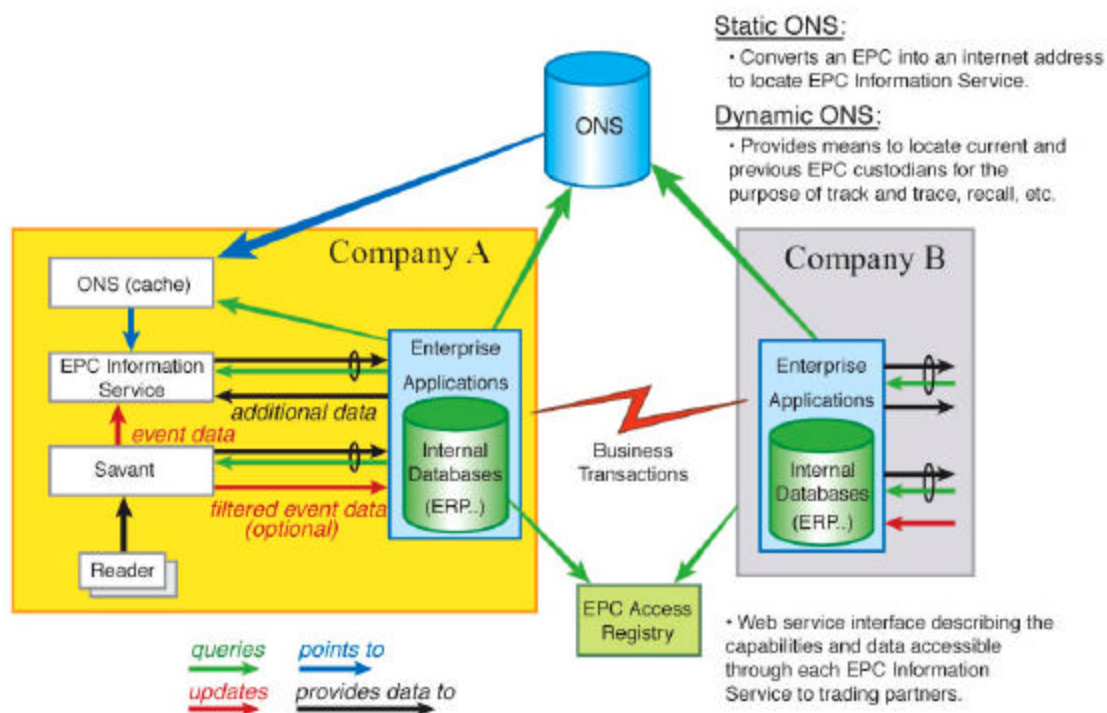
156 the Electronic Product Code, and its concrete realization in the form of RFID tags,
157 Internet URIs, and other representations.

158 2.2.2 Physical Markup Language (PML)

159 The Physical Mark-Up Language (PML) is a collection of common, standardized
160 XML vocabularies to represent and distribute information related to EPC Network
161 enabled objects. The PML standardizes the content of messages exchanged within the
162 EPC network. It is, therefore, part of the Auto-ID Center's effort to develop standardized
163 interfaces and protocols for the communication with and within the Auto-ID
164 infrastructure. The core part of the physical mark-up-language (PML Core) provides a
165 standardized format for the exchange of the data captured by the sensors in the Auto-ID
166 infrastructure, e.g. RFID readers. The Auto-ID PML Core specification 1.0 defines the
167 syntax and semantics of PML Core.

168 2.3 EPC Network Architecture – across Enterprises

EPC Network Architecture-across Enterprises



169

170

Figure 2: EPC Network Architecture: across Enterprises

171 3 ONS Introduction

172 The Auto-ID architecture provides a method for the inclusion of commercial (both
173 physical and otherwise) products within a network of information services. This
174 architecture makes several axiomatic assumptions, the most important being that it should
175 leverage existing Internet technology and infrastructure as much as possible. As such, it

176 adheres to the "hour glass model" [Willinger and Doyle] of the Internet by standardizing
177 on one identifier scheme: the Electronic Product Code (EPC) [EPC].

178 While an addressing scheme by itself is useful, it can only be used within a network when
179 a mechanism is provided to authoritatively lookup information about that identifier [RFC
180 2826]. The combination of this identifier and a resolution mechanism creates a dynamic
181 service that can be used to discover metadata about any product without the need for
182 expensive synchronized databases. This process is the core integrating principle of the
183 Auto-ID concept.

184 This rest of this document is concerned with specifying this EPC resolution service, or
185 Object Name Service (ONS). In keeping with the assumption that the architecture should
186 leverage existing Internet standards and infrastructure, the ONS uses the existing Domain
187 Name System [DNS] for looking up (resolving) information about an EPC. This means
188 that the query and response formats must adhere to the DNS standards, meaning that the
189 EPC will be converted to a domain-name and the results must be a valid DNS Resource
190 Record. Figure 1 describes a typical ONS query:

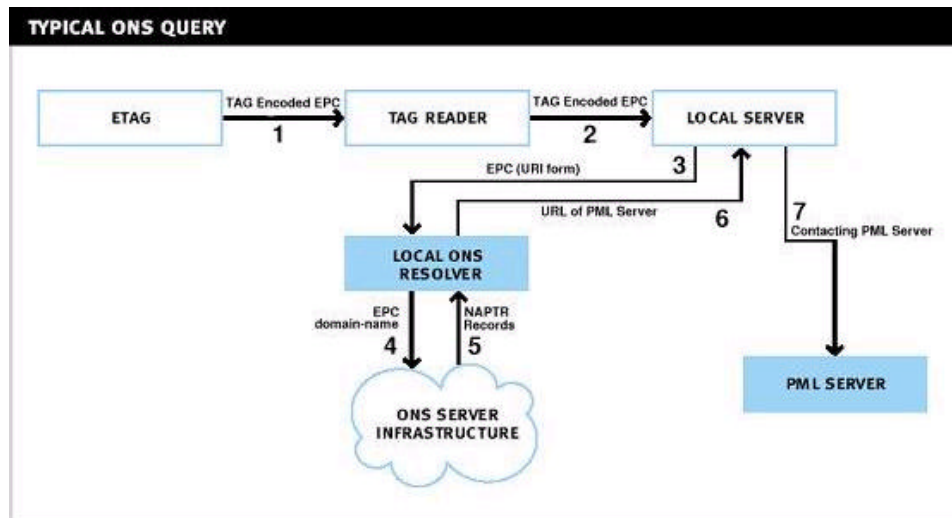


Figure 1: A typical ONS query.

191

192

193

194

195

196

197

198

199

200

201

202

203

204

1. A sequence of bits containing an EPC is read from an RFID tag
(01 0000000000000000000010 00000000000011000 00000000000000110010000)
2. The TAG Reader sends that sequence of bits to a local server
(01 0000000000000000000010 00000000000011000 00000000000000110010000)
3. That local server converts the bit sequence into the URI Form [TAG Data Standards] and sends it to the local ONS Resolver
urn:epc:1.2.24.400
4. The resolver converts the URI form into a domain-name and issues a DNS query for NAPTR records for that domain
24.2.1.onsroot.org
5. The DNS infrastructure returns a series of answers that contain URLs that point to one or more services (for example, an EPCIS Server)

- 205 6. The local resolver strips the URL from the DNS record and presents it
206 back to the local server
207 http://pml.example.com/pml-wsdl.xml
208 7. The local server contacts the correct PML server found in the URL for
209 the EPC in question

210 **4 EPC**

211 The ONS as specified here takes an EPC as input. It assumes that the EPC is in its URI
212 form (i.e. urn:epc:1.2.24.400). This form is generally used by all Auto-ID architectural
213 components beyond the TAG Reader.

214 **4.1 Serial Number Level Queries to the ONS**

215 It is important to note that the 1.0 version of this ONS specification does not query for the
216 full EPC. It specifically stops at the product level. Subsequent queries for information
217 about a given serial number must be resolved by querying the application layer server
218 designated by the ONS results. The ability to specify an ONS query at the serial number
219 level of granularity as well as the architectural and economic impacts of that capability is
220 an open issue that will be addressed in subsequent versions of this document. Its lack of
221 mention here should not be construed as making that behavior legal or illegal.

222 **5 DNS Query Format**

223 In order to query the DNS for the EPC, the URI form specified above must be converted
224 to domain-name form. Generally this follows these steps:

- 225 1. Remove the 'urn:epc:' header (leaving 1.2.24.400)
- 226 2. Remove the serial number field (leaving 1.2.24)
- 227 3. Invert the order of the remaining fields (leaving 24.2.1)
- 228 4. Append '.onsroot.org' (ending up with 24.2.1.onsroot.org)

229
230 The client application's DNS resolver is then used to query for DNS Type Code 35
231 (NAPTR) records. Since there is no standard DNS resolver API, the exact method that an
232 ONS implementation would specify this query depends on the DNS resolution library
233 being used. Once there is an ONS API the details of exactly how the DNS query is
234 constructed will be hidden from the calling application.

235 **6 Results Format**

236 The results of the query will be in the form of one or more NAPTR records. DNS
237 resolvers report their data in many, non-standardized forms (some even hand back raw,
238 unparsed network packets). Since there is no standard DNS API, the exact method for
239 parsing the binary DNS data will depend on the DNS resolver being used. Once there is
240 an ONS API the details of exactly how to parse the DNS responses will be hidden from
241 the calling application. The actual contents of the DNS records are logically formatted as
242 follows:

Order	Pref	Flags	Service	Regexp	Replacement
0	0	u	EPC+pml	!^.*\$!http://company.com/cgi-bin/pmlservice!	.(a period)

243

244 The **Order** field is used to ensure that the NAPTR records are interpreted in the correct
 245 order since DNS does not guarantee ordering of result sets. Since all of the records in a
 246 result set are valid for the EPC in question, the only effect an order value has is to
 247 indicate that some subset of the records are considered to be equivalent. If records are
 248 considered equivalent then the application is free to pick the most appropriate record
 249 given the combination of the Services and Prefs fields. This is used to achieve a type of
 250 load-balancing effect for two or more records.

251 For example if there are 4 records returned and the first three records have an Order of '0'
 252 while the last one has an Order of '1', then the first three are considered equivalent for
 253 load-balancing purposes. If those three records also have the same Pref and Service then
 254 it is valid to pick at random between the three. If the Pref values are different but the
 255 Services are still the same then the Prefs is used to indicate a preference (as in the
 256 preference number for MX records). Records with a lower number should be processed
 257 before those with higher numbers. The Services field is important in this process since
 258 applications will need to consider only those records that have Services that they are
 259 interested in.

260 The **Flags** field is set to 'u' which indicates that the **Regexp** field contains a URI. The
 261 **Service** field contains an indicator of the type of service that can be found at the URI in
 262 question. This feature allows for the ONS service to indicate multiple service end points
 263 for multiple classes of services. The **Replacement** field is not used by Auto-ID but since
 264 it is a special DNS field its value is set to a single period (.) instead of simply a blank.

265 The **Services** is particularly important since it is used to create 'classes' of servers. These
 266 classes are specified and registered with some authority. They can be used to specify very
 267 lightweight services such as a "tell me the basics about this product" which could be
 268 publicly available information or much more complicated services such as "subscribe me
 269 to a service that will tell me if this product is ever recalled". In the example above the
 270 'EPC' portion is used to differentiate this record from other possible NAPTR records. The
 271 'pml' portion is what is used to designate the service class. See the Examples section for
 272 other possible types of services. See [Appendix C](#) for a list of Services that make up an
 273 initial set of registered service classes.

274 The **Regexp** field is used to specify a URI for the service being described. In previous
 275 versions of ONS the result was simply an IP address. This proved insufficient due to the
 276 needs of protocols such as SOAP that are layered over HTTP. In nearly all modern
 277 protocols there is a need for a hostname and additional 'path' information. The reason the
 278 field is in the form of a regular expression is that the NAPTR record is used by other
 279 applications that have the need to conditionally rewrite the URI to include other
 280 information. While none of the examples here make use of this feature, it has not been
 281 determined if this will always be the case. In the future it may become necessary to allow
 282 full regular expression and replacement functions within the regexp field. Therefore,

283 implementers would be wise to not assume that the URI can simply be extracted without
284 any regular expression processing.

285 It is important to note that the Regexp field is a Posix Extended Regular Expression. This
286 form states that the first character encountered becomes the field delimiter between the
287 regular expression and the replacement portion of the entire rewrite expression. In the
288 above example the delimiter is the '!' character. The regular expression portion is '^.*\$'
289 which equates to 'match anything'. The replacement portion is 'http://company.com/cgi-
290 bin/pmlservice'. The choice of '!' as the delimiter instead of a more traditional '/' makes
291 the entire line much easier to read and less error prone.

292 7 Examples

293 In the following examples the EPC in question is urn:epc:1.47400.11015.583865 which
294 represents a disposable Gillette Mach 3. The client application attempts to learn about
295 this product by first turning the EPC into a domain-name: 11015.47400.1.onsroot.org

296
297 The application then queries the DNS for NAPTR records for that domain-name and
298 receives the following records:

Order	Pref	Flags	Service	Regexp	Replacement
0	0	u	EPC+ws	!^.*\$!http://gillette.com/autoid/sensor3.wsdl!	.
0	0	u	EPC+pml	!^.*\$!http://gillette.com/autoid/cgi-bin/pml.php!	.
0	0	u	EPC+html	!^.*\$!http://www.gillette.com/products/grooming_men.asp!	.
0	0	u	EPC+xmlrpc	!^.*\$!http://gateway1.xmlrpc.com/servlet/gillette.com!	.
0	1	u	EPC+xmlrpc	!^.*\$!http://gateway2.xmlrpc.com/servlet/gillette.com!	.

299 7.1 Finding a WSDL file for a product

300 One of the simplest but most powerful examples is where an ERP application is Web
301 Services [\[Web Services\]](#) enabled. This particular application is capable of learning
302 about new products by making various application specific web services calls to public
303 interfaces made available by the manufacturer. In this case the application can simply use
304 the ONS to look to see if a WSDL file exists that describes the Web Services it requires.

305 The application issues the query and receives the NAPTR records above. It iterates
306 through the list looking for the 'ws' service which designates a WSDL file that defines the
307 available web services. It locates that service in the first record and returns the URI found
308 in the Regexp field. The application then hands that URI to its Web Services engine to
309 determine if the proper end points exist. If they do then the application requests the
310 metadata it needs and proceeds with its processing.

311 7.2 Finding a PML server for a product

312 This example shows how an EPC can be used to retrieve PML documents about a
313 product. The PML information could be in the PML Core vocabulary (uninterpreted

314 sensor observations), or could be in some other PML vocabulary defined by EPCglobal.
315 Again, using the same results from above, the Savant uses the second record and extracts
316 the URI from the Regexp. It then uses that URI as the PML end point to persist the given
317 event.

318 **7.3 Finding an HTML formatted web page description of a** 319 **product**

320 This example shows how a very lightweight service can be deployed almost immediately
321 using nothing more than an existing externally available corporate web server containing
322 existing product content. By inserting the third record in the results list above, Gillette
323 can easily point applications to authoritative product data without any modifications to
324 their systems. Applications that understand this service can be very lightweight, using
325 existing web browsers to display the content.

326 **7.4 Finding an XML-RPC gateway to the Web Service interfaces**

327 In some cases a service may be outsourced. In this example Gillette has decided not to
328 expose an XML-RPC [\[XML-RPC\]](#) service of its own. Instead there is an XML-RPC to
329 SOAP gateway run by a third party. In the interest of interoperability Gillette simply adds
330 an ONS entry that points to this gateway, thus enabling new applications with little effort
331 on their part.

332 In this case there are two records for this service and both have the same Order value.
333 This means that the Pref field is used to indicate a preference for one over the other (load
334 balancing and fail-over). If for some reason the record with the Pref of '0' fails or is busy,
335 the one with the Pref of '1' can be used.

336 **8 References**

337 **Willinger and Doyle**

338 Willinger, W. and Doyle, John. *Robustness and the Internet: Design and*
339 *evolution*, 2002. (See http://netlab.caltech.edu/pub/papers/part1_vers4.pdf)

340 **EPC**

341 Brock, David. *The Electronic Product Code (EPC): A Naming Scheme for*
342 *Physical Objects*, 2001. (See
343 <http://www.autoidcenter.org/publishedresearch/MIT-AUTOID-WH-002.pdf>)

344 **RFC 2826**

345 (Internet Architecture Board). *RFC 2826: IAB Technical Comment on the Unique*
346 *DNS Root*, 2000. (See <http://www.ietf.org/rfc/rfc2826.txt>)

347 **DNS**

348 (Internet Engineering Task Force). *RFC 1034: Domain names, Concepts and*
349 *facilities*, ed Mockapetris, P. 2000. (See <http://www.ietf.org/rfc/rfc1034.txt>)

350 **Web Services**

351 (WorldWideWeb Consortium). *Web Services Activity*, 2000. (See
352 <http://www.w3.org/2002/ws/>)

353 **XML-RPC**

354 Winer, Dave. *XML-RPC Specification*, 1999. (See <http://www.xml-rpc.com/spec>)

355 **RFC 2396**

356 T. Berners-Lee, R. Fielding, L. Masinter. *Uniform Resource Identifiers (URI):*
357 *Generic Syntax*, 1998. (See <http://www.ietf.org/rfc/rfc2396.txt>)

358 **RFC 3401**

359 Mealling, Michael. *Dynamic Delegation Discovery System (DDDS) Part One:*
360 *The Comprehensive DDDS*, 2002. (See <http://www.ietf.org/rfc/rfc3401.txt>)

361 **RFC 3402**

362 Mealling, Michael. *Dynamic Delegation Discovery System (DDDS) Part Two:*
363 *The Algorithm*, 2002. (See <http://www.ietf.org/rfc/rfc3402.txt>)

364 **RFC 3403**

365 Mealling, Michael. *Dynamic Delegation Discovery System (DDDS) Part Three:*
366 *The Domain Name System (DNS) Database*, 2002. (See
367 <http://www.ietf.org/rfc/rfc3403.txt>)

368 **9 Appendix A -- Glossary**

369 **Auto-ID**

370 "Automatic Identification" -- An open, global network that can identify anything,
371 anywhere, automatically.

372 **Domain-name**

373 A hierarchical, 'dot' (.) separated namespace used to identify hosts on the Internet

374 **DNS**

375 See Domain Name Service

376 **Domain Name Service**

377 An infrastructure level Internet service used to discover information about a
378 domain name. It was originally developed to map a host name to an IP address,
379 but has since been extended to other uses (such as ENUM, which maps a phone
380 number to one or more communication services.

381 **EPC**

382 See Electronic Product Code

383 **Electronic Product Code**

384 An abstract namespace made up of a EPC Manager code, an Object Class code,
385 and a Serial Number.

386 **EPC Manager code**
387 A code that identifies a manufacturer of objects

388 **ONS**
389 See Object Name Service

390 **ONS Root**
391 The domain-name that is appended to the manager id and which acts as the ‘top’
392 of the DNS tree that contains all of the EPC entries. This document uses the value
393 of ‘onsroot.org’ but this value has not been agreed upon and will be specified by
394 EPC Global, Inc.

395 **Object Name Service**
396 A resolution system, based on DNS, for discovering authoritative information
397 about an EPC

398 **Object Class code**
399 A code that identifies a particular type of object that is created by a particular
400 manufacturer

401 **NAPTR**
402 "Naming Authority PoinTer" -- A DNS record type (35) that contains information
403 about a specific delegation point within some other namespace using regular
404 expressions.

405 **PML**
406 "Physical Markup Language" -- generally information obtained from a Reader or
407 similar sensor

408 **Reader**
409 A radio enabled device that communicates with a TAG

410 **RFID**
411 "Radio Frequency Identification" -- A method of identifying unique items using
412 radio waves. The big advantage over bar code technology is lasers must see a bar
413 code to read it. Radio waves do not require line of site and can pass through
414 materials such as cardboard and plastic.

415 **Regular Expression**
416 A standard language for pattern matching within a string of characters and for
417 composing new strings based on matched subcomponents of the original string
418 (i.e. a search and replace function)

419 **Savant**
420 Distributed network software that manages and moves data related to Electronic
421 Product Codes.

422 **Serial Number**

423 A number that identifies a particular instance of an object class.

424 **TAG**

425 A microchip and antenna combo that is attached to a product. When activated by
426 a TAG Reader the TAG emits its EPC

427 **URI**

428 "Uniform Resource Identifier" -- the superclass of all identifiers that follow the
429 'scheme:scheme-specific-string' convention as specified in RFC 2396 [\[RFC2396\]](#)
430 (e.g., "urn:isbn:2-9700369-0-8" or "http://example.com/news.html")

431 **URL**

432 "Uniform Resource Locator" -- A deprecated term usually used to denote the
433 subclass of URIs that contain DNS domain-names in their authority section.

434 **10 Appendix B -- DDDS Application Specification**

435 The use of NAPTR records is governed by a series of RFCs that define something called
436 the Dynamic Delegation Discovery Service. RFC 3401 [\[RFC 3401\]](#) is the first in the
437 series and provides an introduction to the series. In order to safely use NAPTR records on
438 the public network a specification must exist that describes the values of the various
439 fields. This appendix contains that specification which, when approved by the SAG
440 process, will be extracted and published as an RFC itself.

441 **10.1 Application Unique String**

442 The Application Unique String is the EPC in URI form.

443 **10.2 First Well Known Rule**

444 The First Well Known Rule is the identify function. The output of this rule is the same as
445 the input. This is because the EPC namespace and this Applications databases are
446 organized in such a way that it is possible to go directly from the name to the smallest
447 granularity of the namespace directly from the name itself.

448 **10.3 Expected Output**

449 The output of the last Rewrite Rule is a URI and a Service designator that, together,
450 designate an application context (server and application) that will expose some metadata
451 or services about the EPC.

452 **10.4 Valid Databases**

453 At present only one DDDS Database is specified for this Application. RFC 3403 [\[RFC](#)
454 [3403\]](#) specifies a DDDS Database that uses the NAPTR DNS resource record to contain
455 the rewrite rules. The Keys for this database are encoded as domain-names. The
456 conversion method for this database is as follows:

- 457 1. Remove the 'urn:epc:' header
- 458 2. Remove the serial number field

- 459 3. Invert the order of the remaining fields
460 4. Append the root domain

461 At the time of this publication the value of the root domain is unknown.

462 **10.5 Valid Flags**

463 The 'u' flag which denotes that the current Rule is terminal and that the output of the
464 Regexp is a URI.

465 **10.6 Service Parameters**

466 The Service parameters for this Application take the form of a string of characters with
467 the following ABNF:

```
468           service_field = "EPC+" service_name  
469           service_name = ALPHA *31ALPHANUM
```

470 The valid values for 'service_name' are found in the ONS Service Registry discussed in
471 Appendix C.

472 **11 Appendix C -- Service Field Registrations**

473 The 'service_name' portion of the Service field is a managed space where the values that
474 are available for use are found with some registration authority, probably EPCGlobal, Inc.
475 The goal is to balance the ability to innovate with new services with the desire for
476 interoperability between services. If the service_name simply ends up denoting non-
477 interoperable, proprietary services then the total value of the system is significantly
478 reduced. Conversely, limiting all services to simply those that are standardized by the
479 AutoID standards process limits the ability of the system to innovate.

480 To balance these requirements a special sub-class of services is created which start with
481 an 'x-'. Services of this type do not need to be registered but merely act as a designation
482 of "experimental status". Implementors should not create services in this class and then
483 never register them.

- 484 ➤ At this point registrations must be approved by the SAG standards approval process.
485 It is an open issue as to whether or not this should be relaxed or not.

486 **11.1 Registration Template**

487 A document specifying a service_name must contain the following template. This
488 template is then entered into the service_name registry as soon as the document is
489 published.

490 **Service Name:**

491 The exact sequence of characters that will appear in the Service field

492 **Functional Specification:**

493 A pointer to publicly available documentation for the service.

494 **Valid URI schemes:**

495 A list of registered URI schemes that are valid for this service (e.g. web services
496 can be specified using either the 'http:' or 'https:' scheme)

497 **Security Considerations:**

498 Any security issues associated with use of this service

499 **11.2 Service Registrations**

500 The following are initial services that will be registered as soon as this document is
501 published as a specification:

- 502 • Service Name: pml

503 **Functional Specification:** The PML Service Specification

504 **Valid URI schemes:** http, https

505 **Security Considerations:** See the Security Considerations section of the functional
506 specification.

507

- 508 • Service Name: ws

509 **Functional Specification:** A generic Web Services [\[Web Services\]](#) based service
510 where the application must negotiate what services are available by investigating the
511 WSDL file found at the URI in the Regexp

512 **Valid URI schemes:** http, https

513 **Security Considerations:** Web Services utilise a great deal of existing Internet
514 infrastructure and protocols. It is very easy to use some of them in insecure ways.

515 Any usage of Web Services should be done in the context of a thorough
516 understanding of the dependencies, especially as it relates to the DNS and HTTP.

517

- 518 • Service Name: html

519 **Functional Specification:** Simply returns a URI that will resolve to an HTML page
520 on some server. The assumption is that this page contains information about the
521 product in question.

522 **Valid URI schemes:** http, https, ftp

523 **Security Considerations:** None not already inherent in the use of the
524 WorldWideWeb.

525

- 526 • Service Name: xmlrpc

527 **Functional Specification:** A URI that denotes an HTTP POST capable service on
528 some server that is expecting XML-RPC [\[XML-RPC\]](#) compliant connection.

529 **Valid URI schemes:** http, https

530 **Security Considerations:** None not already inherent in the use of the
531 WorldWideWeb.

532