Service Location Working Group
INTERNET DRAFT

5 February 1999

Erik Guttman
Charles Perkins
James Kempf
Sun Microsystems

# Service Templates and service: Schemes

draft-ietf-svrloc-service-scheme-14.txt

## Status of This Memo

This document is a submission by the Service Location Working Group of the Internet Engineering Task Force (IETF). Comments should be submitted to the srvloc@srvloc.org mailing list.

Distribution of this memo is unlimited.

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet- Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

## 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 is accompanied by a set of well-defined attributes which define the service. These attributes 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 [10] 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) [8] for syntax specifications.

## 1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [6].

The following terminology is used 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.

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 [2] by convention 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 2 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 version 2 [10] 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.

### 1.2.1. Compatibility with SLPv1

This document adopts the encoding conventions of SLPv2.

Compatibility with SLPv1 [[15]] is possible, if the following conventions are observed:

1. Abstract service types must not be used. SLPv2 specifies the use of Service URLs with abstract service types. SLPv1 does not support them. Thus, a service template which is to serve both SLPv1 and SLPv2 must not use abstract service types.

2. The syntax for representing opaque values in this document is consistent with SLPv2. The syntax must be converted for use with SLPv1. Instead of a sequence of "\FF" then "\" HEXDIG HEXDIG for each byte in the opaque value, SLPv1 uses radix-64 notation.

3. Escape characters are significantly differently in SLPv1 and SLPv2. Instead of using escaped byte notation for escaped characters, SLPv1 uses the HTML convention `&' `#' 1*DIGIT `;'.

## 2. Service URL Syntax and Semantics

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

### 2.1. Service URL Syntax

The syntax of the service: URL MUST conform to an 'absolute URI' as defined by [5]. The syntax of a service: URL differs enough from a 'generic URI' that it is best to treat it as an opaque URI unless a specific parser for the service: URL is available.

All service: URLs have the same syntax up to the 'url-part' The syntax for a service URL depends on the service type following the service scheme. All service: URLs have a service access point portion, indicating the address of the service to access.

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 [5].

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            =   ipsite / atsite / ipxsite
ipsite          =   "//" [ [ 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
ipv4-number     =   1*3DIGIT 3("." 1*3DIGIT)
port            =   1*DIGIT
                    ; A port number must be included if the
                    ; protocol field does not have an IANA
                    ; assigned port number.
user            =   *[ uchar / ";" / "+" / "&" / "=" ]
ipxsite         =   "/ipx/" ipx-net ":" ipx-node ":" ipx-socket
ipx-net         =   8 HEXDIGIT
ipx-node        =   12 HEXDIGIT
ipx-socket      =   4 HEXDIGIT
atsite          =   "/at/" at-object ":" at-type "" at-zone
at-object       =   1*31apple-char
at-type         =   1*31apple-char
at-zone         =   1*31apple-char
apple-char      =   ALPHA / DIGIT / safe / escaped
                =   ; AppleAscii [7] values that are not
                =   ; from the restricted range must be escaped.
                =   ; NOTE: The escaped values do NOT correspond
                =   ; to UTF-8 values here:  They are AppleAscii
                =   ; bytes.
url-part        =   [ url-path ] [ attr-list ]
url-path        =   1 * ( "/" *xchar )
                    ; Each service type must define its
                    ; own syntax consistent
                    ; with [5].
attr-list       =   1 * ( ";" attr-asgn )
attr-asgn       =   attr-id / attr-id "=" attr-value
safe            =   "$" / "-" / "_" / "." / "~"
extra           =   "!" / "*" / "'" / "(" / ")" / "," / "+"
uchar           =   unreserved / escaped
xchar           =   unreserved / reserved / escaped
escaped         =   1*(`\' HEXDIG HEXDIG)
reserved        =   ";" / "/" / "?" / ":" / "@" / "&" / "=" / "+"
unreserved      =   ALPHA / DIGIT / safe / extra
```

IPX addresses [14] are composed of a network, node and socket number. The IPX network number is a four-byte number, in network order and expressed in hexadecimal, that signifies an IPX subnet. The node element represents a network interface card. It is a six-byte number, expressed in hexadecimal, that is usually the same as the

node ID of the interface card. The socket element represents a

specific service access point, given an IPX network and node. IPX sockets are analogous to TCP/IP ports, and are not to be confused with Berkeley sockets.

AppleTalk addresses [9] are composed of an object, type and zone. The object is a human readable string. The type is an identifier, not intentended for human readability. The zone refers to the AppleTalk Zone name, which is also human readable. The characters composing these names are drawn from the AppleAscii character set [7]. Thus, they do not equate to escaped ASCII or UTF-8 characters. The characters "=" and "*" are reserved and may not be included in the names even in binary form.

In cases besides the AppleTalk grammar, some characters must be escaped before use. To escape any character, precede the two digits indicating its ASCII value by '%'.

## 2.2. 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 ":" site url-path
```

where <site> has one of the following forms could refer to an <ipsite>, <ipxsite> or <atsite> if the service URL locates to an IP, IPX or AppleTalk service access point, respectively.

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

The <ipsite> 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 <site> is not sufficient for that purpose. For IP addresses, the <site> field begins with "//". Other address families supported are IPX [14] and AppleTalk [9].

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 [11] 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 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 is registered by network services and obtained by client software.

## 2.3. 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 3. 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 [10] (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.

## 2.4. 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.

## 2.5. 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 a 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.

Concrete service templates inherit all attributes defined in the abstract service template from which the concrete service template was derived. Attribute defined in the abstract service template MUST not be defined in the concrete service template as well. This simplifies interpretation of templates.

For example, if an abstract service template for service type 'Abstract' defines an attribute FOO, all concrete service templates derived from the abstract service template, such as 'Abstract:Concrete' will implicitly include the definition of attribute FOO. This derived template MUST NOT redefine FOO, according to the rule above.

A further example is described in Section A.

### 2.5.1. 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.

## 3. 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.

### 3.1. 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 the 0x00-0x7F subrange of UTF-8 [16] (which corresponds to ASCII character encoding) 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 "template-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 [5].

The service template is encoded in a UTF-8 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 `\' HEXDIG HEXDIG 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 service template submission. Hopefully some public domain tools will emerge for translating escaped UTF-8 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 [8] grammar:

```
template      =  tem-attrs attr-defs
tem-attrs     =  schemetype schemevers schemetext schemeurl
schemetype    =  "template-type=" scheme term
schemevers    =  "template-version=" version-no term
schemetext    =  "template-description=" newline desc term
schemeurl     =  "template-url-syntax=" newline url-bnf term
url-bnf       =  *[ com-chars ]
                 ; An ABNF describing the <url-path> production
                 ; in the service: URL grammar of Section 2.1.
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
langtag       =  1*8ALPHA ["-" 1*8ALPHA]
                 ; See [3]
desc          =  *[ com-chars ]
                 ; A block of free-form text for reading by
                 ; people describing the service in a short,
                 ; informative manner.
term          =  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 / ":" / "_" / "$" / "~" / "@" / "." /
                 "|" / "<" / ">" / "*" / "&"
value         =  string / integer / boolean / opaque
string        =  safe-char *[safe-char / white-sp] safe-char
integer       =  [ "+" | "-" ] 1*DIGIT
boolean       =  "true" / "false"
opaque        =  "\FF" 1*( "\" HEXDIG HEXDIG)
                 ; Each byte of opaque value is hex encoded.
                 ; The format corresponds to [10].
                 ; Newlines are ignored in decoding opaque
                 ; values.
com-chars     =  safe-char / white-sp / "," / ";"/ "%"
safe-char     =  attrchar / escaped / " " / "!" / '"' / "'" /
                 "|" / "(" / ")" / "+" / "-" / "." / ":" /
                 "=" / "?" / "[" / "]" / "{" / "/" / "{" /
                 "$"
                 ; All UTF-8 printable characters are
                 ; included except ",", "%", ";", and "#".
escaped       =  1*(`\' HEXDIG HEXDIG)
white-sp      =  SPACE / HT
newline       =  CR / ( CR LF )
```

### 3.2. 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.

Note that the 'template-type', 'template-version', 'template-description' and 'template-url-syntax' have all been defined as attributes. These attributes MAY accompany any service registration using SLPv2.

#### 3.2.1. Definition of a Service Template

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

- template-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 3.2.2 for the conventions governing service type names.

- template-version

  The version number of the service type specification.

- template-description

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

- template-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.

### 3.2.2. 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 "template-type" item in the template document. If the naming authority name is absent it is assumed to be IANA.

### 3.2.3. Version Number

The version number of the service type template is the value of the "template-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 4 for more detail on how to use the template-version attribute.

### 3.2.4. Description

The description is a block of text readable by people in the language of the template and is the value of the "template-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.

### 3.2.5. 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 "template-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> may be defined so that it must include the queue name, 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 [8] following the rules for URL syntax described in [5]. 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 "template-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.

### 3.2.6. 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 [10] 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 6.

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.

#### 3.2.6.1. 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 [5]. 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.

#### 3.2.6.2. 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 byte escaped values 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.

#### 3.2.6.3. 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 6 for more information about translation.

- x or X

  Indicates that clients SHOULD include the indicated attribute in requests for services. Neglecting to include this attribute will not sufficiently differentiate the service. If services are obtained without selecting this attribute they will quite likely be useless to the client.

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.

#### 3.2.6.4. 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 2.1.

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.

#### 3.2.6.5. 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.

#### 3.2.6.6. 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.

#### 3.2.6.7. Conclusion of An Attribute Definition

An attribute definition concludes with a single empty line.

## 4. 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 "template-version" template attribute set to 0.0.

MAJOR revision numbers come before the '.', MINOR revision numbers come after the '.'.

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:

A MINOR revision number signifies the introduction of a compatible change. A MAJOR revision number signifies the introduction of an incompatible change. This is formalized by the following rules:

- 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 is submitted to a special mailing list, see section 5. This document 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 submission.

The list will be available at svrloc-list@iana.org. 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. Templates with version numbers below 1.0 are not submitted to the IANA. From that point onwards, templates are submitted. See Section 5 for details on how templates are submitted to an IANA registry of templates.

## 5. IANA Considerations

It is the responsibility of the IESG (e.g., Applications Area director) to appoint a Designated Expert (see [12].) Anyone may ask for clarification of a service template. This is to solicit input from the concerned community. It is up to the appointed reviewer to determine whether clarification requests are satisfied. It is the reviewer's responsibility to see that all reasonable clarification requests are met before the template is submitted for inclusion in the IANA registry.

When the reviewer has determined that the template submission is ready, he or she 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.

If a dispute arises over the decisions made by the reviewer, the matter may be appealed according to normal IETF procedure as described for the Standards Track process.

The IANA will maintain a mail forwarding alias for the work of this list, so that ``svrloc-list@iana.org'' points to a mail server supplied by a volunteer organization.

The service template submission MUST be of the form:

```
Name of submitter:
Language of service template:
Security Considerations:
Template Text:
----------------------template begins here----------------------
                              . . .
----------------------template ends here----------------------
```

The service template file has a naming convention:

<service-type> "." <version-no> "." <langtag>

Each of these fields are defined in Section 2. They correspond to the values of the template fields "type", "template-version". The files for the example templates in this document (see Section A) are called:

"foo.0.0.en",
"Net-Transducer.0.0.en",
"Net-Transducer:Thermometer.0.0.de" and
"Net-Transducer:Thermomoter.0.0.en".

The reviewer will ensure that the template submission to IANA has the correct form and required fields.

No service type template will be accepted for inclusion in the service template registry unless the submission 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 to include a service template in the registry by email from the reviewer. This message will include the service template itself, which is to be registered.

Neither the reviewer nor the IANA will take any position on claims of copyright or trademark issues with regard to templates.

## 6. Internationalization Considerations

The service: URL must be encoded using the rules set forth in [5]. The character set encoding is limited to specific ranges within the US-ASCII character set [4].

The template is encoded in UTF-8 characters.

### 6.1. Language Identification and Translation

The language used in attribute strings should be identified supplying a Language Tag [3] in the Service Template submission (see Section 5).

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.

An example of a translated service template is included in Section A.

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.

## 7. 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 [10] 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. They are completely fictitious (ie. the examples do not represent real services).

The FOO example shows how to use service templates for an application that has very few attributes. Clients request the FOO server where

their user data is located by including their user name as the value of the user attribute.

The Net-Transducer example shows how abstract service types are defined and how a corresponding concrete instance is defined. A system might support any of several NetTransducer 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. FOO

The FOO service template submission example follows:

```
Name of submitter: "Erik Guttman" <Erik.Guttman@sun.com>
Language of service template: en
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.
Template Text:
-------------------------template begins here-----------------------
template-type=FOO

template-version=0.0

template-description=
  The FOO service URL provides the location of an FOO service.

template-url-syntax=
  url-path= ; There is no URL path defined for a FOO URL.

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

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

This template could be internationalized by registering another version, say in German:

Name of submitter: "Erik Guttman" <Erik.Guttman@sun.com>
Language of service template: de
Security Considerations:
  Wenn die USER und GROUPS Eigenschaften inbegriffen sind,
  besteht die Moeglichkeit, dass die Liste der Identitaeten
  von Benutzern oder Gruppen endeckt verden kann.  Diese
  Information wurde unter anderen Umstaenden schwierig zu
  entdecken sein.

Template Text:
----------template begins here----------
template-type=FOO

template-version=0.0

template-description=
  Der FOO Service URL zeigt die Stelle von einem Foo Service an.

template-url-syntax=
  url-path= ; Es gibt keinen fuer den FOO URL definierten Pfad.

users= string M L O
# Die Liste aller Users, die der FOO Server unterstuetzt.

groups= string M L O
# Die Liste aller Gruppen, die der FOO Server unterstuetzt.
----------template ends here----------

 Note that the attribute tags are not translated.  If translations
 are desired, the suggested convention for doing so is to define a
 separate attribute called localize-<tag> for each attribute tag which
 is to be localized.  This will aid in displaying the attribute tags
 in a human interface.

 For example, in this case above, the following two attributes could
 be defined:

localize-users= string
Benutzer

localize-groups= string
Gruppen

 The attributes (in SLPv2 attribute list format) for a service
 registration of a FOO service based on this template, in German,
 could be:

(users=Hans,Fritz),(groups=Verwaltung,Finanzbuchhaltung),
(template-type=FOO),(template-version=0.0),(template-description=

Der FOO Service URL zeigt die Stelle von einem Foo Service an.),
(template-url-syntax= \OD url-path= ; Es gibt kein fuer den FOO
URL definiert Pfad. \OD),(localize-users=Benutzer),
(localize-groups=Gruppen)

Anyone obtaining these attributes could display "Benutzer=Hans,Fritz" in a human interface using the included information. Note that the template attributes have been included in this registration. This is OPTIONAL, but makes it possible to discover which template was used to register the service.

## A.2. Abstract Service Type: Net-Transducer

An example submission of an abstract service type template is:

```
Name of submitter: "Erik Guttman" <Erik.Guttman@sun.com>
Language of service template: en
Security Considerations:
  See the security considerations of the concrete service types.
Template Text:
-------------------------template begins here----------------------
template-type=Net-Transducer

template-version=0.0

template-description=
  This is an abstract service type.  The purpose of the Net-
  Transducer service type is to organize into a single category
  all network enabled Transducers which have certain properties.

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

sample-units= string L
# The units of sample that the Transducer provides, for instance
# C (degrees Celsius), V (Volts), kg (Kilograms), etc.

sample-resolution= string L
# The resolution of the Transducer.  For instance, 10^-3 means
# that the Transducer has resolution to 0.001 unit.

sample-rate= integer L
# The speed at which samples are obtained per second.  For
# instance 1000 means that one sample is obtained every millisecond.

-------------------------template ends here-----------------------
```

## A.3. Concrete Service Type: Net-Transducer:Thermometer

This is another service template submission example, supplying a concrete service type corresponding to the abstract template above.

```
Name of submitter: "Erik Guttman" <Erik.Guttman@sun.com>
Language of service template: en
Security Considerations:
  There is no authentication of the Transducer output.  Thus,
  the Thermometer output could easily be spoofed.
Template Text:
-------------------------template begins here----------------------
template-type=service:Net-Transducer:Thermometer

template-version=0.0

template-description=
  The Thermometer is a Net-Transducer capable of reading temperature.
  The data is read by opening a TCP connection to one of the ports
  in the service URL and reading an ASCII string until an NULL
  character is encountered.  The client may continue reading data at
  no faster than the sample-rate, or close the connection.

template-url-syntax=
  url-path     = "ports=" ports-list
  port-list    = port / port "," ports
  port         = 1*DIGIT
                 ; See the Service URL <port> production rule.
                 ; These are the ports connections can be made on.

location-description=string
# The location where the Thermometer is located.

operator=string O
# The operator to contact to have the Thermometer serviced.

-------------------------template ends here-----------------------
```

## A.4. service: URLs and SLP

A user with an FOO enabled calendar application should not be bothered with knowing the address of their FOO server. The calendar client program can use SLP to obtain the FOO service: URL automatically, say 'service:foo://server1.nosuch.org', by issuing a Service Request. In the event that this FOO server failed, the Calendar client can issue the same service request again to find the backup FOO server, say 'service:foo://server2.nosuch.org'. In both

cases, the service: URL conforms to the FOO service template as do the associated attributes (user and group.)

A network thermometer based on the above template could be advertised with the SLPv2 attribute list:

```
URL        = service:net-transducer:thermometer://v33.test/ports=3211
Attributes = (location-description=Missile bay 32),
 (operator=Joe Agent), (sample-units=C),
 (sample-resolution=10^-1),(sample-rate=10),
 (template-type=service:net-transducer:thermometer),
 (template-version=0.0),(template-description=
  The Thermometer is a Net-Transducer capable of reading temperature.
  The data is read by opening a TCP connection to one of the ports
  in the service URL and reading an ASCII string until an NULL
  character is encountered.  The client may continue reading data at
  no faster than the sample-rate, or close the connection.),
 (template-url-syntax= \0D "ports=" port-list \OD
  port-list = port / port "," ports \OD
  port = 1*DIGIT \OD
  ; See the Service URL <port> production rule. \OD
  ; These are the ports connections can be made on.\OD)
```

This might be very useful for a technician who wanted to find a Thermometers in Missile bay 32, for example.

Note that the template attributes are advertised. The template-url-syntax value requires explicit escaped CR characters so that the ABNF syntax is correct.

## B. Full Copyright Statement



## C. Acknowledgments

Thanks to Michael Day and Leland Wallace for assisting with the IPX and AppleTalk address syntax portions. Ryan Moats provided valuable feedback throughout the writing of this document.

References

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

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

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

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

[5] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax. RFC 2396, August 1998.

[6] S. Bradner. Key Words for Use in RFCs to Indicate Requirement Levels. RFC 2119, March 1997.

[7] Apple Computer. Inside Macintosh. Addison-Wesley, 1993.

[8] D. Crocker and P. Overell. Augmented BNF for Syntax Specifications: ABNF. RFC 2234, November 1997.

[9] S. Gursharan, R. Andrews, and A. Oppenheimer. Inside AppleTalk. Addison-Wesley, 1990.

[10] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol version 2. draft-ietf-svrloc-protocol-v2-04.txt, March 1998. (work in progress).

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

[12] T. Narten and H. Alvestrand. RFC 2434: Guidelines for writing an IANA considerations section in RFCs, October 1998. Status: BEST CURRENT PRACTICE.

[13] C. Newman and J. G. Myers. ACAP -- Application Configuration Access Prototol. RFC 2244, November 1997.

[14] Inc Novell. IPX RIP and SAP Router Specification. Part Number 107-000029-001, Version 1.30, May 1996.

[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 ISO 10646. RFC 2279, January 1998.

## Authors' Addresses

Questions about this memo can be directed to:


| Erik Guttman | Charles E. Perkins | James Kempf |
|---|---|---|
| Sun Microsystems | Sun Microsystems | Sun Microsystems |
| Bahnstr. 2 | 15 Network Circle | 15 Network Circle |
| 74915 Waibstadt | Menlo Park, CA 94303 | Menlo Park, CA 94303 |
| Germany | USA | USA |
| +49 7263 911484 | +1 650 786 6464 | +1 650 786 5890 |
| +1 650 786 5992 | +1 650 786 6445 (fax) | +1 650 786 6445 (fax) |
| erik.guttman@sun.com | cperkins@sun.com | james.kempf@sun.com |