Service Location Working Group   Erik Guttman
INTERNET DRAFT   Charles Perkins
                                                                James Kempf
31 October 1997   Sun Microsystems

Service Templates and service: Schemes

draft-ietf-svrloc-service-scheme-05.txt

Status of This Memo

Abstract

The ''service:'' URL scheme name is used to define URLs (called ''service: URLs'' in this document) that are primarily intended to be used by the Service Location Protocol in order to distribute service access information. These schemes provide an extensible framework for client-based network software to obtain configuration information required to make use of network services. When registering a service: URL, the URL SHOULD be accompanied by a set of well-defined attributes which define the service. These attributes SHOULD convey configuration information to client software, or service characteristics meaningful to end users.

This document describes a formal procedure for defining and standardizing new service types and attributes for use with the

''service:'' scheme. The formal descriptions of service types and attributes are templates that are human and machine understandable. They SHOULD be used by administrative tools to parse service registration information and by client applications to provide localized translations of service attribute strings.

Contents

1. Introduction

This document describes a URL scheme, called service: URL, which defines network access information for network services using a formal notation. In addition it describes how to define a set of attributes to associate with a service: URL. These attributes will allow end users and programs to select between network services of the same type that have different capabilities. The attributes are defined in a template document that is readable by people and machines.

A client uses attributes to select a particular service. Service selection occurs by obtaining the service: URL that offers the right configuration for the client. Service type templates define the syntax of service: URLs for a particular service type, as well as the attributes which accompany a service: URL in a service registration.

Templates are used for the following distinct purposes:

1. Standardization

   The template is reviewed before it is standardized. Once it is standardized, all versions of the template are archived by IANA.

2. Service Registration

   Servers making use of the Service Location Protocol [15] register themselves and their attributes. They use the templates to generate the service registrations. In registering, the service must use the specified values for its attributes.

3. Client presentation of Service Information

   Client applications may display service information. The template provides type information and explanatory text which may be helpful in producing user interfaces.

4. Internationalization

   Entities with access to the template for a given service type in two different languages may translate between the two languages.

   A service may register itself in more than one language using templates, though it has been configured by an operator who registered service attributes in a single language.

All grammar encoding follows the Augmented BNF (ABNF) [9] for syntax specifications.

1.1. Terminology

This section introduces some terminology for describing service: URLs.

service scheme

A URL scheme whose name starts with the string "service:" and is followed by the service type name, constructed according to the rules in this document. An example is "service:lpr:" for the lpr print service [14].

service: URL

A URL constructed according to the service scheme definition. It typically provides at least the following: The name of an access protocol, and an address locating this service. The service: URL may include url path information specific to the type of service, as well as attribute information encoded according to the URL grammar. The service: URL is used by the Service Location Protocol to register and discover the location of services. It may be used by other protocols and in documents as well.

service type

A name identifying the semantics by which the remainder of the service: URL is to be understood. It may denote either a particular network protocol, or an abstract service associated with a variety of protocols. If the service type denotes a particular protocol, then the service type name SHOULD either be assigned the name of a particular well known port [3] by convention or or be the Assigned Numbers name for the service [1].

abstract service type

A service type name which is associated with a variety of different protocols. An example is given in Section A. Section 3 discusses various ways that abstract types can be accommodated.

service registration

A service: URL and optionally a set of attributes comprise a service registration. This registration is made by or on behalf of a given service. The URL syntax and attributes must conform to the service template for the registered service.

service template

A formal description of the service attributes and service scheme associated with a particular service type.

1.2. Service Location Protocol

The Service Location Protocol [15] allows service: URLs to be registered and discovered, though service: URLs may be also used in other contexts.

Client applications discover service registrations by issuing queries for services of a particular type, specifying the attributes of the service: URLs to return. Clients retrieve the attributes of a particular service by supplying its service: URL. Attributes for all service registrations of a particular type can also be retrieved.

Services may register themselves, or registrations may be made on their behalf. These registrations contain a service: URL, and possibly attributes and digital signatures.

2. Related work

The "Finding Stuff" work by Ryan Moats, Martin Hamilton, and Paul Leach uses service: URLs to provide access information about arbitrary network protocols through DNS [11]. DNS SRV Resource Records are a mechanism which provides a way to obtain a service by type for a given domain [10], without specifying which instance of the service type meet particular requirements.

3. Service URL Syntax and Semantics

This section describes the syntax and semantics of service: URLs.

3.1. Service URL Syntax

The syntax of the service: URL MUST conform to [6]. The only exception is that the <password> field has been omitted from the

<site> production, since plain text transmission of passwords is now discouraged. Note that the syntax for the <sap> field depends upon the service type definition. The <sap> field is the service access point, and describes how to access the service. In addition, although both upper case and lower case characters are recognized in the <service-type> field for convenience, the name is case-folded into lower case. Service types are therefore not distinguished on the basis of case, so, for example, "http" and "HTTP" designate the same service type. This is consistent with general URL practice, as outlined in [7].

The ABNF for a service: URL is:

```
     service: URL   =   "service:" service-type ":" sap
     service-type   =   abstract-type ":" url-scheme / concrete-type
     abstract-type  =   type-name [ "." naming-auth ]
     concrete-type  =   protocol [ "." naming-auth ]
     type-name      =   resname
     naming-auth    =   resname
     url-scheme     =   resname
                        ; A recognized URL scheme name, standardized
                        ; either through common practice or through
                        ; approval of a standards body.
     resname        =   alpha [ 1*(alpha / digit / "+" / "-") ]
     sap            =   "//" site [ url-part ]
     site           =   [ [ user "@" ] hostport ]
     hostport       =   host [ ":" port ]
     host           =   hostname / hostnumber
     hostname       =   *( domainlabel "." ) toplabel
     alphanum       =   alpha / digit
     domainlabel    =   alphanum / alphanum *[alphanum / "-"] alphanum
     toplabel       =   alpha / alpha *[ alphanum / "-" ] alphanum
     hostnumber     =   ipv4-number / ipv6-number
     ipv4-number    =   1*3digit 3("." 1*3digit)
     ipv6-number    =   32hex
     3digit         =   digit digit digit
     port           =   1*digit
                        ; A port number must be included if the
                        ; protocol field does not have an IANA
                        ; assigned port number.
     user           =   *[ uchar / ";" / "+" / "&" / "=" ]
     url-part       =   [ url-path ] [ attr-list ]
     url-path       =   1 * ( "/" *xchar )
                        ; Each service type must define its
                        ; own syntax consistent
                        ; with [6].
     attr-list      =   1 * ( ";" attr-asgn )
```

```
        attr-asgn      =   attr-id / attr-id "=" attr-value
        safe           =   "$" / "-" / "_" / "." / "~"
        extra          =   "!" / "*" / "'" / "(" / ")" / "," / "+"
        uchar          =   unreserved / escaped
        xchar          =   unreserved / reserved / escaped
        escaped        =   "%" hex hex
        hex                "a" / "b" / "c" / "d" / "e" / digit
        reserved       =   ";" / "/" / "?" / ":" / "@" / "&" / "=" / "+"
        unreserved     =   alpha / digit / safe / extra
```

Certain characters must be escaped before use. To escape any character, precede the two digits indicating its ASCII value by '%'.

3.2. Service URL Semantics

The service scheme-specific information following the "service:" URL scheme identifier provides information necessary to access the service. As described in the previous subsection, the form of a service: URL is as follows:

```
service: URL = "service:" service-type ":" sap
```

where <sap> has the following form:

```
//addr-spec/url-path;attr-list
```

The <service-spec> field includes the <service-type> field. As discussed in Section 1, the <service-type> can be either a concrete protocol name, or an abstract type name.

The <service-part> field includes a site specification (the <site> field) in the format specified by [6]. The <site> field is typically either a domain name (DNS) or an IP network protocol address for the service, and possibly a port number. Note that use of DNS hostnames is preferred for ease of renumbering. The <site> field can be null if other information in the service URL or service attributes is sufficient to use the service.

The <sap> field allows more information to be provided (by way of <url-path> and <attr-list>) that can uniquely locate the service or resource if the <addr-spec> is not sufficient for that purpose.

An <attr-list> field appears at the end of the <url-part> field, but is never required to exist in any service location registration. The <attr-list> field is composed of a list of semicolon (";") separated attribute assignments of the form:

```
attr-id "=" attr-value
```

or for keyword attributes:

```
   attr-id
```

Attributes are part of service: URLs when the attributes are required to access a particular service. For instance, an ACAP [13] service might require that the client authenticate with it through Kerberos. Including an attribute in the service registration allows the ACAP client to make use of the correct SASL [12] authentication mechanism. The ACAP server's registration might look like:

```
   service:acap://some.where.net;authentication=KERBEROSV4
```

Note that there can be other attributes of an ACAP server which are not be appropriate to include in the URL. For instance, the list of users who have access to the server is useful for selecting an ACAP server, but is not required for a client to use the registered service.

Attributes associated with the service: URL are not typically included in the service: URL. They are stored and retrieved using other mechanisms. The service: URL is uniquely identified with a particular service agent or resource, and is used when registering or requesting the attribute information. The Service Location Protocol specifies how such information SHOULD be registered by network services and obtained by client software.

3.3. Use of service: URLs

The service: URL is intended to allow arbitrary client/server and peer to peer systems to make use of a standardized dynamic service access point discovery mechanism.

It is intended that service: URLs be selected according to the suitability of associated attributes. A client application can obtain the URLs of several services of the same type and distinguish the most preferable among them by means of their attributes. The client uses the service: URL to communicate directly to a service.

Attributes are specified with a formal service template syntax described in Section 4. If a service: URL registration includes attributes, the registering agent SHOULD also keep track of the attributes which characterize the service.

Registrations can be checked against the formal attribute specification defined in the template by the client or agent representing the client. Service registration are typically done using the Service Location Protocol [15] (SLP). SLP provides a

mechanism for service: URLs to be obtained dynamically, according to the service's attributes.

It is also possible to obtain service: URLs from documents and using other protocols. In this case, the URL may not be accompanied by the service attributes. The context in which the URL appears SHOULD make it clear, if possible, when the service is appropriate to use. For example, in a mail message, a service might be recommended for use when the user is in a branch office. Or, an HTML document might include a service: URL as a pointer to a service, describing in text what the service does and who is authorized to use it.

3.4. Specifying the Service Type-Specific URL Syntax

When a service type is specified, the specification includes the definition of the syntax for all URLs that are registered by services of that particular type. For instance, the "lpr" service type may be defined with service: URLs in the following form:

```
service:printer:lpr://<address of printer>/<queue name>
```

The section of the URL after the address of the printer:

```
"/" <queue name>
```

is specific to the lpr service type and corresponds to the <url-path> field of the general service: URL syntax. This part is specified when the lpr service type is specified.

3.5. Accommodating Abstract Service Types

An abstract service type is a service type that can be implemented by a variety of different service agents.

In order to register an service: URL for an abstract service type the 'abstract-type' grammar rule described in section 3.1 is used. This will result in a URL which includes enough information to use the service, namely, the protocol, address and path information. Unlike 'concrete' service: URLs, however, the service type is not enough to determine the service access. Rather, an abstract service type denotes a class of service types. The following subsection discusses this point in more detail.

3.5.1. Advertising Abstract Service Types

Some services may make use of several protocols that are in common use and are distinct services in their own right. In these cases an abstract service type is appropriate. What is essential is that all the required information for the service is clearly defined.

For example, suppose a network service is being developed for dynamically loading device drivers. The client requires the following three pieces of information before it can successfully load and instantiate the driver:

1. The protocol used to load the driver code, for example, "ftp", "http" or "tftp"

2. A pathname identifying where the driver code is located, for example "/systemhost/drivers/diskdrivers.drv",

3. The name of the driver, for example, "scsi".

The temptation is to form the first two items into a URL and embed that into a service: URL. As an example which should be avoided,

```
service:ftp:/x3.bean.org/drivers/diskdrivers.drv;driver=scsi
```

is a service: URL which seems to indicate where to obtain the driver.

Rather, an abstract service-type SHOULD be used. The service type is not "ftp", as the example indicates. Rather, it is "device-drivers". The service: URL that should be used, consistent with the rules in section [6], is the following:

```
service:device-drivers:ftp://x3.bean.org/drivers/diskdrivers.drv;
driver=scsi;platform=sys3.2-rs3000
```

Other URLs for the same service using other protocols are also supported, as in:

```
service:device-drivers:tftp://x2.bean.org/vol3/disk/drivers.drv;
driver=scsi;platform=sys3.2-rs3000

service:device-drivers:http://www.bean.org/drivers/drivpak.drv;
driver=scsi;platform=sys3.2-rs3000
```

Using SLP, a search for the service type "device-drivers" may return all of the three URLs listed above. The client selects the most appropriate access protocol for the desired resource.

The fundamental requirement is that the abstract service type MUST be well specified. This requirement is imposed so that program code or human users have enough information to access the service. In every case, a well-specified abstract type will include either an access protocol and a network address where the service is available, or an embedded URL for a standardized URL scheme that describes how to access the service. In the example above, there are three further requirements: A URL path is included for access protocols indicating the document to download, and two attributes are included to characterize the driver.

4. Syntax and Semantics of Service Type Specifications

Service type specifications are documents in a formal syntax defining properties important to service registration. These properties are:

1. General information on the service type specification itself,

2. The syntax of the service type-specific part of the service URL,

3. The definition of attributes associated with a service.

The service type specification document is the service type template.

The following subsections describe the syntax and semantics of service type templates.

4.1. Syntax of Service Type Templates

Service template documents are encoded in a simple form. They may be translated into any language or character set, but the template used for standardization MUST be encoded in UTF8 [16] and be in English.

A template document begins with a block of text assigning values to five document identification items. The five identification items can appear in any order within the block, but conventionally the "type" item, which assigns the service type name, occurs at the very top of the document in order to provide context for the rest of the the document. The attribute definition item occurs after the document identification items.

All items end with a blank line. The reserved characters are ";", "%", "=", ",", "#", LF, and CR. Reserved characters MUST be escaped. The escape sequence is the same as described in [6].

The service template is encoded in a UTF8 character set, but submitted as a part of an internet-draft, which is encoded in ASCII

characters. All characters which are outside of the ASCII range MUST be escaped using the % HEX HEX syntax. For example, the letter e accent aigue would be represented as "%c3%a9". Unfortunately, this will detract from the readability of the service template in the internet draft. Hopefully some public domain tools will emerge for translating escaped UTF8 characters into humanly readable ones.

Values in value lists are separated by commas. A value list is terminated by a newline not preceded by a comma. If the newline is preceded by a comma, the value list is interpreted to continue onto the next line.

Attribute identifiers, attribute type names, and flags are all case insensitive. For ease of presentation, upper and lower case characters can be used to represent these in the template document. Newlines are significant in the grammar. They delimit one item from another, as well as separating parts of items internally.

String values are considered to be a sequence of non-whitespace tokens potentially with embedded whitespace, separated from each other by whitespace. Commas delimit lists of strings. String values are trimmed so as to reduce any sequence of white space interior to a string to a single white space. Preceding or trailing white space is removed. For example:

```
      " some value , another example "

   is trimmed to

      "some value" and "another example".
```

Note that there can be no ambiguity in string tokenization because values in value lists are separated by a comma. String tokens are not delimited by double quotes (") as is usually the case with programming languages.

Attribute tags and values are useful for directory look-up. In this case, decoding of character escapes and trimming white space MUST be performed before string matching. In addition, string matching SHOULD be case insensitive.

Templates obey the following ABNF [9] grammar:

```
   template      =  tem-attrs attr-defs
   tem-attrs     =  schemetype schemevers schemelang
                    schemetext schemeurl
   schemetype    =  "type" "=" scheme termdef
   schemevers    =  "version" "=" version-no termdef
```

```
      schemelang     =  "language" "=" isolang termdef
      schemetext     =  "description" "=" newline desc-text termdef
      schemeurl      =  "url-syntax" "=" newline url-bnf termdef
      url-bnf        =  *[ com-chars ]
                        ; An ABNF describing the <url-path> production
                        ; in the service: URL grammar of Section 3.1.
      scheme         =  service-type [ "." naming-auth ]
      service-type   =  scheme-name
      naming-auth    =  scheme-name
      scheme-name    =  alpha [1*schemechar] [ "." 1*schemechar ]
      schemechar     =  alpha / digit / "-" / "+" /
      version-no     =  1*digit "." 1*digit
      isolang        =  2*3lower-alpha ;see [4]
      desc-text      =  *[ com-chars ]
                        ; A block of free-form text for reading by
                        ; people describing the service in a short,
                        ; informative manner.
      termdef        =  newline newline
      attr-defs      =  *( attr-def / keydef )
      attr-def       =  id "=" attrtail
      keydef         =  id "=" "keyword" newline [help-text] newline
      attrtail       =  type flags newline [value-list] [help-text]
                        [value-list] newline
      id             =  1*attrchar
      type           =  "string" / "integer" / "boolean" / "opaque"
      flags          =  ["m"/"M"] ["l"/"L"] ["o"/"O"] ["x"/"X"]
      value-list     =  value newline / value "," value-list /
                        value "," newline value-list
      help-text      =  1*( "#" help-line )
                        ; A block of free-form text for reading by
                        ; people describing the attribute and
                        ; its values.
      help-line      =  *[ com-chars ] newline
      attrchar       =  schemechar / ":" / "_" / "$" / "~" / "@" / "." /
                        "|" / "<" / ">" / "*" / "&"
      value          =  string / integer / boolean / opaque
      string         =  safe-char *[safe-char / white-sp] safe-char
      integer        =  [ "+" | "-" ] 1*digit
      boolean        =  "true" / "false"
      opaque         =  1*digit ":" 4*radix64-char
                        ; The digits define the original length of
                        ; the opaque value.  The restricted character
                        ; string is the radix-64 encoding of the
                        ; opaque value( [8], Sect.  6.8.)
                        ; Newlines are ignored in decoding radix-64
                        ; values.
      com-chars      =  safe-char / white-sp / "," / ";"/ "%"
      safe-char      =  attrchar / escaped / " " / "!" / '"' / "'" /
```

```
                       "|" / "(" / ")" / "+" / "-" / "." / ":" /
                       "=" / "?" / "[" / "]" / "{" / "/" / "{" /
                       "$"
                       ; All UTF8 printable characters are
                       ; included except ",", "%", ";", and "#".
      escaped       =  "%" hex hex
      hex           =  digit / "A" / "B" / "C" / "D" / "E" /
                       "a" / "b" / "c" / "d" / "e"
      white-sp      =  space / tab
      newline       =  CR / ( CR LF )
```

4.2. Semantics of Service Type Templates

The service type template defines the service attributes and service: URL syntax for a particular service type. The attribute definition includes the attribute type, default values, allowed values and other information.

4.2.1. Definition of a Service Template

There are six items included in the service template. The semantics of each item is summarized below.

- type

  The scheme name of the service scheme. The scheme name consists of the service type name and an optional naming authority name, separated from the service type name by a period. See 4.2.2 for the conventions governing service type names.

- version

  The version number of the service type specification.

- language

  The language of the service type specification.

- description

  A description of the service suitable for inclusion in text read by people.

- url-syntax

  The syntax of the service type-specific URL part of the service: URL.

- attribute definitions

  A collection of zero or more definitions for attributes associated with the service in service registrations.

Each of the following subsections deals with one of these items.

4.2.2. Service Type

The service scheme consists of the service type name and an optional naming authority name separated from the service type name by a period. The service scheme is a string that is appended to the 'service:' URL scheme identifier, and is the value of the "type" item in the template document. If the naming authority name is absent it is assumed to be IANA.

4.2.3. Service Type Language

The service type language is a RFC 1766 Language Tag defining the language of the template [4] and is the value of the "language" item.

4.2.4. Version Number

The version number of the service type template is the value of the "version" item. A draft proposal starts at 0.0, and the minor number increments once per revision. A standardized template starts at 1.0. Additions of optional attributes add one to the minor number, and additions of required attributes, changes of definition, or removal of attributes add one to the major number. The intent is that an old service template still accurately, if incompletely, defines the attributes of a service registration if the template only differs from the registration in its minor version. See Section 5 for more detail on how to use the version attribute.

4.2.5. Description

The description is a block of text readable by people in the language of the template and is the value of the "description" item. It should be sufficient to identify the service to human readers and provide a short, informative description of what the service does.

If the service type corresponds to a particular protocol, the protocol specification must be cited here. The protocol need not be a standardized protocol. The template might refer to a proprietary

specification, and refer the reader of the template to a contact person for further information.

4.2.6. Syntax of the Service Type-specific URL Part

The syntax of the service type-specific part of the service: URL is provided in the template document as the value of the "url-syntax" item. The <url-path> field of the service: URL is designed to provide additional information to locate a service when the <addr-spec> field is not sufficient. The <url-path> field distinguishes URLs of a particular service type from those of another service type. For instance, in the case of the lpr service type, the <url-path> must include the queue name [14], but other service types may not require this information.

The syntax for the <url-path> field MUST accompany the definition of a new service type, unless the URL scheme has already been standardized and is not a service: URL. The syntax is included in the template document as an ABNF [9] following the rules for URL syntax described in [6]. There is no requirement for a service scheme to support a <url-path>. The <url-path> field can be very simple, or even omitted. If the URL scheme has already been standardized, the "url-syntax" item SHOULD include a reference to the appropriate standardization documents. Abstract service types may defer this field to the template documents describing their concrete instances.

4.2.7. Attribute Definition

The bulk of the template is typically devoted to defining service type-specific attributes. An attribute definition precisely specifies the attribute's type, other restrictions on the attribute (whether it is multi-valued, optional, etc), some text readable by people describing the attribute, and lists of default and allowed values. The only required information is the attribute's type, the rest are optional. Registration, deregistration and the use of attributes in queries can be accomplished using the Service Location Protocol [15] or other means, and discussion of this is beyond the scope of the document.

Attributes are used to convey information about a given service for purposes of differentiating different services of the same type. They convey information to be used in the selection of a particular service to establish communicate with, either through a program offering a human interface or programmatically. Attributes can be encoded in different character sets and in different languages. The procedure for doing this is described in Section 7.

An attribute definition begins with the specification of the attribute's identifier and ends with a single empty line. Attributes definitions have five components (in order of appearance in a definition):

1. An attribute identifier which acts as the name of the attribute,

2. Attribute descriptors (type and flags),

3. An optional list of values which are assigned to the attribute by default,

4. An optional block of text readable by people providing a short, informative description of the attribute,

5. An optional list of allowed values which restrict the value or values the attribute can take on.

4.2.7.1. The Attribute Identifier

An attribute definition starts with the specification of the attribute's identifier. The attribute's identifier functions as the name of the attribute. Note that the characters used to compose an attribute identifier are restricted to those characters considered unrestricted for inclusion in a URL according to [6]. The reason is that services can display prominent attributes in their service: URL registrations. Each attribute identifier must be unique in the template. Since identifiers are case folded, upper case and lower case characters are the same.

4.2.7.2. The Attribute Type

Attributes can have one of five different types: string, integer, boolean, opaque, or keyword. The attribute's type specification is separated from the attribute's identifier by an equal sign ("=") and follows the equal sign on the same line. The string, signed integer, and boolean types have the standard programming language or database semantics. Integers are restricted to those signed values that can be represented in 32 bits. The character set used to represent strings is not specified at the time the template is defined, but rather is determined by the service registration. Booleans have the standard syntax. Opaques are radix64 numbers [8] that can be used to represent any other kind of data. Keywords are attributes that have no characteristics other than their existence (and possibly the descriptive text in their definition).

Keyword and boolean attributes impose restrictions on the following parts of the attribute definition. Keyword attribute definitions MUST have no flag information following the type definition, nor any default or allowed values list. Boolean attributes are single value only, i.e., multi-valued boolean attributes are not allowed.

4.2.7.3. Attribute Flags

Flags determine other characteristics of an attribute. With the exception of keyword attributes, which may not have any flags, flags follow the attribute type on the same line as the attribute identifier, and are separated from each other by whitespace. Flags may appear in any order after the attribute type. Other information must not follow the flags, and only one flag identifier of a particular flag type is allowed per attribute definition.

The semantics of the flags are as follows:

- o or O

  Indicates that the attribute is optional. If this flag is missing, the attribute is required in every service registration.

- m or M

  Indicates that the attribute can take on multiple values. If this flag is present, every value in a multi-valued attribute has the same type as the type specified in the type part of the attribute definition. Boolean attributes must not include this flag.

- l or L

  Indicates that attribute is literal, i.e. is not meant to be translated into other languages. If this flag is present, the attribute is not considered to be readable by people and should not be translated when the template is translated. See Section 7 for more information about translation.

- x or X

  Indicates that clients MUST specify the attribute and a value in a service query in order to narrowly focus which service: URLs are returned. The query will be rejected by DA's that utilize templates if the attribute is not included, regardless of whether the other attributes match.

The values for multivalued attributes are an unordered set. Deletions of individual values from a multivalued attribute are not supported, and deletion of the attribute causes the entire set of values to be removed.

4.2.7.4. Default Value or List

If the attribute definition includes a default value or, in the case of multivalued attributes, a default values list, it begins on the second line of the attribute definition and continues over the following lines until a line ends without a comma. As a consequence, newlines cannot be embedded in values unless escaped. See Section 3.1.

Particular attribute types and definitions restrict the default values list. Keyword attributes must not have a list of defaults. If an optional attribute's definition has an allowed values list, then a default value or list is not optional but required. The motivation for this is that defining an attribute with an allowed values list is meant to restrict the values the attribute can take on, and requiring a default value or list assures that the default value is a member of the given set of allowed values.

The default value or list indicates what values the attribute is given if no values are assigned to the attribute when a service is registered. If an optional attribute's definition includes no default value or list, the following defaults are assigned:

1. String values are assigned the empty string,

2. Integer values are assigned zero,

3. Boolean values are assigned false,

4. Opaque values are assigned a byte array containing no values,

5. Multi-valued attributes are initialized with a single value.

For purposes of translating nonliteral attributes, the default values list is taken to be an ordered set, and translations MUST maintain that order.

4.2.7.5. Descriptive Text

Immediately after the default values list, if any, a block of descriptive text SHOULD be included in the attribute definition. This text is meant to be readable by people, and should include

a short, informative description of the attribute. It may also provide additional information, such as a description of the allowed values. This text is primarily designed for display by interactive browsing tools. The descriptive text is set off from the surrounding definition by a crosshatch character ("#") at the beginning of the line. The text should not, however, be treated as a comment by parsing and other tools, since it is an integral part of the attribute definition. Within the block of descriptive text, the text is transferred verbatim, including indentation and line breaks, so any formatting is preserved.

4.2.7.6. Allowed Values List

Finally, the attribute definition concludes with an optional allowed values list. The allowed values list, if any, follows the descriptive text, or, if the descriptive text is absent, the initial values list. The syntax of the allowed values list is identical to that of the initial values list. The allowed values list is also terminated by a line that does not end in a comma. If the allowed values list is present, assignment to attributes is restricted to members of the list.

As with the default values list, the allowed values list is also considered to be an ordered set for purposes of translation.

4.2.7.7. Conclusion of An Attribute Definition

An attribute definition concludes with a single empty line.

5. A Process For Standardizing New Service Types

New service types can be suggested simply by providing a service type template and a short description about how to use the service. The template MUST have its "version" template attribute set to 0.0.

The minor version number increments once with each change until it achieves 1.0. There is no guarantee any version of the service template is backward compatible before it reaches 1.0.

Once a service template has reached 1.0, the definition is "frozen" for that version. New templates must be defined, of course, to refine that definition, but the following rules must be followed:

- Any new optional attribute defined for the template increases the minor version number by one. All other attributes for the version must continue to be supported as before. A client which

supports 1.x can still use later versions of 1.y (where x<y) as it ignores attributes it doesn't know about.

- Adding a required attribute, removing support for an attribute or changing definition of an attribute requires changing the major version number of a service template. A client application may be unable to make use of this information, or it may need to obtain the most recent service template to help the user interpret the service information.

The template should be submitted as an 'individual contribution' Internet Draft. The Internet Draft must include a 'template begins here' and 'template ends here' marking, in text, so that it is trivial to cut and paste the template from the internet draft.

A notice must be posted to the SVRLOC WG mailing list for review. Ideally, experts in the implementation and deployment of the particular protocol are consulted so as to add or delete attributes or change their definition to make the template as useful as possible. The mailing list will be maintained even when the SVRLOC WG goes dormant for the purpose of discussing service templates.

All published versions of the template must be available on-line, including obsolete ones.

Once consensus is achieved, the template should be reissued with possible corrections, having its Version number set to 1.0. If there is no comment on the template after 3 months, it should be considered to have been accepted. See Section 6 for details on how templates are submitted to an IANA registry of templates.

6. IANA Considerations

The Applications Area directors appoint a set of reviewers, including a 'lead reviewer'. Any of these reviewers may ask for clarification of a service template. If no reviewers dissent, the lead reviewer will submit the template to the IANA for inclusion in a registry. Mailing list participants supply input to the process but do not make the decision whether to accept a service template or request changes or clarifications.

The service template file has a naming convention:

```
<service-type> "." <version-no> "." <isolang>
```

Each of these fields are defined in Section 3. They correspond to the values of the template fields "type", "version" and "lang". The files for the example templates in this document

are called "acap.0.0.en", "network-management.0.0.en" and "network-management:snmp.0.0.en". See Section A.

No Internet Draft describing a service type template will be accepted unless it includes a security considerations section and contact information for the template document author.

The IANA will maintain a registry containing both the service type templates, and the template description document containing the service type template, including all previous versions. The IANA will receive notice by email from the reviewers, which will contain a reference to the Internet Draft that contains the service template. This Internet Draft will be edited to remove the Internet Draft headers and replace them with a simple header stating "This document contains a Service Type Template."

Should any trademark or copyright issues arise due to the naming of the Service Type or attributes in the Service Template, the offending names may have to be changed. The owner of the trademark may demand that this be done. The filer of the Template that requires renaming will decide the new names to use. If such issues arise, the committee of reviewers in consultation with the IESG directorate will proceed to satisfy these conditions. The IANA should simply notify the committee and they will pursue the action: The IANA is not expected to resolve trademark issues with Service Type templates.

7. Internationalization Considerations

The service: URL must be encoded using the rules set forth in [6]. The character set encoding is limited to specific ranges within the US-ASCII character set [5].

The template is encoded in UTF8 characters.

7.1. Language Identification and Translation

The language used in attribute strings should be identified using the "language" template item as defined by [4].

A program can translate a service registration from one language to another provided it has both the template of the language for the registration and the template of the desired target language. All standardized attributes are in the same order in both templates. All non-arbitrary strings, including the descriptive help text, is directly translatable from one language to another. Non-literal attribute definitions, attribute identifiers, attribute type names, attribute flags, and the boolean constants "true" and "false" are

never translated. Translation of attribute identifiers is prohibited because, as with domain names, they can potentially be part of a service: URL and therefore their character set is restricted. In addition, as with variable identifiers in programming languages, they could become embedded into program code.

All strings used in attribute values are assumed translatable unless explicitly defined as being literal, so that best effort translation (see below) does not modify strings which are meant to be interpreted by a program, not a person.

There are two ways to go about translation: standardization and best effort.

When the service type is standardized, more than one document can be submitted for review. One service type description is approved as a master, so that when a service type template is updated in one language, all the translations (at least eventually) reflect the same semantics.

If no document exists describing the standard translation of the service type, a 'best effort' translation for strings should be done.

8. Security Considerations

Service type templates provide information that is used to interpret information obtained by the Service Location Protocol. If these templates are modified or false templates are distributed, services may not correctly register themselves, or clients might not be able to interpret service information.

The service: URLs themselves specify the service access point and protocol for a particular service type. These service: URLs could be distributed and indicate the location of a service other than that normally want to used. The Service Location Protocol [15] distributes service: URLs and has an authentication mechanism that allows service: URLs of registered services to be signed and for the signatures to be verified by clients.

Each Service Template will include a security considerations section which will describe security issues with using the service scheme for the specific Service Type.

A. Service Template Examples

The text in the template example sections is to be taken as being a single file.

The ACAP example shows how to use service templates for an application that has very few attributes. Clients request the ACAP server where their user data is located by including their user name as the value of the user attribute.

The Network-Management example shows how abstract service types are defined and how a corresponding concrete instance is defined. A system might support any of several Network-Management services. Here we give only one concrete instance of the abstract type. It is not necessary to register concrete templates for an abstract service type if the abstract service type template is completely clear as to what possible values can be used as a concrete type, and what their interpretation is.

A.1. ACAP

```
-------------------------template begins here-----------------------
type=ACAP

version=0.0

lang=en

description=
  The ACAP service URL provides the location of an ACAP service.

url-syntax=
  url-path= ; There is no URL path defined for an ACAP URL.

users= string M L O
# The list of all users which the ACAP server supports.

groups= string M L O
# The list of all groups which the ACAP server supports.
--------------------------template ends here------------------------
```

The Internet Draft describing the ACAP scheme template must indicate contact information and security considerations, e.g.,

```
contact="Erik Guttman" <Erik.Guttman@sun.com>

security considerations=
  If the USER and GROUPS attributes are included a
  possibility exists that the list of identities for users or groups
  can be discovered. This information would otherwise be difficult
  to discover.
```

A.2. Abstract Service Template Example: Network-Management

The Internet Draft for the service type template contains the following text:

```
  -------------------------template begins here-----------------------
  type=Network-Management

  version=0.0

  lang=en

  description=
    This is an abstract service type.  The purpose of the network-
    management service type is to organize into a single category
    information crucial to properly managing networked hosts.  This
    will allow all network-management services of a host, as well
    as basic host configuration to be obtained by a single query
    using SLP.

  url-syntax=
    url-path=  ;  Depends on the concrete service type.
               ;  See these templates.
  --------------------------template ends here------------------------
```

In addition, the following format might be used for the needed contact and security considerations information.

```
                         ..........

  contact="Erik Guttman" <Erik.Guttman@sun.com>

  security considerations=
    See the security considerations of the concrete service types.
```

A.3. Concrete Service Template Example: Network-Management:SNMP

```
  -------------------------template begins here-----------------------
  type=service:network-management:snmp

  version=0.0

  lang=en

  description=
    The 'service:network-management:SNMP:' URL provides information
    about the SNMP manageability of a given host.  Namely, if this
    URL exists for a host (denoted by the <addr-spec> in the URL,)
```

```
    the host supports SNMP.  The path contains an enumeration of
    the MIB groups that are supported by the host.  The OID "1.3.6.1."
    is assumed as a prefix to each of the OID terms below.

  url-syntax=
    url-path     = ( [port-list] [comm-string] [oid-list])
                   ; None of the attributes listed in the URL path
                   ; are required.  They MAY be included.
    port-list    = ";ports=" port-list
    ports        = port / port "," ports
                   ; See the Service URL <port> production rule.
                   ; This field is defined as an attribute, below.
    comm-string  = ";read-community-string=" 1*uchar
                   ; See the Service URL <uchar> production rule.
                   ; This field is defined as an attribute, below.
    oid-list     = ";oids=" oids
                   ; This field is defined as an attribute, below.
    oids         = oid / oid "," oids
    oid          = DIGIT [ "." (DIGIT ".") DIGIT]

  ports=integer M L
  161
  # This attribute must be included.  It lists all ports on which
  # SNMP Agents are listening.

  read-community-string=string L O
  # The read community string may be included as an attribute of
  # a service:network-managment:snmp: URL.  This is useful in
  # cases where the community string is PUBLIC and ease of access
  # to the SNMP Agent is desired.  See the 'security considerations.'

  oid=string M L O
  # This attribute identifies a list of 'top level' MIB groups.
  # This is entirely optional, as such values can be obtained
  # directly using SNMP.  The value of including this information
  # in a service:network-managment:snmp: URL is that it will save
  # the Manager time; the MIB information can be obtained along
  # with the URL without requiring additional sequential requests
  # being sent to the managed system.
  --------------------------template ends here------------------------

                        .........

  contact="Erik Guttman" <Erik.Guttman@sun.com>

  security considerations=
    The read-community-string MUST only be included if the value
    of this string is considered to be public.  If this attribute
```

is included, absolutely anyone may access the SNMP Agent and get information from it. This may be desirable in some cases. If this string is considered confidential information, the read-community-string MUST NOT be included in the URL path nor in service registrations of the URL made through SLP or other protocols.

A.4. service: URLs and SLP

A user with an ACAP enabled email client application should not be bothered with knowing the address of their ACAP server. The mail client program can use SLP to obtain the ACAP service: URL automatically, say 'service:acap://server1.nosuch.org', by issuing a Service Request. In the event that this ACAP server failed, the Email client can issue the same service request again to find the backup ACAP server, say 'service:acap://server2.nosuch.org'. In both cases, the service: URL conforms to the ACAP service template as do the associated attributes (user and group.)

An SNMP based network Manager can use SLP to obtain service:network-management:SNMP URLs. This allows the network Manager to proceed to manage the hosts identified by these URLs without having to scan networks one address at a time, etc. SLP provides the capability to send multicast or directed broadcasts to obtain this information from every managed host.

References

[1] Protocol and service names, October 1994. ftp://ftp.isi.edu/in-notes/iana/assignments/service-names.

[2] Address family numbers, October 1995. ftp://ftp.isi.edu/in-notes/iana/assignments/address-family-numbers.

[3] Port numbers, July 1997. ftp://ftp.isi.edu/in-notes/iana/assignments/port-numbers.

[4] H. Alvestrand. Tags for the Identification of Languages. RFC 1766, March 1995.

[5] ANSI. Coded Character Set -- 7-bit American Standard code for Information Interchange. X3.4-1986, 1986.

[6] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Locators (URL): Generic Syntax and Semantics. RFC1738 as amended by RFC1808

[7] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL). RFC 1738, December 1994.

[8] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC 2045, November 1996.

[9] D. Crocker and P Overell. Augmented BNF for Syntax Specifications: ABNF. draft-ietf-drums-abnf-04.txt, September 1997. (work in progress).

[10] A. Gulbrandsen and P. Vixie. A DNS RR for specifying the location of services (DNS SRV). RFC 2052, October 1996.

[11] R. Moats and M. Hamilton. Finding Stuff (Providing information to support service discovery). draft-ietf-svrloc-advertise-00.txt, February 1997. (work in progress).

[12] J. Myers. Simple Authentication and Security Layer (SASL). RFC 2222, October 1997.

[13] J. G. Myers. ACAP -- Application Configuration Access Prototol. draft-ietf-acap-spec-04.txt, June 1997. (work in progress).

[14] Pete St. Pierre. Definition of lpr: URLs for use with Service Location. draft-ietf-svrloc-lpr-scheme-01.txt, November 1997. (work in progress).

[15] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service Location Protocol. RFC 2165, July 1997.

[16] F. Yergeau. UTF-8, a transformation format of unicode and ISO 10646. RFC 2044, October 1996.

Authors' Addresses

Questions about this memo can be directed to:

Erik Guttman
Sun Microsystems
Bahnstr. 2
74915 **Waibstadt**
Germany
+49 7263 911484
erik.guttman@sun.com

Charles E. Perkins
Sun Microsystems
901 San Antonio Rd.
Palo Alto, CA, 94303
USA
1 650 786 6464
1 650 786 6445 (fax)
charles.perkins@sun.com

James Kempf
Sun Microsystems
901 San Antonio Rd.
Palo Alto, CA, 94303
USA
1 650 786 5890
1 650 786 6445 (fax)
james.kempf@sun.com