ontoprise

know how to use Know-how

SMW$^+$ User Manual

SMW+

Semantic Enterprise Wiki

for Semantic MediaWiki

Version 1.4.3

Authors

Denny Vrandecic, Dominika Wloka, Markus Krötzsch, Yaron Koren, et al.

Publisher

ontoprise GmbH

An der RaumFabrik 29
76227 Karlsruhe

http://www.ontoprise.de

References

The content of this document uses material from the article “SMW User Manual” (http://semantic-mediawiki.org/wiki/Help:User_manual) published by Markus Krötzsch published under the “CC-By (Attribution 3.0 Unported)” licence.

License notice

History

Title	Year	Authors	Publisher	Network location
Help:User manual	2009	Denny Vrandecic, Markus Krötzsch, Yaron Koren, et al.	Markus Krötzsch	http://semantic-mediawiki.org/wiki/Help:User_manual
SMW+ User Manual for Semantic MediaWiki Version 1.4.3	2009	Dominika Wloka	Ontoprise GmbH	http://smwforum.ontoprise.com/smwforum/index.php/Help:Semantic_MediaWiki_1.4.3

Table of Content

1 Introduction to Semantic MediaWiki

Semantic MediaWiki (SMW) is an extension of MediaWiki – the wiki application best known for powering Wikipedia – that helps to search, organise, tag, browse, evaluate, and share the wiki's content. While traditional wikis contain only text which computers can neither understand nor evaluate, SMW adds semantic annotations that allow a wiki to function as a collaborative database. Semantic MediaWiki was first released in 2005, and currently has over ten developers, and is in use on hundreds of sites. In addition, a large number of SMW extensions have been created that extend the ability to edit, display and browse through the data stored by SMW: the term "Semantic MediaWiki" is sometimes used to refer to this entire family of extensions.

2 Administration of Semantic MediaWiki

The following pages describe how to download, install, and configure Semantic MediaWiki (SMW) as a site administrator. If you want to learn about using SMW as a wiki user (with or without administrator rights in the wiki), then the user manual is the right place for you. But basic installation and maintenance of SMW does not necessarily require to be familiar with its usage.

Since SMW is an extension to MediaWiki, a current release of MediaWiki is needed first. This can be obtained in various ways as described on the MediaWiki homepage. Please install this software and make sure that it works as expected before installing SMW. As an extension to MediaWiki, SMW usually requires only very little modifications of the basic system to run. Adjustments might be useful for customising its behaviour, or for tweaking its performance.

2.1 Downloading Semantic MediaWiki

Semantic MediaWiki can be obtained from various sources. Complete file releases are distributed via the SMW project page on SourceForge, while the current development version can be accessed through MediaWiki's Subversion (SVN) system. See Help:Installation for details on installing SMW.

2.1.1 File releases

Semantic MediaWiki is released more or less regularly as a tarball on SourceForge. Those stable releases are recommended for production use. The current stable release of Semantic MediaWiki can be obtained from the project's download page.

2.1.2 SVN access

SMW releases and current development files can be accessed online via *Subversion*. The SMW subversion directory shows the current up-to-date content of the **development repository**. To check out a working copy of the current development files, you can use the following command:

```
svn checkout
http://svn.wikimedia.org/svnroot/mediawiki/trunk/extensions/SemanticMediaWi
ki/
```

Typically this is used directly from within the extension folder of your MediaWiki installation. Of course, a Subversion client needs to be installed for this to work.

Stable releases of SMW can be found in the Subversion directory http://svn.wikimedia.org/svnroot/mediawiki/tags/extensions/SemanticMediaWiki. This source can be used to install stable versions using SVN instead of downloading archives as described above.

2.2 Installation of Semantic MediaWiki 1.4.3

This page describes how to install **Help:Semantic MediaWiki 1.4.3** after having downloaded it (see the bottom of this page for other versions). Installation and upgrade notes change from version to version: the file INSTALL shipped with SMW contains instructions for the particular version you may have downloaded.

To upgrade a wiki running an older version of SMW, please make sure to have read the relevant section *before* starting the upgrade.

For installing SMW, please check the exact requirements and follow the instructions carefully. The installation does no irreversible changes to your MediaWiki database (it just adds some tables that can be deleted again). However, there is currently no automated process to remove annotations from articles texts in the case that Semantic MediaWiki is to be uninstalled again, but one could achieve this with some script that eliminates annotations on editing.

2.2.1 Disclaimer

Make sure you understand the legal disclaimer in the file COPYING.

2.2.2 Requirements

- MediaWiki 1.11.* or greater (tested from MW 1.11 to 1.16alpha (r54352)).
- PHP 5.x or greater installed and working
- MySQL >= 4.0.14 (version required by MediaWiki) or OR PostgreSQL >= 8.3 (note that support for the latter is still preliminary and requires some extra steps, see PostgreSQL for details)

Notes:

- SMW uses the PHP mb_*() multibyte functions such as mb_strpos in the php_mbstring.dll extension. This is standard but not enabled by default on some distributions of PHP. See the PHP manual for details.
- For installation and upgrade, SMW needs the rights to create new tables (CREATE) and to alter tables (ALTER TABLE). Both can be removed again after SMW was set up. The script SMW_setup.php can use the DB credentials from AdminSettings.php for this purpose, avoiding the need of extra rights for the wiki DB user.
- SMW creates and alters temporary tables for certain semantic queries. To do this, your wikidb user must have privileges for CREATE TEMPORARY TABLES. If this is not desired, the SMW features which require this right can be disabled by adding the following to Localsettings.php:

```
 $smwgQSubcategoryDepth=0;

 $smwgQPropertyDepth=0;

 $smwgQFeatures         = SMW_ANY_QUERY & ~SMW_DISJUNCTION_QUERY;

 $smwgQConceptFeatures = SMW_ANY_QUERY & ~SMW_DISJUNCTION_QUERY &
~SMW_CONCEPT_QUERY;
```

2.2.3 Installation

If you upgrade an existing installation of Semantic MediaWiki, also read the remarks in the section Upgrading existing installations below! Installing SMW basically requires three (or four) easy steps:

(1) Copy files. Extract the downloaded archive, or check out the current files from SVN to obtain the directory "SemanticMediaWiki" that contains all relevant files. Copy this directory to "[wikipath]/extensions/" (or extract/download it to this place). We abbreviate "[wikipath]/extensions/SemanticMediaWiki" as "[SMW_path]" below.

(2) Enable the extension. Insert the following two lines into the file "[wikipath]/LocalSettings.php":

```
include_once("$IP/extensions/SemanticMediaWiki/includes/SMW_Settings.php");
enableSemantics('example.org');
```

where example.org should be replaced by your server's name (or IP address). This string is only used as a globally unique name for identifying the wiki's exported data on the Semantic Web, and a valid server name works very well for that purpose. There is no need to worry if a wiki has more than one server name – just pick one.

(3) *[optional]* Adjust namespaces. If didn't make any change the your wiki's namespaces, you can skip this step. If you have defined your own custom namespaces, you have to set the parameter $smwgNamespaceIndex before including SMW_Settings.php. Semantic MediaWiki uses ten additional namespace indexes, in the range from 100 to 109. Note, 100 and 101 are only needed if $smwgSMWBetaCompatible is set. 106 and

107 are reserved for the SemanticForms extension and not used by SMW. See the documentation within SMW_Settings.php for details. If you add namespaces after installing SMW, then you have to assign them to higher numbers than those used by Semantic MediaWiki.

Note: Semantic MediaWiki only evaluates semantic annotations in some namespaces. For example, by default it ignores semantic annotations in talk pages. If you want to change the namespaces with annotations (likely if you have added your own custom namespaces), then you have to change the array variable $smwgNamespacesWithSemanticLinks in SMW_Settings.php.

(4) Setup database. In your wiki, log in as a user with admin status and go to the page "Special:SMWAdmin" to do the final setup steps. Two steps are needed: at first, trigger the database setup ("Database installation and upgrade"). Afterwards, activate the automatic data update ("Data repair and upgrade"). Note that the first step requires permissions to alter/create database tables, as explained in the above note. The second step takes some time; go to Special:SMWAdmin to follow its progress.

2.2.3.1 Testing your Installation

If you are uncertain that everything went well, you can do some testing steps to check if SMW is set up properly.

Go to the Special:Version page. You should see Semantic MediaWiki (version nn) listed as a Parser Hook there.

Create a regular wiki page named "TestSMW", and in it enter the wiki text

```
Property test:  [[testproperty::Dummypage]]
```

When previewing the page before saving, you should see a Factbox at the bottom of the article that shows your input. After saving the page, click on the link "Browse properties" in the page's toolbox. This view should show Testproperty with value Dummypage.

2.2.3.2 Running SMW on older versions of MediaWiki

In general, it is not recommended to run older versions of MediaWiki, since every new release brings also security fixes. If your site is still running on PHP4, SMW is not supported. It would hardly be possible to backport the code to the old PHP version. If you have PHP5 but an older MediaWiki version, additional patches/modifications might be needed. Download an older release of SMW and have a look at the included INSTALL instructions to find out whether other changes are recommended therein.

2.2.4 Upgrading existing installations

Existing SMW installations that already use the default store (SMWSQLStore2) can simply follow the above installation steps. This covers all sites with SMW 1.4.1 or greater, and all sites with SMW 1.2 to SMW 1.3 that do not use a special setting to switch back to the old store ("SMWSQLStore" or "SMWRAPStore"). In the latter cases the new data upgrade feature will ensure that your data is refreshed to the new version. To speed up the update, use MediaWiki's script runJobs.php. When upgrading from SMW versions 1.4, it can also be useful to do a manual data update to ensure that the values for new special properties are stored correctly, but this is not strictly required for normal operation.

Older SMW versions or SMW versions that use the old store can upgrade in the same fashion, but first need to remove any setting that configures SMW to use the old store. Again, all data will immediately be available when the data upgrade has finished.

Directly after enabling SMW 1.4.3, it can potentially happen that the site shows SQL errors due to background jobs trying to access the new DB structure even before you initialised it. This can even block you from reaching SMWAdmin. In this case, either use the script SMW_setup.php from the command-line, or access your database and delete all entries in the table "job". After this, no errors should show up while accessing the special page for initialisation.

Especially when upgrading from SMW 1.3 or older, extensions of SMW (e.g. Semantic Forms) need to be updated to a version that is compatible with SMW 1.4.3. See your extension's homepage for details. If used, it is suggested to update to Semantic Result Formats (SRF) in version 1.4.6 or greater. The formats have been migrated to this extension package and are enabled there by default after installation.

2.2.4.1 Changed configuration options

Some configuration options (used in LocalSettings.php) have changed since SMW 1.0. All settings are documented in detail in the file SMW_Settings.php. The following list gives the version in which a setting was first introduced, so it can be ignored if you already run this version or a more recent one.

- **[1.3]** Since SMW 1.3, the Factbox is hidden by default and will only appear. As a compensation, the toolbox (usually on the left below the search field) shows a link to Special:Browse. To use the display behaviour of the Factbox as it was before SMW 1.3, set $smwgShowFactbox = SMW_FACTBOX_NONEMPTY; The new link and the Factbox during editing can also be configured; see SMW_Settings.php for details.
- **[1.2.1]** SMW will no longer support nested link syntax in property values by default, since this was known to cause problems. To re-enable this, set $smwgLinksInValues = true;
- **[1.2]** If your wiki uses <ask> syntax or the Relation: namespace, you may want to set $smwgSMWBetaCompatible = true; *before* the line including SMW_Settings.php in LocalSettings.php. Consider changing <ask> to #ask to be able to drop SMW beta compatibility.
- **[1.2]** If you had a setting $smwgQDisjunctionSupport = false, then you now need the following settings:

```
$smwgQFeatures          = SMW_ANY_QUERY & ~SMW_DISJUNCTION_QUERY;
```

```
 $smwgQConceptFeatures = SMW_ANY_QUERY & ~SMW_DISJUNCTION_QUERY &
~SMW_CONCEPT_QUERY;
```

- **[1.0]** $smwgQEqualitySupport is one of SMW_EQ_NONE, SMW_EQ_SOME, SMW_EQ_FULL (it was true or false until SMW 1.0 RC1-3).
- **[1.0]** $smwgQDefaultNamespaces is now NULL by default, so that all namespaces are queried. The default in SMW RC 1-3 was "array(NS_MAIN, NS_IMAGE)".
- **[1.0]** $smwgQDefaultLinking now defaults to 'all' such that all query results are linked. This is not a performance issue any more. The default until SMW RC 1-3 was 'subject'.

If you currently set any of these parameters in your LocalSettings.php, you need to update this file. Please see SMW_Settings.php for more documentation on the available parameters.

2.3 Configuring the Concept Caching Feature

In order to speed up **semantic query** answering for queries that use **concepts**, Semantic MediaWiki offers a special mechanism for pre-computing query results. This feature is especially useful for large wikis that still want to make use of complex queries in a controlled way. In particular, there are various options to specify which queries should be computed "life" and which queries should be answered only if a cache is available.

The performance increase gained by caching a concept is comparable to the effect of replacing every use of this concept by a MediaWiki category. It affects all semantic queries that use this concept as well as the display of the concept page as such. On the other hand, pre-computed results for a concept query might become out of date, so the displayed results might no longer agree with the contents of the wiki. It is possible to speciy how old a cached result should at most be until SMW will attempt to recompute its results (see below).

2.3.1 Creating and managing concept caches

Under the default configuration, SMW will use a concept cache whenever it is available and no older than one day. To create a cache for a concept, the maintenance script SMW_conceptCache.php can be used. Like any other SMW script, it is executed by changing into the directory [SMWpath]/maintenance and running php SMW_conceptCache.php. Doing this will display a documentation on how to use this script. If this fails, then you probably use a non-standard directory structure on your wiki – read [SMWpath]/maintenance/README to find out how to run scripts in your case.

The script SMW_conceptCache.php has three basic modes of operation that are selected by according parameters:

- php SMW_conceptCache.php --status shows the status of all concept caches on your site (including the case that there is no cache for a concept). You can use this now to see which concept pages you got.

- php SMW_conceptCache.php --create creates new caches for all concepts, or updates them if they are already there.
- php SMW_conceptCache.php --delete deletes the caches of all concepts.

Each of these actions refers to *all concepts*. There are a number of parameters to restrict the operation to only some concepts:

- --concept "Concept name" Process only the one concept of the given name. The name should not include a namespace prefix, but it needs surrounding " if the name contains spaces.
- --hard Process only concepts that are not allowed to be computed online according to the current wiki settings. See below for further details.
- --old <min></min> Process only concepts with caches older than <min>minutes or with no caches at all.</min>
- --update Process only concepts that already have some cache, i.e. do not create any new caches. For the opposite (only concepts without caches), use --old with a very high number.
- -s <startid></startid> Process only concepts with page id of at least <startid></startid>
- -e <endid></endid> Process only concepts with page id of at most <endid></endid>

These options can be combined to restrict the selection of concepts further. For example, the call

```
php SMW_conceptCache.php --create --update --old 30
```

will create caches for all concepts that already have a cache, but where the cache is older than 30 minutes.

2.3.2 Configuring SMW to use caches

SMW has basically three options for handling concepts in queries and on the conept page:

- Compute the elements of the concepts when needed, using the current wiki data.
- Retrieve the elements of the concepts from a cache.
- Reject the use of the concept completely, treating it like a query with no results.

The default behaviour of SMW is use available caches if those are not older than one day, and to otherwisecompute concept elements on the fly as long as the concept would be allowed as a (non-concept) inline query according to the current wiki settings. It is possible to create concepts that are not allowed as inline queries, since the restrictions on the size and complexity of concepts are less strict than for inline queries. Results for such concepts will by default only be returned from cache (no matter how old it is), and otherwise not be supplied. The following sections explain the relevant parameters to configure that behaviour.

2.3.3 Which queries are allowed as inline?

Three parameters are used to determine how "complex" a query is: *size*, *depth*, and the types of query *features* it uses. The size is essentially the overall number of query conditions found in the query. The depth is the maximal number of chained property statements. For example [[Some property::value]] has depth 1, and [[property1.property2::value]] has depth 2. The query features are the types of query conditions that are used. You can use #ask with format=debug to see the ize and depth of a query.

SMW has configuration options to set a maximal size and depth, and to restrict the available features for queries that are used inline or on special pages. These parameters and their default values are:

- $smwgQMaxSize = 12;
- $smwgQMaxDepth = 4;
- $smwgQFeatures = SMW_PROPERTY_QUERY | SMW_CATEGORY_QUERY | SMW_CONCEPT_QUERY | SMW_NAMESPACE_QUERY | SMW_CONJUNCTION_QUERY | SMW_DISJUNCTION_QUERY;

The allowed features are simply all available features by default, but one could restrict that on large wikis. To allow only category and concepts queries, e.g., one would set:

```
$smwgQFeatures = SMW_CATEGORY_QUERY | SMW_CONCEPT_QUERY;
```

in LocalSettings.php. To allow also the conjunction (intersection) of these, one would use

```
$smwgQFeatures = SMW_CATEGORY_QUERY | SMW_CONCEPT_QUERY |
SMW_CONJUNCTION_QUERY;
```

If a query is not allowed according to these options, then it will be cut-down to a simpler query. Concepts may use queries that do not meet these requirements. If this happens, the concept is simply not computed "life" at all, and results are only shown when a cache has been created for that concept. Concepts that are affected by this can be selected by the option --hard in php SMW_conceptCache.php. In that way, users can create (propose) concepts, and administrators may supply caches for them if feasible.

2.3.4 Which queries are allowed in concepts?

For concepts, SMW supports settings similar to those explained above, but with slightly different defaults:

- $smwgQConceptMaxSize = 20;
- $smwgQConceptMaxDepth = 8;
- $smwgQConceptFeatures = SMW_PROPERTY_QUERY | SMW_CATEGORY_QUERY | SMW_NAMESPACE_QUERY | SMW_CONJUNCTION_QUERY | SMW_DISJUNCTION_QUERY;

As before, these can be changed in LocalSettings.php. The defaults show that some queries will be allowed in concepts while not being allowed in #ask. As discussed before, such concepts will not be enabled until they have some cache.

2.3.5 Hard and simple concepts

The above sections explained that some concepts in SMW are considered "hard". By default, this is the case if they represent queries that would not be allowed as inline queries. This can be changed by setting the value of the option $smwgQConceptCaching. The possible settings are:

- $smwgQConceptCaching = CONCEPT_CACHE_HARD; The default setting as explained above.
- $smwgQConceptCaching = CONCEPT_CACHE_ALL; All concepts are considered to be "hard", i.e. concepts will never be computed online and always rely on caches.
- $smwgQConceptCaching = CONCEPT_CACHE_NONE; No concepts will be considered hard. Concepts may still use caches if available (see next section), but they do not depend on them in any case. This can be useful if the concept namespace is write-restricted to a certain trusted user group who will be the only ones who can create new concept queries.

The default setting in SMW is $smwgQConceptCaching = CONCEPT_CACHE_HARD;

2.3.6 When will available caches be used?

Hard concepts will always try to use a cache, since they simply cannot display anything without a cache. For simple concepts, re-computing the results online is possible, and SMW will use the cache age to decide what to do. The configuration parameter $smwgQConceptCacheLifetime specifies how old (in minutes) a cache is allowed to be. Older caches are not used if the configuration of SMW allows to compute their elements online. The default setting is

- $smwgQConceptCacheLifetime = 24*60;

corresponding to one day.

3 Browsing Interfaces

3.1 The Factbox

The factbox is a box at the bottom of wiki pages which summarises the semantic data that was entered into the page. This also helps editors to check whether Semantic MediaWiki actually «understood» the supplied information as intended. Users can read the Factbox for getting a quick overview, and for using its links to further information. Note that the Factbox might be switched off by the site administrator, as some wikis do not wish to show the added information on each page (see Configuring SMW 1.2).

Factboxes show information in two columns: the left column displays the *property* that some information belongs to (e.g. *population*), while the right column shows the value of that property (e.g. *3,410,000*). Each property name is a link to the property's article in the wiki, where one can normally find more information about a property's meaning and usage.
Annotations in the Factbox usually provide links to involved wiki pages, and for properties that support Units of Measurement, the Factbox also shows converted values in other units.

The icon 🔍 next to each property value links to a simple search at Special:SearchByProperty (see below). For example, if an article contains the annotation Germany then its Factbox links to a search listing all pages with the same annotation (i.e. everything located in Germany). Similarly, the header of the Factbox shows an icon ◐ that links to a simple semantic browser for the given page (see below).

The Factbox may also show an icon ⓘ that when clicked links to external web services that provide more information about a property value. For example, a property for specifying geographic coordinates might link to online mapping services that provide aerial images and maps of the chosen location. Wiki administrators can choose the links that each property displays; see Adding service links to a wiki for details on how to do it, i.e. built-in properties that are relevant to SMW, are displayed in italics and show a tooltip when hovering the mouse over them (requires JavaScript). This emphasises their special meaning and helps editors to spot errors.

Finally, the factbox contains a link to retrieve the Factbox contents in the machine-readable OWL/RDF format as explained in the section on Semantic Web technologies in SMW.

Editors can put the magic words on any page to hide or display (if nonempty) the Factbox on any page, independently of the global wiki configuration.

3.2 Semantic Browsing

Special:Browse offers a simple browsing interface for the computer-readable data of Semantic MediaWiki. Users start by entering the name of a page. The special then displays all semantic properties of that page (similar to the content of the Factbox), and all semantic links that lead to that page. By clicking on the icon ◐, the user can browse to another article. For a more detailed description and some configuration settings, see Help:Browse.

3.3 Simple search interfaces

Semantic MediaWiki provides a number of very simple search forms that allow users to find specific information. These search features are accessed in various special pages:

- Special:SearchByProperty has a simple search form for finding *semantic backlinks*. Users enter a property name and target value. The search returns a list of all pages that have that property with that value. If you search for a property of a numeric type, and there are only few results, nearest results will be shown as well. This can be switched off by setting $smwgSearchByPropertyFuzzy to *false* in your local settings. This service is directly accessible through the 🔍 link within the Factbox or the browse page.

- Special:PageProperty displays all values some page has for some property. Users enter a page and a property name. The search displays a list of all values of the property on that page. If Factboxes are enabled in the wiki, the same information can also be read off the Factbox of the page. Since SMW 1.4.3, it is also possible to retrieve *all* values that a property has been given throughout the wiki, simply by leaving the subject input empty.

3.4 Viewing all properties, types, and values

Each property has an own page in the Property namespace, similar to the category pages in MediaWiki's Category namespace. These property pages show all pages using the property together with the property's value(s) on that page, possibly with service links

Besides the links in the Factbox and the normal MediaWiki search, there are also special ways of finding properties of a wiki:

- Special:Properties lists properties that appear in annotations ordered by the frequency of their usage. Most frequently used properties are displayed on top.
- Special:UnusedProperties lists property pages that are *not* in any annotations. This may indicate that a property was abandonded and should be deleted.
- Special:WantedProperties lists properties that are used but do not have a descriptive page. This is not desirable, since users then cannot find any documentation on such properties, so that confusion may arise regarding their proper use.
- Special:Types lists the available datatypes for properties. See Help:Editing for an introduction into datatypes and their relevance in SMW.

These special pages are particularly useful for wiki gardeners and editors, since they give some overview of how properties are used in the wiki.

These approaches all search for semantic properties of pages. To simply view existing pages about properties, categories, and types, *even if unused or garbled*, use MediaWiki's Special:Allpages to display all pages within these namespaces.

3.5 External tool support

Semantic MediaWiki makes semantic knowledge available to external tools via its OWL/RDF export, and it therefore is possible to write external tools that implement further advanced browsing and searching functionality. Currently, many tools that work on RDF output are not very user-friendly yet, but the supplied data format RDF is easy to process and could be integrated in much more elaborate web or desktop applications. For more information, there is a list of tools that have been tested with Semantic MediaWiki so far. Feel free to add your own tool there as well.

4 Semantic Search

Semantic MediaWiki includes an easy-to-use **query language** which enables users to access the wiki's knowledge. The syntax of this query language is similar to the syntax of annotations in Semantic MediaWiki. This query language can be used on the special page Special:Ask, in concepts, and in inline queries. This page provides a short introduction to semantic search in general. More detailed explanations are found on other pages of this manual:

- Help:Selecting pages: explains the basic way to describe what pages should appear in a query result. This is the core of SMW's query language.
- Help:Displaying information: introduces *printout statements* as a way of showing additional information in queries, such as property values or category assignments.
- Help:Concepts: shows how queries can be saved in *concepts*, which are a kind of «dynamic categories» offerend by SMW.
- Help:Inline queries: explains ways of including query results into wiki pages, and shows how to format the query results for display. This is the purpose of the SMW parser functions #ask and #show.
- Help:Inferencing: explains how one can specify general schematic knowledge in SMW (and what this is in the first place). This feature is used by SMW to smartly deduce facts that were not directly entered into the wiki.

Naturally, answering queries requires additional resources, and the administrators of some sites can decide to switch off or restrict query features in order to ensure that even high-traffic sites can handle the additional load.

4.1 Introduction

Semantic queries specify two things:

1. Which pages to select
2. What information to display about those pages

All queries must state some *conditions* that describe what is asked for. You can select pages by name, namespace, category, and most importantly by property values. For example, the query

```
[[Located in::Germany]]
```

is a query for *all pages with the "Located in" property with a value of "Germany"*. If you enter this in Special:Ask and click "Find results", SMW executes the query and displays results as a simple table of all matching page titles. If there are many results, they can be browsed via the navigation links at the top and bottom of the query results, for example a query for all persons on semanticweb.org.

The second point is important to display more information. In the example above, one might be interested in the population of the things located in Germany. To display that on Special:Ask, one just enters the following into the printout box on the right:

```
?Population
```

and SMW displays the same page titles and the values of the Population property on those pages, if any. Printout statements may have some additional settings to further control how the property is displayed.

1.1 Selecting Pages

The most important part of the Semantic search features in Semantic MediaWiki is a simple format for describing which pages should be displayed as the search result. Queries select wiki pages based on the information that has been specified for them using *Categories*, *Properties*, and maybe some other [MediaWiki features such as a page's namespace. The following paragraphs introduce the main query features in SMW.

4.1.1 Categories and property values

In the introductory example, we gave the single condition [[Located in::Germany]] to describe which pages we were interested in. The markup text is exactly what you would otherwise write to *assert* that some page has this property and value. Putting it in a semantic query makes SMW return all such pages. This is a general scheme: *The syntax for asking for pages that satisfy some condition is exactly the syntax for explicitly asserting that this condition holds.*

The following queries show what this means:

1. [[Category:Actor]] gives all pages directly or indirectly (through a sub-, subsub-, etc. category) in the category.
2. [[born in::Boston]] gives all pages annotated as being about someone born in Boston.
3. [[height::180cm]] gives all pages annotated as being about someone having a height of 180cm.

By using other categories or properties than above, we can already ask for pages which have certain annotations. Next let us combine those requirements:

```
[[Category:Actor]] [[born in::Boston]] [[height::180cm]]
```

asks for everybody who is an actor *and* was born in Boston *and* is 180cm tall. In other words: when many conditions are written into one query, the result is narrowed down to those pages that meet *all* the requirements. Thus we have a logical AND. By the way:

queries can also include line breaks in order to make them more readable. So we could as well write:

```
[[Category:Actor]]

[[born in::Boston]]

[[height::180cm]]
```

to get the same result as above. Note that queries only return the articles that are positively known to satisfy the required properties: if there is no property for the height of some actor, that actor will not be selected.

When specifying property values, SMW will usually ignore any initial and trailing whitespace, so the two conditions [[height::180cm]] and [[height:: 180cm]] mean the same. Datatypes such as number may have additional features such as ignoring commas that might be use to separate the thousands. SMW will also treat synonymous page names the same, just like MediaWiki would usually consider `Semantic wiki`, `Semantic_wiki`, and `semantic wiki` to refer to the same page.

4.1.2 Property values: wildcards and comparators

In the examples above, we gave very concrete property conditions, using «Boston» and «180cm» as values for properties. In many cases, one does not look for only one particular values, but for a whole range of values, such as all actors that are taller than 180cm. In some cases one may even just look for all pages that have any values for a given property at all. For example, the deceased people could be those which have a value for the property «date of death». Such general conditions are possible with the help of *comparators* and *wildcards*.

- **Wildcards** are written as "+" and allow any value for a given condition. For example, [[born in::+]] returns all pages that have any value for the property «born in».

Comparators are special symbols like < or >. They are placed after `::` in property conditions. SMW currently supports the following comparators:

- **>** and **<**: greater than/less than or equal
- **!**: unequal
- **~**: «like» comparison for strings (disabled by default)

Comparators work only for property values, but not for conditions on categories. A wiki installation can limit which comparators are available, which is done by the administrator by modifying the value of `$smwgQComparators` as explained in the file SMW_Settings.php.

4.1.2.1 Greater than or equal, less than or equal

With numeric values, you often want to select pages with property values within a certain range. For example

```
[[Category:Actor]] [[height::>6 ft]] [[height::<7 ft]]
```

asks for all actors that are between 6 feet and and 7 feet tall. Note that this takes advantage of the automatic unit conversion: even if the height of the actor was set with [[height::195cm]] it would be recognized as a correct answer (provided that the datatype for height understands both units, see Help:custom units). Note that the comparator means greater/less than *or equal* – the equality symbol `=` is not needed.

Such range conditions on property values are mostly relevant if values can be ordered in a natural way. For example, it makes sense to ask [[start date::>May 6 2006]] but is is not really helpful to say [[homepage URL::>http://www.somewhere.org]].

If a datatype has no natural linear ordering, Semantic MediaWiki will just apply the alphabetical order to the normalised datavalues as they are used in the RDF export. You can thus use greater than and less than to select alphabetic ranges of a string property. For example, you could ask [[surname::>Do]] [[surname::<G]] to select surnames between «Do» and up to «G». For wiki pages, the comparator refers to the name of the given page (without the namespace prefix).

Here and in all other uses of comparators, it might happen that a searched for value really starts with a symbol like `&lt`. In this case, SMW can be prevented from interpreting the symbol as a comparator if a space is inserted after `::`. For example, [[property:: <br>]] really searches for pages with the value «<br>» for the given property.

4.1.2.2 Not equal

You can select pages that have a property value which is unequal to a given value. For example, [[Area code::!415]] will select pages that have an area code which is not «415». Note that this is query description does not look for pages which do *not* have an area code 415. Rather, it looks for all pages that (also) have a code unequal to 415. In particular, pages that have no area code at all cannot be the result of the above query.

As with the (default) equality comparator, the use of custom units may require rounding in numeric conversions that can lead to unexpected results. For example, [[height::!6.00 ft]] may still select someone whose height displays as «6.00 feet» simply because the exact numeric value is not really 6. In such situations, it might be more useful to query for pages that have a property value outside a certain range, expressed by taking a disjunction (see below) of conditions with < and >.

4.1.2.3 Like

The comparator ~ works only for properties of Type:String and Type:Geographic coordinate. For strings, in a like condition one uses '*' wildcards to match any sequence of characters and '?' to match any single character. For example, one could ask "[[Address::~*Park Place*]]" to select addresses containing the string «Park Place», or "[[Honorific::~M?.]]" to select both «Mr.» and «Ms.». For coordinates, this comparator takes a coordinate and finds all points that are close to it (provided they match the other criteria). The default distance away is 5 miles, or 8.05 kilometers, its equivalent. This distance can be changed by adding the "distance=" parameter, which can take a value in either miles or kilometers. So, to find all pages with a coordinate value within 3 kilometers of the Eiffel Tower, you could add "[[Coordinates::~25.0955°N, 55.342083°E]]|distance=3 km".

4.1.3 Unions of query results: disjunctions

Disjunctions are OR-conditions that admit several alternative conditions on query results. SMW has two ways of writing disjunctions in queries:

- The operator OR is used for taking the union of two queries.
- The operator || is used for disjunctions in values, page, and category names.

In any case, the disjunction requires that at least one (but maybe more than one) of the possible alternatives is satisfied (logical OR). For example, the query

```
[[born in::Boston]] OR [[born in::New York]]
```

describes all pages of people born in Boston *or* New York. This can also be written with || as as [[born in::Boston||New York]]. In the latter case, «Boston||New York» describes a value that may be either of the two alternatives. Writing queries with || is usually more concise, but not all disjunctions can be written in this way. The following is an example that can not be expressed with ||:

```
[[born in::Boston]] OR [[Category:Actor]]
```

The || syntax can be used not only in property values, but also with catgories, like in the query [[Category:Musical actor||Theatre actor]].

4.1.4 Describing single pages

So far, all conditions depended on some or the other annotation given within an page. But there are also conditions to directly select some pages, or pages from a given namespace.

Directly giving some page title (possibly including a namespace prefix), or a list of such page titles separated by `||`, selects the pages with those names. An example is the query

```
[[Brazil||France||User:John Doe]]
```

which has three results (at least if the pages exist). Note that the result does not display any namespace prefixes; see the hover box or status bar of the browser, or follow the links to determine the namespace. Restricting the set based on an attribute value one could ask, e.g., «Who of Bill Murray, Dan Aykroyd, Harold Ramis and Ernie Hudson is taller than 6ft?». But direct selection of articles is most useful if further properties of those articles are asked for, e.g. to simply print the height of Bill Murray.

To select a category in this way, a `:` must be put before the category name. This avoids confusing [[Category:Actor]] (return all actors) and [[:Category:Actor]] (return the category «Actor»).

4.1.5 Restricting results to a namespace

A less strict way of selecting given pages is via namespaces. The default is to return pages in every namespace. To return pages in a particular namespace, specify the namespace with a «wildcard», e.g. write [[Help:+]] to return every page in the «Help» namespace. Since the main namespace usually has no prefix, write [[:+]] to select only pages in the main namespace.

Disjunctions work again with the `||` syntax as above. For example, to return pages in either the main or «User» namespace, write [[:+||User:+]]. To return pages in the «Category» namespace, a `:` is again needed in front of the namespace label to prevent confusion.

4.1.6 Subqueries and property chains

Enumerating multiple pages for a property is cumbersome and hard to maintain. For instance, to select all actors that are born in a Italian city one could write:

```
[[Category:Actor]] [[born in::Rome||Milan||Turin||Florence||...]]
```

To generate a list of all these Italian cities one could run another query

```
[[Category:City]] [[located in::Italy]]
```

and copy and paste the results into the first query. What one would like to do is to use the city query as a *subquery* within the actor query to obtain the desired result directly. Instead of a fixed list of page names for the property's value, a new query enclosed in

<q> and </q> is inserted within the property condition. In this example, one can thus write:

```
[[Category:Actor]] [[born in::<q>[[Category:City]] [[located
in::Italy]]</q>]]
```

Arbitrary levels of nesting are possible, though nesting might be restricted for a particular site to ensure performance. For another example, to select all cities of the European Union you could write:

```
  [[Category:Cities]]

  [[located in::<q>[[member of::European Union]]</q>]]
```

(no results within this wiki)

In the above example, we essentially have cosntructed a *chain* of properties «located in» and «member of» to find things that are located in something which is a member of the EU. Queries can be written in a shorter form for this common case:

```
[[Category:Cities]] [[located in.member of::European Union]]
```

This query has the same meaning as above, but with much less special sybols required. In general, chains of properties are created by listing all properties separated by dots. In the rare case that a property should contain a dot in its name, one may start the query with a space to prevent SMW from interpreting this dot in a special way.

NOTE: It is not possible to use a subquery to obtain a list of properties that is then used in a query. See #Subqueries for properties below.

4.1.7 Using templates and variables

Arbitrary templates and variables can be used in a query. An example is a selection criteria that displays all future events based on the current date:

```
 [[Category:Event]]

 [[end date::>{{CURRENTYEAR}}-{{CURRENTMONTH}}-{{CURRENTDAY}}]]
```

Another particularly useful variable for inline queries is `{{FULLPAGENAME}}` for the current page with namespace, which allows you to reuse a generic query on many pages. For an example of this, see Property:Population. Read about inline queries for more information.

4.1.8 Sorting results

It is often helpful to present query results in a suitable order, for example to present a list of European countries ordered by population. Special:Ask has a simple interface to add a sorting condition to a query. The name of the property to sort by is entered into a text input, and ascending or descending order can be selected. SMW will usually attempt to sort results by the *natural order* that the values of the selected property may have: numbers are sorted numerically, strings are sorted alphabetically, dates are sorted chronologically. The order therefore is the same as in the case of the < and > comparators in queries. If no specific sorting condition is provided, results will be ordered by their page name.

It is possible to provide more than one sorting condition. If multiple results turn out to be equal regarding the first sorting condition, the next condition is used to order them and so on. A query for actors, e.g., could be ordered by year of birth and use the last name of the actor as a second ordering condition. All actors that were born in the same year would thus be ordered alphabetically by their last name instead of appearing in random order.

Sorting a query can also influence the result of a query, because it is only possible to sort by property values that a page actually has. Therefore, if a query is ordered by a property (say «Population») then SMW will usually restrict the query results to those pages that have at least one value for this property (i.e. only pages with specified population appear). Therefore, if the query does not require yet that the property is present in each query result, then SMW will silently add this condition. But SMW will always try to find the ordering property withint the given query first, and it is even possible to order query results by subproperties. Some examples should illustrate this:

- `[[Category:City]] [[Population::+]]` ordered by «Population» will present the cities with population in ascending order. The query result is the same as without the sorting.

- `[[Category:City]]` ordered by «Population» will again present the cities with population in ascending order. The query result may be modified due to the sorting condition: if there are cities without a population given, then these will no longer appear in the result.

- `[[Category:City]] [[has location country.population::+]]` ordered by «Population» will present the cities ordered by the population of the countries they are located in. The query result is not changed, but «population» now refers to a property used in a subquery.

If a property that is used for sorting has more than one value for some page, then this page will still appear only once in the result list. The position that the page takes in this case is not defined by SMW and may correspond to either of the property values. In the above examples, this would occur if one city would have multiple population numbers specified, or if one city is located in multiple countries each of which has a population. It is suggested to avoid such situations.

Query results displayed in a result table can also be ordered dynamically by clicking on the small sort icons found in the table heading of each column. This function requires JavaScript to be enabled in the browser and will sort only the displayed results. So if, e.g., a query has retrieved the twenty world-largest cities by population, it is possible to sort these twenty cities alphabetically or in reverse order of population, but the query will certainly not show the twenty world-smallest cities when reversing the order of the population column. the dynamic sorting of tables attepts to use the same order as used in SMW queries, and in particular orders numbers and dates in a natural way. However, the alphabetical order of strings and page names may slightly vary from the wiki's alphabetic order, simply because there are many international alphabets that can be ordered in different ways depending on the language preference.

4.1.9 Linking to Semantic Search Results

Links to semantic query results on Special:Ask can be created by means of the inline query feature in SMW as explained in its documentation. It is not recommended to create links directly, since they are very lengthy and use a specific encoding. Developers who create extensions that link to Special:Ask should also use SMW's internal functions for building links. Understanding the details of SMW's encoding of queries in links is therefore not required for using SMW.

4.1.10 Things that are not possible

4.1.10.1 Subqueries for properties

It is not possible to use a subquery to obtain a list of properties that is then used in a query. One can, however, use a query that returns a list of properties, and copy and paste the result into another query. Alternatively, one can use the **template** results format to pass properties directly to another query.

4.1.10.2 Queries with special properties

SMW currently does not support queries for the values of any of SMW's built-in Special properties such as «Has type», «Allows value» or «Equivalent URI».

4.2 Displaying Information

Queries in Semantic MediaWiki return a list of pages, and the default result of a query therefore simply displays the selected pages' titles. Additional information such as a page's property values or categories, can be included into a query result by using additional **printout statements** that are introduced here. In Special:Ask, printout statements can simply be entered into the input box on the right, with one statement per line.

There are different kinds of printout statments, but all of them can be recognised by the question mark ? that they start with. The important difference between printout statements and query descriptions is that the former do not restrict the result set in any way: even if some printout has no values for a given page, an empty field will be printed, but the page is still part of the result.

4.2.1 Printing property values

The most common form of printout statements are property printouts, that make SMW display all values assigned to a certain property. These are written simply as a question mark followed by the property name, e.g.

```
?population
```

prints the values for «population» of all query results. On Special:Ask, the result of each printout is shown in a table column that is labelled by the name of the property. It is possible to change that label for a printout, and this will be very useful when using queries on wiki pages (it is not really relevant on Special:Ask of course). The equality symbol is used to change the label:

```
?population = Number of inhabitants
```

The above still prints population, but with the modified label in the table header. As mentioned above, property printouts may have an empty result for some pages, e.g. if something does not have any population. Property conditions with wildcards (see above) can be used to ensure that all elements in a query result have some value for the printed property, if this is desired.

4.2.2 Printing categories

There are two ways to print category information: either SMW prints all categories assigned to some page, or SMW checks for one particular category. The first case is achieved by the printout

```
?Category
```

where «Category» is the name of the Category namespace in the local language. This printout will show all catgories that are directly used on a result page. The other option is to ask for one particular category, such as

```
?Category:Actor
```

The result then will contain a column «Actor» that contains X for all pages that directly belong to that category, and is empty otherwise. Again, one can change the label using equality:

```
?Category:Actor = A
```

will merely display an «A» as the header of the result column which might be more sensible given that the entries in that column are very short. It is also possible to change the way in which this kind of category queries are formatted, as described below.

4.2.3 The main result column

All queries by default display the main list of result pages in the first column. In some cases, it can be useful to move that to another position. This is not relevant for Special:Ask, but can be quite useful in inline queries. A special printout statment is available for this purpose:

```
?
```

This single question mark addresses the «unlabelled result column» that shows the main result list. As before, different labels can be assigned with the equality symbol, e.g.

```
? = Results
```

4.2.4 Display format

Many printout statements can be further customised by giving a *printout format* which can be given after a property name, separated by the symbol #. The available formats depend on the type of the printout and involved property.

4.2.4.1 Plain (unformatted) printouts

A general format that is supported by most types of printouts is the *plain format* (or *empty format*), available since SMW 1.4.3. Printouts with this format will avoid all forms of beautification or linking in their presentation, and return a plain value instead. This is particularly useful when results are further processed in templates or parser functions. To select the plain output format, a hyphen ("-") or simply nothing is used as a printout string, as in the following examples:

```
?population# -

?capital #
```

Both printouts select the plain format. Spaces do not matter and can be inserted to increase readability. For numerical properties like the population number, the plain format is a simple number string without commas to separate digits. For properties of type page, the plain output is simply the name of the page without any link.

4.2.4.2 Formats for specific printout types

For properties that support units, queries can thus determine which unit should be used for the output. To print the height in cm, e.g., one would use the following:

```
?height#cm
```

this assumes that the property height is aware of the unit «cm». For properties of type date, the output format "ISO" is available to obtain results in a technical format that conforms to the ISO 8601 standard. Other datatypes may have different printout formats. See the types documentation for details.

For printouts of the form `?Category:Actor`, the display format can be used to modify what SMW will display for cases where a page is (or is not) in the category. The following is an example:

```
?Category:Actor#an actor, not an actor
```

This will show the text «an actor» for all pages that are actors, and the text «not an actor» otherwise. This can, for example, also be used in combination with small images to display icons for certain categories.

4.3 Concepts

It is possible to store queries in Semantic MediaWiki on dedicated pages, called **concepts**. These pages can be viewed as «dynamic categories», i.e. as collections of pages that are not created manually, but that are computed by SMW from the description given by a query. An example could be the concept of European cities. In traditional MediaWiki installations, one may have a category called *European cities* that holds all such cities. In SMW, one would instead define the concept «European cities» by saying that it contains all cities that are located in Europe. No city page needs to be changed, and yet one can create many concepts about cities (such as «capital», «Italian city», or «large coastal city located at a river»).

4.3.1 Creating a concept

A concept is a page in the Concept: namespace that is always described by a semantic query, as explained in Help:Semantic search. For example, the Concept:Semantic Web events 2008 describes certain events in 2008. Its concept page contains the following text to do that:

```
{{#concept:  [[Category:Event]] [[start date::> Jan 1 2008]] [[start
date::< Dec 31 2008]]

| Events in the year 2008 that have been announced on semanticweb.org.

  To add more events, go to the page "Events" on semanticweb.org.

}}
```

The parser function #concept is used to define concepts. Its first parameter is a concept description. Its second parameter is a short text that describes the concept. This description is optional and can also be left away. It is exploited in some uses of concepts in SMW to have a concise short description of the concept (e.g. as a default description in RSS feeds). The complete concept page will then show this data, and give a preview of the results.

It is possible to have other content on the concept page as well. Any normal wiki text can go before and after the use of #concept but it will not have any effect on the definition of the concept. The #concept parser function can only be used on pages in the Concept: namespace, and it can only be used once on each such page.

4.3.2 Using concepts

Concept pages as such can be browsed to view the contents of some concept, similar to category pages. But they can also be used in other semantic queries just like categories. For example, the following query would show all pages in the above concept of events which are also located in Germany:

```
[[Concept:Semantic Web events 2008]] [[located in Germany]]
```

Note that this would look almost the same if we would have a category called «Semantic Web events 2008». therefore, concepts are also like stored queries that can be reused in other queries if desired.

SMW's inline queries may also use concepts, and in some cases even the concept description is used to beautify an output. Concept descriptions are also included in SMW's RDF export in form of OWL class descriptions, so that other Semantic Web tools can download and reuse the concept descriptions.

4.4 Inline Queries

Semantic MediaWiki includes a simple query language for semantic search, so that users can directly request certain information from the wiki. Readers who do not wish to learn the query syntax can still profit from this feature: **inline queries** dynamically include query results into pages. So queries created by a few editors can be consumed by many readers.

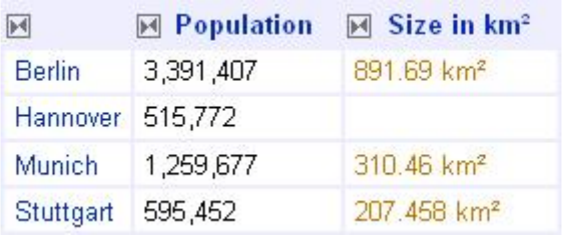
Inline queries are similar to other semantic search features, and can also be restricted on a site in order to ensure sufficient performance. Since inline queries exploit the existing caching mechanisms of MediaWiki, most requests for a page with such dynamic contents can be served without any performance impact whatsoever.

4.4.1 Introduction to #ask

The basic way of writing an inline query is to use the parser function #ask. The query string (See selecting pages for syntax) and any printout statements are directly given as parameter, like in the following example:

```
{{#ask: [[Category:City]] [[located in::Germany]]

| ?population

| ?area#km² = Size in km²

}}
```

Here we query for all cities located in Germany, and two additional printout statements are used (a simple one and one with some extra settings). This displays the following result on a page:



	Population	Size in km²
Berlin	3,391,407	891.69 km²
Hannover	515,772	
Munich	1,259,677	310.46 km²
Stuttgart	595,452	207.458 km²

It is common to put the query as the first parameter behind #ask:. All other parameters are separated by |, just like for other parser functions. The exact formatting of the inline query is not essential, but it is good to use line breaks to make it more readable to other editors. As with all templates, one line per parameter, starting with the | is most accepted in practice.

Note that all the arguments to the #ask: function are ignored by the page parsing, hence the above example does not add a category or a «located in» property annotation to this page. A few more things to note are:

- The pipe '|' symbol is used to separate the conditions from the property to display.
- The conditions for display are a single argument to the #ask function, so there are no '|' symbols between them.
- White space and line breaks can be used within the #ask function, SMW is fairly flexible there.

- The format of the results display changes when you request display of additional properties. SMW picks an appropriate default format for query results, but you also have detailed control of the appearance of query results.

Knowing the basics of query string and printout statements therefore is enough to write many kinds of queries. But there are many cases where the standard table output of a query may not be the best choice, or where further settings are desired (like the maximum number of results that should be displayed). For this purpose, inline queries have a number of other possible parameters that one can use to control their appearance in detail. The general syntax for #ask therefore is the following:

```
{{#ask: argument 1 | argument 2 | … }}
```

Most of this page explains the various arguments one may use in inline queries.

Prior to Semantic MediaWiki 1.0 there was a different syntax for inline queries using an <ask> tag, which is still enabled on some wikis. Please see the old documentation page for details on this feature. It is strongly recommended to use only the new syntax now.

4.4.2 The #show parser function

A common usage of queries is to display only a single property value for a single page. For example, one could insert the population of Berlin into some article, and use a query instead of manual copying to achieve this. SMW has a special shortcut to make such queries simpler. For example, one can write

```
{{#show: Berlin | ?population}}
```

to display the population of Berlin (Result: «»). The function otherwise works like an inline query, and all parameters available for inline queries can also be used on #show if desired. The above function can also be written as an #ask query as follows:

```
{{#ask: [[Berlin]] | ?population = }}
```

Here the equality symbol assigns another label for displaying the property, and this label is empty. Without this, the result would display «Population:» before the actual number.

4.4.3 Standard parameters for inline queries

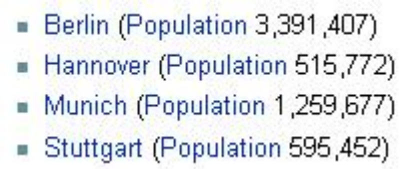
In general, an inline query is a request to find a number of **pages** that satisfy certain requirements. The query must answer three questions:

1. Which pages are requested? (query description)
2. What information should be displayed about those pages? (printout statements)
3. How should the results be formatted within the page?

The first two points are explained in their respective manual pages. The third point is important to smoothly include query results in pages, yet is largely independent of the first two. Without further settings, queries often produce tables like above or simple lists (if no additional printouts are used). An example of another possible format are bulleted lists, which one can create with the parameter format=ol:

```
{{#ask: [[Category:City]] [[located in::Germany]]

 | ?Population

 | format=ul

}}
```

This will produce the following output:



- Berlin (Population 3,391,407)
- Hannover (Population 515,772)
- Munich (Population 1,259,677)
- Stuttgart (Population 595,452)

SMW implements a wide variety of output formats for inline queries, and allows you to futher control results display using a MediaWiki template. The parameter format is one of the most important parameters for selecting the appearance of query results and will be explained in more detail below. The following table gives an overview of common parameters that can be used in basically all queries:

Parameter	Possible values	Description
format	a format name (see below)	selected output format; some formats allow further parameters (see #Result formats)
limit	non-negative number	maximal number of pages selected (in the case of a table: rows)
offset	number	where to start
sort	property name or a list of property names separated by ,	name of properties to use for sorting queries (see Help:Selecting pages)
order	ascending/asc, descending/desc/reverse, or a list of those if more than one property is used for sorting	defines how results should be ordered, only applicable if sort is used, ascending is the default (see Help:Selecting pages)
headers	show, hide	shows or hides the labels/headers

		used in some output formats such as «table», hide is default
mainlabel	plain text	title of the first column (the one with the page titles in it), default is no title; set to - to suppress printing the page titles
link	none, subject, all	defines which article names in the result are hyperlinked, all normally is the default
default	plain text	if, for any reason, the query returns no results, this will be printed instead
intro	plain text	initial text that prepends the output, if at least some results exist
outro	plain text	text that is appended to the output, if at least some results exist
searchlabel	plain text	text for continuing the search (default is «… further results»)

In addition to the above, some formats have their own parameters that control special aspects of the format. These special settings are described in the documentation of each format.

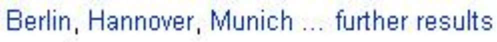
Result limits and links to further results

You can set the parameter limit to restrict the maximum number of results that are returned. For example, the query

```
{{#ask: [[Category:City]] [[located in::Germany]]

  | limit=3

}}
```

displays at most 3 cities in Germany. Even if you do not specify a value for limit, SMW always applies some limit to the results a query returns. Depending on a site's settings, it might be possible to increase the number of displayed results by specifying a higher value for limit. However, there is usually a maximum limit that cannot be exceeded, set by the wiki administrator based on performance considerations.

Running the above query produces:



Berlin, Hannover, Munich ... further results

This shows that whenever a query does not display all results due to a limit, it will normally show a link to «further results». The text of this link can be modified by setting the parameter searchlabel. If the value of searchlabel is empty, then no link to further results appears. Some output formats (see below) never display the search link, or display it only if a searchlabel was specified.

An interesting application of limit and searchlabel is to display *only* a link to the results of a search, without showing any result inline. You achive this by specifying a limit of «0» or «-1». For instance, the query

```
{{#ask: [[Category:City]] | limit=0 | searchlabel=Click to browse a list of cities }}
```

displays: Click to browse a list of cities. this link will only appear if there are any results at all. In other words, SMW will still compute the query to check if there are any results. If this is not needed, or if a link should be shown in any case, one can use the limit «-1». SMW will then only print a link to further results, even if no results exist at all. This also saves some computation time on the server.

4.4.3.1 Introduction and default text

If no articles satisfy the conditions of a query, *nothing is shown*. This is sometimes a useful behaviour, but often certain texts should be shown or not shown depending on whether the query has results or not. For example, one may want the query to show an output of the following form:

```
Upcoming conferences: ISWC2008, IJCAI2007, …
```

where the list of conferences is generated by a suitable query. If the query (for whatever reason) would not return any results, the page would look as follows

```
Upcoming conferences:
```

which is not desirable. Two parameters exist to prevent this.

- default: this parameter can be set to a default text that should be returned when no results are obtained. In the above example, one would probably write something like

```
Upcoming conferences: {{#ask: ... | default=none}}
```

so that, if no result is obtained, the article will display

```
Upcoming conferences: none
```

- intro: this parameter specifies a text that should be prepended to the output of a query, but only if one or more results exist. In the above example, one could write

```
{{#ask: ... | intro=Upcoming conferences:_}}
```

so that, if no result is obtained, nothing will be printed at all. Note that we use «_» to encode the final space. This is needed for initial and final spaces in any parameter, since those are otherwise removed internally (this is always the case in MediaWiki and is not specific to SMW).

Both of the above solutions will show the intended output if results are found. It is also possible to combine both parameters if desired. The parameters can also include MediaWiki markup, such as links or templates, as long as this does not confuse MediaWiki in recognising the #ask function.

Also note that if the set of pages selected in a query is empty, no header row or blank line, not even any blank space, is produced. This can also be useful to «hide» queries that are not applicable. **However, it is not recommended to insert great amounts of queries into every page, based on the assumption that this can do no harm since no output is generated.** Indeed, answering queries requires much computational resources and should not be done without a purpose.

Using default texts for queries is also a good habit in general, since it may happen that a query will no longer have any results in some future, e.g. due to changes in the way the wiki organises its data. Such queries that once worked properly may be forgotten so that nobody notices the query on a page labouring to display nothing.

4.4.3.2 Sorting results

It has been explained in Help:Selecting pages that query results can be ordered by one or more properties. As explained there, Special:Ask has additional input fields to specify sort properties and ordering. In inline queries, sort properties are defined with the parameter sort, and the order can be set with the parameter order. The value of order should be ascending or descending, or one of the short forms «asc» and «desc», or «reverse». An example is the following query for the three largest cities in Germany:

```
{{#ask: [[Category:City]] [[Located in::Germany]]

 | ?Population

 | sort=Population

 | order=descending

 | limit=3

}}
```

As explained in Help:Selecting pages, sorting conditions may impose restrictions on the set of query results. In the above case, only German cities that have a value for population are considered. If more than one property is used for sorting, the parameters sort and order can be set to lists of property names and orders, repsectively, separated by commas. The following is an example:

```
{{#ask: [[Category:City]] [[Located in::Germany]]

 | ?State

 | ?Population

 | sort=State,Population

 | order=ascending,descending

}}
```

This query would return all German cities for which a state and population was specified. These results will be ordered by the name of the state they are located in (ordered alphabetically). Cities that are located in the same state will be ordered by their population, largest first («descending»).

Disjunctions in queries. SMW 1.2 allow you to use the keyword **OR** in inline queries to take the union of two queries in one result set. For example, the next query shows up to 100 countries and cities of Europe, ordered by their population:

```
 {{#ask: [[Category:Country]] [[located in ::Europe]] OR

         [[Category:City]] [[located in ::Europe]]

   | sort=population | order= descending

 }}
```

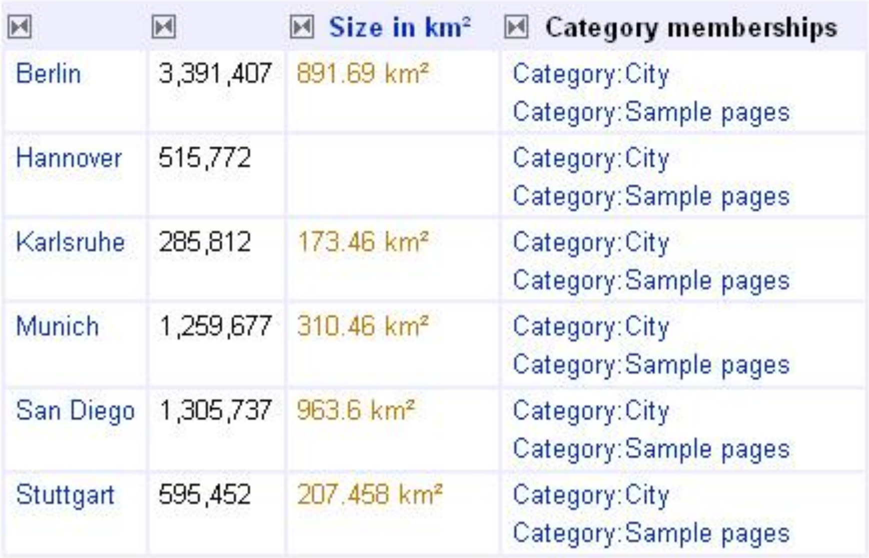
4.4.3.3 Configuring labels/table headers

Queries that return more than just the selected articles (e.g. the population in the above example) will display labels that describe the various output fields. By default, the label just displays the name of the requested property, or the text «Category» if categories are displayed. Labels for properties normally display as a link to the respective pages in the Property: namespace.

In the table format, the labels appear as column headers. In other formats, the labels might appear right before the output fields. The texts used for these labels can be controlled as explained in Help:Displaying information, using the equality symbol after printouts. Example:

```
{{#ask: [[Category:City]]

 |?Population=

 | ?Area#km²=Size in km²

 | ?Category=Category memberships

 | format=table

 | default=nothing found in Category:City

}}
```

This query will produce:



		Size in km²	Category memberships
Berlin	3,391,407	891.69 km²	Category:City Category:Sample pages
Hannover	515,772		Category:City Category:Sample pages
Karlsruhe	285,812	173.46 km²	Category:City Category:Sample pages
Munich	1,259,677	310.46 km²	Category:City Category:Sample pages
San Diego	1,305,737	963.6 km²	Category:City Category:Sample pages
Stuttgart	595,452	207.458 km²	Category:City Category:Sample pages

It is possible to use empty printout labels to have no label for a result column at all. In tables, however, the table header will still be shown even if all printouts use empty labels. To remove the header of a table completely, the parameter headers can be used. Two values are possible:

- show: display labels (default)
- hide: hide all labels and table headers

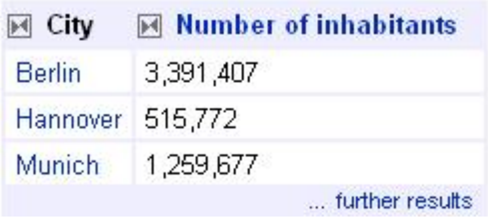
This setting works for tables as well as for other outputs. In the latter case, the value hide will hide all printout labels, even if they have a non-empty label set in the query.

4.4.3.4 Changing the first result column

Most queries by default display the actual result pages in the first result position, e.g. as the first column in a table. The header of this column is usually blank. To change the label, or to hide the whole first column, the parameter mainlabel can be used. Normally, the text given to that parameter will simply be used as a header for the first column, for example in the query

```
{{#ask: [[Category:City]] [[Located in::Germany]]

 | mainlabel=City

 | ?Population=Number of inhabitants

 | limit=3

}}
```

This will produce the table:

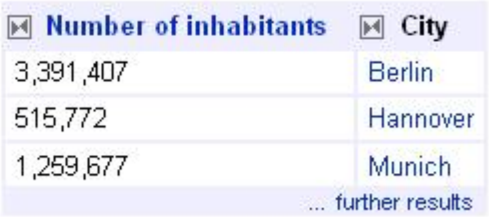


City	Number of inhabitants
Berlin	3,391,407
Hannover	515,772
Munich	1,259,677
	... further results

The parameter mainlabel can also be used to completely hide the first column. This happens if the value of this parameter is set to «-» (minus symbol). To insert the list of main results at another position, the printout statement «?», i.e. the question mark without any additions, can be used. For example, modifying the example above to display the city name after Population,

```
{{#ask: [[Category:City]] [[Located in::Germany]]

 | ?Population=Number of inhabitants

 | ?=City

 | mainlabel=-

 | limit=3

}}
```

This results in the table:



Number of inhabitants	City
3,391,407	Berlin
515,772	Hannover
1,259,677	Munich
	... further results

4.4.4 Result formats

The parameter format determines how the results of a query are displayed in the article. If it is omitted, all queries are displayed as tables (format table), unless there would be only one column, in which case the results are displayed as a comma-separated list (format list). In addition to the formats provided by SMW, extensions can provide additional formats; see Semantic Result Formats and Semantic Google Maps for two such extensions. The following formats are available in SMW by default:

Format	Description	Additional parameters (usually optional)
list	Comma-separated list, with additional outputs shown in parentheses	sep, template
ol	Ordered list, with additional outputs shown in parentheses	sep, template
ul	Bulleted list, with additional outputs shown in parentheses	sep, template
table	Tabular output	
broadtable	Tabular output, where the table is as wide as the article.	
category	List in columns, with first letters as section headers, in the style of MediaWiki category pages	sep, template, delim, userparam, columns
embedded	Embed selected articles.	embedonly (if set, don't show article link), embedformat (can be ol, ul, h1, h2 ..., h6)
template	Print results by passing result fields as parameters to a given template.	template (mandatory)
count	Just the number of results (a count of the number of matching pages), instead of the results themselves	

debug	Debugging information for analysing problems in query answering.	
rss	Print links to RSS feeds for query results.	title, description
csv	Export result table as CSV (comma-separated values), *available since SMW 1.2.1*	sep

4.4.5 Exporting query results: RSS, etc.

Some of the above formats especially Semantic Result Formats enable data export from the wiki. Since Semantic MediaWiki release 1.4.2 the iCalender format and vCard format is part of the Semantic Result Formats. Two aspects of those formats are special:

1. They do not show any results at the place where they are inserted.
2. They use fixed standard formats for exporting (non-fixed, free-form) wiki content. Hence it must be explained which wiki properties belong to which part of the data export format.

The first point means that only a link to Special:Ask will be shown at the place where the query is inserted. This link is similar to the normal «further results» link, but will use a more adequate default text (something like «RSS»). Yet it is possible to change the link text with the parameter searchlabel as for other queries.

The second point makes it necessary to relate printout statements (properties) to the data fields available in the export format. For example, *vCard* is a data format that can encode many kinds of contact data about a person, but it cannot represent arbitrary properties. To specify which wiki properties belong to which of the available data fields, the label of the property printout is used. For example, vCard supports (among many others) the data fields «firstname», «lastname» and «homepage». A query could thus be

```
{{#ask: [[Category:Person]]

  | ?firstname

  | ?lastname

  | ?url = homepage

  | format=vcard

}}
```

Here the wiki would have properties called «firstname» and «lastname», but the homepage of a person is stored in a property called «url». The label «homepage» is given to the latter so that vCard recognises the special meaning of this property. With this method, wikis can use arbitrary property names (in any language) and still export to standard formats. See the pages of the above formats for details on the data fields they support.

4.4.6 Using templates for custom formatting

Some of the above result formats support the use of wiki template to fully control the display of an inline query. This works for the formats template, list, ol and ul. If a template is specified, all result «rows» are formatted using this template. The name of the template (without the initial «Template:») is given in the parameter template, so the query has the following general form:

```
 {{#ask: ... | format=template/list/ol/ul | template=templatename }}
```

For each result in an inline query, SMW then calls the specified template, where the result and printout values are used as numbered template parameters. So a query that would display three columns when formatted as a table, will set three template parameters. These values can then be used in the template in the normal way writing {{{1}}}, {{{2}}}, etc. Each parameter refers to one "field" or column in the results that would be displayed by the inline query for each selected page. Normally the first field a query displays is the page title (see #Changing the first result column), so parameter {{{1}}} is the page title, and {{{2}}}, {{{3}}}, ... are the other properties displayed by the query. A number of examples are given below.

The template feature allows greater flexibility in the display of the query, including:

- Changing the order in which output is displayed, or omitting or duplicating output;
- Displaying images depending on query results;
- Creating links for property values;
- Using CSS styles to vary font-size, alignment, background-color, etc. by column in tables.

If you do use a template to adjust the appearance of links, you will probably need to set the parameter | link=none | to disable SMW's automatic linking of article names; your template will then have to add [[ ]] around anything you want to be a link.

To understand how to create a template for formatting some query, it is useful to look at the query with format=table first. For example, queries that refer to a single page only (like the ones one would use with #show) hide the page title of the result page, so that the parameter {{{1}}} refers to the first printout statement. Using the printout statement ? or specifying any value for mainlabel willchange this.

The following examples all use Template:Query output demo that basically contains the following wiki text:

```
{{{2}}} people squeeze into the {{{3}}} of {{{1}}}.
```

The following queries illustrate the effect of this template:

```
{{#ask: [[Category:City]] [[Area::+]] [[Population::+]]

  | ?Population=Inhabitants

  | ?Area#km²=Size in km²

  | format=template

  | template=Query output demo

  | limit=3

}}
```

Result:

In the above example you can see how the template ignores any header labels specified in the query such as «Size in km^2». Yet the values the template displays do use the units specified in ?Area**#km^2**=Size in km^2, and will similarly respect all given display formats (see Help:Displaying information).

Below is a similar query sorted by population that uses format=ol to produce a numbered list.

```
{{#ask: [[Category:City]] [[Area::+]] [[Population::+]]

  | ?Population

  | ?Area#km²

  | format=ol

  | template=Query output demo

  | limit=3

  | sort=population

  | order=desc

}}
```

Result:

If we directly specify a single page, then normally the query results do not include the page, so to reuse the same template in the query below we must tell the query to display the page title as the first column by adding |?

```
{{#ask: [[Berlin]]
```

```
  | ?

  | ?Population

  | ?Area#km²

  | format=template

  | template=Query output demo

}}
```

Result:

The same can be accomplished using #show even though this may not be the most typical use of this function:

```
{{#show: Berlin

  | ?

  | ?Population

  | ?Area#km²

  | format=template

  | template=Query output demo

}}
```

Result:

Templates may also include other parser functions such as conditionals and even queries. Examples of complex query formats can be found on the following pages (external links, may change):

- Upcoming events on semanticweb.org's Main Page: the events section on this page displays only certain major events. Each such event is formatted with a template that uses another inline query to find sub-events (co-located workshops, tutorials, etc.) for a given event.
- Publications on korrekt.org: all lists on this page are created with templated queries. Conditionals (#if and #ifeq) are used to change the format of a result depending on its publication type and provided data (majorpublications have boldfaced titles).

4.5 Basic Inferencing

Semantic search can be used to find pages based on the information that users have entered about them into the wiki. This simplifies many tasks, but it still requires that semantic information is entered manually in the first place. In some cases, one would expect the wiki to be «smarter» and to deduce some facts even if they were not entered directly. In some cases, SMW can draw such inferences automatically, as described in this article.

4.5.1 Subcategories

MediaWiki supports a hierarchy of categories that helps to simplify the category statements needed on each page. As an example, assume that a wiki uses categories «Person», «Woman», and «Man». It is obvious to a human that every page that belongs to the category «Woman» should also belong to the category «Person». But «Woman» is clearly more specific, and many wikis (including Wikipedia) have a policy to use only the most specific categories on some page — otherwise the page would often have to contain dozens of categories that are hard to maintain. To indicate that one category is more specific than another, MediaWiki supports a *category hierarchy* where users can state that one category is a subcategory of another, simply by putting a category on the subcategory's page, e.g. the page Category:Woman could contain the text

```
[[Category:Person]]
```

For details, see the MediaWiki handbook.

By default, SMW uses this subcategory information in semantic queries: when asking for all pages that belong to a category, it will also find all pages that belong to any subcategory of that category. In the above example, the query [[Category:Person]] would also return the pages in categories «Man» and «Woman». In other words, the actual query corresponds to

```
[[Category:Person]] OR [[Category:Woman]] OR [[Category:Man]]
```

If the category hierarchy is deeper, then SMW will also include further subcategories. For example, one may have a category «Mother» of all women that have children, and this again would be a subcategory of «Woman». Then the above query would retrieve all pages in category «Mother» as well.

SMW's mechanism of subcategory inferencing can be restricted or disabled by the site administrator. Normally, it supports only a certain maximal depth of category hierarchies, so it may not return all results if there are very long chains of subcategories involved. Using a manually created query with OR as above is a work-around in this case, but it does of course not take into account any changes in the category hierarchy.

In some cases, wikis use categories and category hierarchies that are not suitable for being treated in the above way. For example, Wikipedia uses a category called «Cities» not for collecting all cities but for all articles that are related in some ways to cities. Even the category «Cities in Canada» is used to collect all pages that have some relationship with that topic. This is not an actual problem of categories or category hierarchies: the semantic query [[Catgegory:Cities]] still returns all pages related to that topic, it just does not return actual cities only. So one might argue that the name of the category is confusing in some sense, but this is merely a matter of how to organise a wiki. If a wiki has no category for actual cities as such, then no semantic query can produce all cities directly.

A more serious problem in large wikis might be what is called «semantic drift». This occurs if the exact intention of some category is not really specified, e.g. because it lacks a detailed description on its page. Different users then may have slightly different readings of the categories meaning, and this may influence how they use sub-category statements. For example, some editors may reasonably say that «Priest» is a subcategory of «Religious office» (referring to the job category), while others may deem «Female priest» to be a subcategory of «Priest» (referring to the class of people having that job) – but this would imply that all pages in «Female priest» are also implicitly categorised in «Religious office», thus confusing people and job occupations. It is therefore important to always clearly describe on a category page what should go into a category and what shouldn't, and also to point to alternative catgegories that may be suitable.

4.5.2 Subproperties

Just like categories, also properties can be more specific than one another. For example, a wiki may have the property «capital of» to relate cities to countries, and a property «located in» that generally describes that some city is located in some country. Now it happens to be the case that every capital city also must be located in the country that it is capital of. In other words, «capital of» is a *subproperty* of «located in». Whenever a user states that a page is a capital of some country, SMW should then also conclude that the page has an (implicit) «located in» relation to that country as well. To say that in the wiki, the following can be entered on the page Property:Capital of:

```
[[subproperty of::Property:located in]]
```

Once this has been stated, a query [[located in::Germany]] will also return the capital Berlin even if no «located in» property is given on that page. Similar considerations as in the case of cateogries apply, and detailed descriptions on property pages are a good method for avoiding semantic drift.

4.5.3 Equality of pages: redirects

It often happens that a thing can be referred to by different names, such as in the case of Semantic MediaWiki which is synonymous with SMW. In MediaWiki, this is solved by *redirects* that forward readers from one page to another. But synonyms may be even more important in a semantic wiki, where one wishes to organise content and make it more accessible. If different editors use different page names in annotations, then it is hard to create queries which will still display a unified view on the wiki content.

SMW therefore treats all redirects between pages as synonyms. This means that it does not matter at all whether a redirect page or the actual target page is used in a query or annotation. SMW internally uses only redirect targets to work with, and all functions will take the redirect structure into account. This mechanism works only for immediate redirects: redirects that point to other redirect pages are not supported and should be resolved (this is also the case in MediaWiki anyway).

Since SMW 1.2, it is also possible to use redirect on properties and categories with the same effect, so multiple synonyms for properties can be created. It is not suggested to use that feature for the case of categories though, simply because MediaWiki's category functions will still ignore category redirects such that some wiki features will not work as expected. Redirects between normal pages and properties, properties and categories, etc. are not supported in a special semantic way. They still create normal MediaWiki redirect pages but nothing else.

4.5.4 Inferencing and printout statements

Printout statements do generally not perform any inferencing, i.e. they will only return the statements that are explicitly made for some page. This is desired in some situations, and it may be a limitation in others. A work-around can be to use a tempalte for annotation, and to give two property values explicitly in that template, essentially by writing something like

```
[[capital of::located in::Germany]]
```

which is the same as writing [[capital of::Germany]] and [[located in::Germany]], but it will show only one link to Germany.

4.5.5 Inferencing features that are not supported

It sometimes happens that ambitious contributors in a wiki will create properties that also suggest a specific meaning for automated deduction. It should therefore be noted that SMW does *not* support any of the following features:

- Transitivity
- Inverse properties
- Domain and range restrictions
- Number restrictions and functional properties

Even if properties that sound like the above are introduced, and even if these are linked to well-known properties in ontology languages such as OWL, RDFS, SKOS, etc., SMW will not use these annotations to perform smarter queries. To prevent confusion, it is suggested to not use names that resemble established notions in existing ontology languages, or at least to clearly document this limiation on the property pages.

To some extent, one may be able to craft queries to achieve a similar effect. The sample pages Germany and California show examples of queries for inverse relationships; the sample page Germany shows an example of a subquery that approximates a transitive relationship to some extent.

5 Editing

This section explains how to **edit pages** in Semantic MediaWiki. As explained in the introduction, SMW introduces special markup elements which allow editors to provide «hints» to computer programs on how to interpret some piece of information given in the wiki. Such hints are called **semantic annotations** and they are created with a special markup of SMW. Besides this, editing in SMW is just the same as in MediaWiki. Users who are not familiar with basic editing yet, should first read about how to edit pages in MediaWiki. Editors may or may not provide annotations on wiki pages as they like – it is an added feature that is completely voluntary.

5.1 Overview of SMW editing features

Annotations in Semantic MediaWiki can be viewed as an extension of the existing system of categories in MediaWiki. Categories are a means to *classify* articles according to certain criteria. For example, by adding to an article, the page is *tagged* as describing a city. MediaWiki can use this information to generate a list of all cities in a wiki, and thus help users to browse the information.

Semantic MediaWiki provides a further means of structuring the wiki. Wiki pages have links and text values in them, but only a human reader knows what the link or text represents. For example, «is the capital of Germany with a population of 3,396,990» means something very different from «plays football for Germany and earns 3,396,990 dollars a year». SMW allows you to annotate any link or text on the page to describe the *meaning* of the hyperlink or text. This turns links and text into explicit **properties** of an article. The property capital of is different from on national football team of, just as the property population is different from annual income.

This addition enables users to go beyond mere categorisation of articles. Usage and possible problems with using these features are similar to the existing category system. Since categories and properties merely emphasize a particular part of an article's content, they are often called *(semantic) annotations*. Information that was provided in an article anyway, e.g. that Berlin is the capital of Germany, is now provided in a formal way accessible to software tools.

The user manual explains basic annotations with properties, the creation of custom units for numerical properties, and the use of MediaWiki templates to simplify annotation.

Besides annotations, SMW also allows editors to embed **semantic queries** into articles. Thereby, readers of the wiki can view ready-made query results without having to learn the SMW query language. This feature is explained in the section on inline queries.

Categories

Categories are an editing feature of MediaWiki, and the main reference for their use is the MediaWiki documentation on categories. Categories are used as universal "tags" for articles, describing that the article belongs to a certain group of articles. To add an article to a category Example category, just write

```
[[Category:Example category]]
```

anywhere in the article. The name of the category (here, "Example category") is arbitrary but, of course, you should try to use categories that already exist instead of creating new ones. Every category has its own article, which can be linked to by writing . The category's article can be empty, but it is strongly recommended to add a description that explains which articles should go into the category.

On wikis like Wikipedia, categories are used for many different purposes. For example, the Cities category contains both individual cities, related subcategories like "City nicknames" and abstract concepts like "Digital city".

In Semantic MediaWiki-using sites, categories tend to be used much more sparingly, since inline queries make many categories superfluous. For example, a subcategory like Large cities could be replaced by a query for articles with Category:Cities with an area larger than 10 km², or a population larger than 1,000,000. In addition, categories tend to be used more exactly: a page like "Digital city" might end up in a category like "City-related terms" instead of "Cities", so that it wouldn't show up in a query on the "Cities" category.

5.2 Properties and types

Properties and types are the basic way of entering semantic data in Semantic MediaWiki. Properties can be viewed as «categories for values in wiki pages». They are used by a simple mark-up, similar to the syntax of links in MediaWiki:

```
[[property name::value]]</nowiki>
```

This statement defines a *value* for the property of the given *property name*. The page where this is used will just show the text for *value* and not the property assignment.

Existing links can be directly augmented with such property information, while other **types of data** (such as numbers or calendar dates) need an additional editing step.

5.2.1 Turning links into properties

Consider the Wikipedia article on Berlin. This article contains many links to other articles, such as «Germany», «European Union», and «United States». However, the link to «Germany» has a special meaning: it was put there since Berlin is the capital of Germany. To make this knowledge available to computer programs, one would like to «tag» the link

```
[[Germany|Germany]]
```

in the article text, identifying it as a link that describes a «capital property». With Semantic MediaWiki, this is done by putting a property name and **::** in front of the link inside the brackets, thus:

```
[[capital of::Germany]]
```

In the article, this text still is displayed as a simple hyperlink to «Germany». The additional text capital of is the name of the **property** that classifies the link to Germany. As in the case of categories, the name of the property is arbitrary, but users should try to re-use properties that already appear elsewhere.

To simplify this re-use, every property has its own article in the wiki, just as every category has an article. You can see all the properties in use in the wiki with the Special:Properties page. Just as category articles are prefixed with Category:, all property articles are prefixed with Property: to distinguish them from other articles. So you can also also use MediaWiki's Special:Search page to find existing properties. As with categories, a property's article can be empty, but it is strongly recommended to add a description that explains the intent of the property and its proper usage.

There are various ways of adding properties to pages:

What it does	What you type
Classify a link with the property "example property."	`Classify a link with the property "example property."`
Make alternate text appear in place of the link.	`Make alternate text appear in place of the link.`

To hide the property Link from appearing at all, use a space as the alternate text.	`To hide the property link from appearing at all`
	`use a space as the alternate text.`

	Note: The space after | is necessary. If left away, the MediaWiki pipe trick applies, but rarely with desirable effects. Even if a space is given, SMW will not print anything, which should be the desired result in most cases (to make a space appear in the text, use &ndash; as a space symbol).
To make an ordinary link with :: without creating a property, escape the markup with a colon in front, e.g. The C++ .	`The C++ operator  .`
To assign one value to multiple properties, add :: between each name, e.g. link.	`e.g. link.`

5.2.2 Turning values in text into properties

There is other useful information in wiki articles besides links to other articles. For example, there is a number in the Berlin article giving its population. To make this knowledge available to computer programs, one would like to "tag" the text

```
3,396,990
```

in the article, identifying it as a value for the "population property". With Semantic MediaWiki, this is done by putting the property name and :: in front of the text and surrounding it with [[ ]] brackets, thus:

```
[[population::3,396,990]]    .
```

This works fine. However, it creates a link to a *3,396,990* page, and having an article for every population value probably does not make sense. Furthermore, if you wanted to create a list of all German cities ordered by population, numeric order is different from the alphabetical order that you would expect for article names. For example, in alphabetical order, "1,000,000" comes before "345". We want to be able to tell Semantic MediaWiki that "population" is a number, not a link to a page in the wiki. The way to do this is to specify a **type** for the "population" property; see the next section for more information.

5.2.3 Data types for properties

Semantic MediaWiki has several built-in *data types* that we can choose for properties. For our previous population example, the appropriate type is called Type:Number; the prefix "Type:" is again a separate namespace that distinguishes descriptive articles about types from normal pages. We want to give property "population" a special property that specifies it has "type:number". To support this Semantic MediaWiki has a built-in special property called Property:Has type. We use the same syntax for this special property as for any other property, so in the Property:Population article, we write:

```
[[has type::number]]
```

(You don't need to specify the Type: namespace here.)

Semantic MediaWiki knows a number of *special properties* like Property:has type. Regardless of whether these properties have their own articles in the wiki, they have a special built-in meaning and are not evaluated like other properties.

Data types are very important for evaluating properties. Firstly, the data type determines how tools should handle the given values, e.g. for displaying values and sorting values in search results. Secondly, the data type is required to understand which values have the same meaning, e.g. the values "1532", "1,532", and "1.532e3" all encode the same number. Finally, some data types have special behavior, as will be described below. For these reasons, every property used should be defined with a data type.

The reason we didn't have to specify a data type for the "capital of" property above is that the default data type is Type:Page, which displays as a link. Even though Type:Page is the default, you should explicitly specify a datatype for every property, just to prevent confusion or later redefinition with an unintended type.

The same mark-up for properties that are links to pages also works for properties of other datatypes. Here are some more examples.

What it does	What you type
Assign the value 1,234,567 to the property "example".	`This page has a value for "example" of [[example::1,234,567]] ."`
Assign a numeric value, but showing different text in the article.	`This page has a value for "example" of [[example::999,331|about a million]] .`

Specifying the type in a property's article, e.g. This property is a number.	This property is a [[has type::number]] .
Combining MediaWiki markup with property values, e.g. John's username is john *This is not recommended; you should use a template instead.*	John's username is [[username::john|[mailto:john@example.com john]]].

5.2.3.1 List of data types

Using different types, properties can be used to describe very different kinds of values. A complete list of available types is available from Special:Types. The available types are:

- Type:Page - links to pages (the default)
- Type:String - text strings
- Type:Number - integer and decimal numbers with optional exponent
- Type:Boolean - restricts the value of a property to true/false (also 1/0 and yes/no)
- Type:Date - specifies particular points in time
- Type:Geographic coordinate - describes geographic locations. It recognizes different forms of geographic coordinates.
- Type:Text - like Type:String but can have unlimited length; the tradeoff is values of this type cannot be selection or sort criteria in queries..
- Type:Code - like Type:Text but displays its value in a HTML pre-formatted box. The value displays as regular text everywhere else (query results, factbox, "Pages using the property", etc.).
- Type:Temperature - can't be user-defined since converting temperature units is more complicated than multiplying by a conversion factor.

For specifying URLs and emails, there are some special variations of the string data type:

- Type:URL - displays an external link to its URL object.
- Type:Annotation URI: properties of this type are interpreted as relations to external objects, denoted by the URI. They are special since they are interpreted as *annotation properties* on export. See the type's page for documentation.
- Type:Email - displays an e-mail address as a link (with mailto:).

5.2.3.2 Enumerations and "Allows value"

SMW does not have an "enumerated" data type, i.e. a type that has a limited set of allowed values; instead, for any property, you can limit its possible values by using the special property Property:Allows value to enumerate its permitted values. This works for every data type.

5.2.3.3 Units

Type:Number allows a unit after the numeric value to distinguish values (e.g. "30.3 mpg" versus "47 km/liter"), but does not know how to convert between them. To support automatic conversion and multiple unit formats, you can define your own datatype with custom units. These automatically convert values to and from standard representations, so that users are free to use their preferred unit in each article yet still query and compare with property values in other articles.

5.2.3.4 Properties with multiple types

In human language it is easy to introduce multiple facts at once. For example, "John F. Kennedy was the 35th president of the U.S.A., serving from 1961 until his assassination in 1963." This is information about John F. Kennedy that belongs in his wiki page, but it shifts to information about his presidency. You could simply have a property "Presidency_details" of Type:String and put the text in it. But it will only be meaningful to humans, you can't query on it or sort it to produce a list of presidents.

You can't nest semantic annotations, so you cannot have a string property that contains additional annotations.

Often the best way to represent this is to create an article for the object of the property, so this can be annotated with the additional information. So the property "Has_presidency" would be of Type:Page, and then the article "Presidency of JFK" would have the properties "Of country::U.S.A.", "Count::35", "Start date::1961-01-20", :End date:1963-11-22", etc. Wikipedia frowns on so-called "stub" articles, but in a semantic wiki they are appropriate as they provide information for semantic queries and browsing.

It is also possible to create a property in Semantic MediaWiki that takes multiple values; these are sometimes called "n-ary relations". So you could create a property "Has presidency" that would have Type:Page; Type:Number; Type:Date; Type:Date, where the four values represent the country, the count of the presidency, the start date, and the end date. See Help:Many-valued properties for more information.

5.2.4 Special properties

We mentioned the special property Property:Has type that you use to define the data type of a property. SMW has other predefined special properties that have special meaning (even if you do not create property pages for them in your wiki). You *cannot* use these names for your own properties, but since SMW 1.4.0 you can use them in browsing and querying interfaces just like all other properties. For more information, see and the individual property pages.

5.2.5 Silent annotations using #set

Instead of using the standard double-brackets markup, you can also define semantic data using the #set parser function. This function takes in pairs of property names and values and stores them semantically; but it does not print anything to the screen. An example call would be:

```
{{#set:population=3,396,990|country=Germany}}
```

The #set call is specifically helpful when trying to save a String or Text value that contains square brackets, such as wiki-links; such brackets often don't work with conventional SMW markup.

5.2.6 Defining recurring events

Another type of complex data is recurring events, which are events that have multiple dates, defined according to a preset rule (such as a weekly meeting). You can define the date values for these using the #set_recurring_event parser function, which, like #set, is "silent" and does not print anything. An example call would be:

```
{{#set_recurring_event:property=Has date|start=January 4, 2010|end=June 8,
2011|unit=week|period=1|include=March 16, 2010;March 23, 2010|exclude=March
15, 2010;March 22, 2010}}
```

5.3 Custom units

This page explains ways that pages can have more control over the display and conversion of units. Units can be used for properties of Type:Number, and make annotation more flexible: everybody can view and enter data in his or her preferred unit without restricting mutual understanding. For example, some people might prefer a distance given in "miles" over one given in "km". In other cases, it might not be suitable to display a distance in "km" if "microns" are more appropriate.

5.3.1 Custom types with unit conversions

SMW has built-in support for some types that can handle units (e.g. Type:Temperature), but many more can easily be added. Types that support units can accept, convert, and display values in several different units. You can see this in the factbox of articles like "Berlin", where the area is given in multiple units.

In order to support such features, SMW needs to know how to convert values from one unit into another. This is rather easy in many cases, but can also involve more complex computation in other situations. We distinguish two cases:

1. Conversion between the desired units is *proportional*, i.e. you just have to multiply one value with a fixed conversion factor to get the value in another unit. For example, this is the case for converting between kilometres and miles.
2. Conversion between units is *not proportional* and more complex computations are needed. For example, this occurs for temperatures, since you need to add and multiply in order to get from °C to °F.

For all unit conversions of the first kind, you can easily create custom types that support those units. For the second situation, we discuss some possibilities below.

5.3.1.1 Creating new datatypes with propertional unit conversion

Before thinking about creating a new datatype, make sure that the type does not already exist by consulting Special:Types.

If the desired type does not exist, you can create a new one easily. First, you need to create a page in the Type namespace. For example, you might want to create Type:Power. In the new article, you should first write some sentences on the purpose and use of this new type. In our case this would say that we mean the physical quantity that is usually measured in *Watt*. This also helps others to find and reuse your type when searching for keywords.

To specify the supported units, you use a special property corresponds to. For example, to specify the **main unit** of the new datatype for power, we add

```
[[Corresponds to::1 W]]
```

The value "1 W" states two things: (1) the type understands the unit "W" and (2) the unit "W" is its main unit (that is what the "1" is for). Intuitively, the statement says "One quantity of this type corresponds to 1W." It is easy to specify further units, e.g.

```
[[Corresponds to::0.001 kW]]

[[Corresponds to::0.0013410220 hp]].
```

This says that the type also understands the units "kW" and "hp", and that 1 quantity of the main unit corresponds to 0.001 kW and to 0.0013410220 hp. In this way, you can support arbitrary units, so long as their relationship to the main unit can be described in this easy manner.

In many cases, there are multiple ways of referring to one unit. E.g. we would like to allow users to write "W" as well as "Watt" and maybe even "Watts". A short way of doing this is to separate additional units with "," instead of making multiple "corresponds to" statements with the same factor. For example we would write:

```
[[Corresponds to::1 W, Watt, Watts]]

[[Corresponds to::0.0013410220 hp,bhp,horsepower]]
```

The very first symbol in the "corresponds to:: 1 *xx*" is the main unit. After saving the page for the new type, the "corresponds to" statements will appear as special properties in its factbox. The type now can be used like any other type. For example, we could make a new Property:Engine power that is of type power, by adding to its page:

```
[[Has type::Power]]
```

This property will understand all the units defined in "corresponds to" statements in its datatype, and will show conversions between them (without duplicates, i.e. SMW does not dipslay "W" and "Watt"). Internally, the values will all be converted to the main unit, and the RDF export will show the value in this unit as well. The display of units within the wiki is highly customizable and need not involve the main unit, see below.

5.3.1.2 Unit conversions that are not proportional

Note that you can only specify a proportional conversion factor, a multiplier. So you cannot have different bases for different units, logarithmic scale conversions, etc. Thus you can't create a custom type for temperature which converts degrees Celsius to Fahrenheit. In the case of temperature, SMW already supplies a built-in Type:Temperature that handles this conversion, but this might not be true in other cases.

SMW does not allow specifying customized non-proportional units in the wiki. One workaround for this is to use Type:Number which also *accepts* unit strings after a given number. The type does not know how to convert between those values, but it still recognizes the unit and will be able to distinguish different units. If all users of a certain exotic type agree on using the same unit, the functionality would be similar to having real unit support. If someone still uses another unit, then the given value will at least not be confused with values in other units.

A more elaborate way of fixing the situation is to write a small script that implements the required conversion. It is not difficult to extend SMW in this fashion, and one could simply copy and adjust the code for Type:Temperature (which is below 70 lines, including comments). Upon implementing such a custom type, properties that needed to use Type:Number can be changed to the new type without any negative effects on existing articles. When confronted with unsupported units, custom types will still behave like simple number datatypes.

5.3.2 Customizing the display of units

Through the use of floating-point numbers, a single type can support a broad range of units. For example, a single length type can easily support both light years and nanometers. However, if someone uses a property "Elevation" to specify the height of a mountain, then it would hardly be useful to display this value in light years or nanometers.

SMW allows you to specify which units a property should display from all the units its type supports. This information is specific to the property : two properties can both use Type:Length while still having different appearances in the wiki. If no preferences on display are given, a property will display all of its type's units, with the main unit (the one with conversion factor 1) first.

To specify custom display units, add the special property display units to the property's page, mentioning each unit you want displayed separated by commas. For example, the article Property:Height could contain the statements:

```
[[display units::km,ft,miles]]
```

This results in the factbox displaying only those three units for values of the property Height, even though its Type:Length might support a dozen other units. Similarly, the tooltip for each such value will show those conversions. This customization works for all properties that use types with unit support, no matter whether the type was customized or built-in.

If you change the first display unit, consider displaying the type's "Standard Unit" to users as one of the other display units, since SMW still converts values to the standard unit when storing them.

5.4 Semantic Templates

Semantic templates are a method of including the markup that Semantic MediaWiki introduces through MediaWiki templates. This has several advantages:

- users specify annotations without learning any new syntax,
- annotations are used consistently, i.e. users do not have to look for the right properties or categories when editing a page,
- since templates also have other functions, such as rendering flashy infoboxes in an article, users are motivated to use them.

For these reasons, semantic templates are a very popular way of introducing semantics into a wiki.

5.4.1 Simple semantic templates – an example

Templates, with or without semantics, can have a very simple form. For example, when giving the value for surface area of an astronomical object in a wiki page, you might want it to display as

6.088×10^{18} m^2

which you can achieve by writing

```
 6.088 × 10<sup>18</sup> m²
```

This is cumbersome to write, so you might develop a Template:Surface_area for areas so that editors can simply write

```
 {{surface area|6.088|18}}
```

and the template expands to your desired markup.

Thus MediaWiki templates have immense value for normalizing and simplifying (once users understand the template syntax in general and particular) display in any wiki.

With the introduction of Semantic MediaWiki, you would probably want values for surface area to become semantic annotations so that they appear in the factbox and you can query them. So you might create a semantic property named Property:Surface area that uses or reuses a custom datatype Type:Area. Obviously you would like to have both the annotation and the visual appearance. You could write the following:

```
 [[Surface area::6.088e18 m²|6.088 × 10<sup>18</sup> m²]]
```

the semantic annotation uses the scientific format for a number that Semantic MediaWiki can parse, and the alternate text after the pipe '|' symbol is the complex display you want.

But this is even less user friendly, and highly error-prone. Instead, you could write or adapt the Surface_area template to hide the complicated markup *and* perform the semantic annotation. Then, just as before editors can write

```
{{surface area|6.088|18}}
```

which is much more readable. To achieve this, Template:Surface area is coded as follows:

```
 [[Surface area::{{{1}}}e{{{2}}} m²|{{{1}}} × 10<sup>{{{2}}}</sup> m²]]
```

See the sample page Sol and view its source to see this semantic template in use.

Note that the "surface area::" property in the template does not annotate the template article itself; it takes effect only on inclusion. This is because the default setting when installing Semantic MediaWiki is that SMW does not parse pages in the Template: namespace for semantic annotations . If this setting was changed (by the site admin) then you should surround the template code with <includeonly> tags to prevent the template's article itself from being annotated. As with a regular MediaWiki template, you can add text within <noinclude> tags to provide some user documentation on the template page.

5.4.2 Infobox-style semantic templates

Many MediaWiki sites make use of more complicated templates to present standard information. For example, Wikipedia articles on cities and villages use standard templates in which editors specify common items of information, such as this (from wikipedia:San Diego, California):

```
{{Infobox Settlement

|image_skyline            = Sandiego_skyline_at_night.JPG

|imagesize                =

|image_caption            = San Diego Skyline | March 31, 2007
```

```
|official_name              = City of San Diego

|settlement_type            = [[City]]

|nickname                   = America's Finest City

|motto                      = Semper Vigilans ([[Latin]]: Ever Vigilant)

|image_flag                 = Flag of San Diego, California.svg

...
```

Usually the template (in this case, wikipedia:Template:Infobox Settlement) displays this information in a nicely-formatted table. Such regular templatized items of information are ideal to be mapped into properties in Semantic MediaWiki, so that articles using the template will acquire semantic annotations without any changes to their pages.

The sample page California shows a simple "infobox" display template adapted to make semantic annotations.

5.4.3 Using semantic templates correctly

While the above pattern allows you to create all kinds of semantic templates of arbitrary complexity, there are some issues to consider.

5.4.3.1 Automatic annotation requires strict formats

You can annotate template fields automatically, but in this case the supplied values must adhere to the expected format. For example, it is a good idea to annotate the population of a city with a property of type number. However, in an infobox template such as the one at wikipedia:France, the entry supplied for population is not a single number, or even a set of numbers! Instead, there are multiple numbers and textual explanations of their meaning. Such special cases should be kept in mind when designing semantic templates.

Also, there are cases where existing templates can be evaluated in a quite semantic way, but the user still has to add semantic mark-up to make the data machine-processable. E.g. in the case of France, one might decide to leave "population" a normal text-entry, and leave it to the user to specify [[population::...]] where appropriate in this text.

5.4.3.2 Optional entries and conditionals

In the normal course of operating a wiki, most pages will probably not have their infoboxes completely filled, and you might not want to show empty rows in such cases. Blank rows can also lead to warning messages appearing on the screen, due to the fact that an empty value is annotated in such cases (although you can get around this by setting smwgInlineErrors to 'false'). To remove rows with blank values, you can add conditionals into the template code, which include a row (and its annotation) only if a non-empty value was provided.

This can be achieved with the help of the ParserFunctions extension to MediaWiki. This extension is independent of Semantic MediaWiki, and you can refer to the original documentation to this extension or further information. Wikipedia contains many examples of parser functions in templates, as for instance in wikipedia:Template:Taxobox. Using parser functions typically results in difficult-to-read template code, but the simplification for users can be substantial.

5.4.4 Queries in templates

Inline queries are often added to templates as well, and they almost always contain the {{PAGENAME}} variable; see e.g. ow:Template:Ask. Such queries are often used to aggregate a set of pages that have some property pointing to the page in question, such as the template for a country page using query to show a list of each country's cities.

5.4.5 Annotation in a template

You can also add annotations directly to templates, i.e. assign them semantic properties. Though this is not recommended, since templates don't represent a real-world entity, it can be done, if the site administrator enables it.

5.4.6 Editing templates

A popular extension, Semantic Forms, lets users add and edit the contents of templates using forms.

5.5 Service links

Semantic MediaWiki can provide links to online services when printing certain data in the Factbox and in property pages. For example, when an article contains a geographic coordinate, it is useful to provide links to online mapping services so that users can retrieve a map of that location with one click. Another example is to provide a link to an online currency converter for financial amounts. This page explains how you can add such features to a semantic wiki (without writing PHP code to support a new datatype).

5.5.1 Service links for properties

The information for additional links to online services in the factbox is associated with the property used. For example, this wiki's Property:Coordinates will show various links to online maps when it appears in the factbox, whereas other properties that also use Type:Geographic coordinate might not show this. This is crucial in many applications, since the datatype alone usually does not say much about the type of link. For example a property "IMDb number" might be used for a movie's id number at IMDb, but not every property of Type:Number should display a link to IMDb.

To make a property display service links, add the special property provides service on its page. For example, the article Property:Coordinates might include the annotation

```
[[provides service::online maps]]
```

Here, "online maps" is the name of a set of service links provided by the wiki. The next section explains how you specify these service links.

After you specify that a property provides service links, its property values in the factbox and on the property's own page will show an icon ⓘ that displays the service links when clicked.

5.5.2 Providing service links

In a nutshell, a wiki administrator puts the text specifying the appearance of service links in a special message article in the "MediaWiki" namespace named MediaWiki:Smw service *service name*. Continuing our example for coordinates, the text for [[provides service::online maps]] is in the message article MediaWiki:Smw service online maps.

Normally only users that have *sysop* (administrator) privileges in the wiki can add or edit pages in the MediaWiki namespace, hence only they can modify service links. This is a reasonable restriction for most wikis: since service links may appear in thousands of factboxes, they need to be trusted. Adding or modifying services should usually be discussed among many users before an administrator puts the change into practice.

All users, however, are free to associate properties with available services as described above.

5.5.2.1 The Mediawiki:Smw_service_*service_name* format

If you look at MediaWiki:Smw service online maps, though the message might be hard to read due to the long lines, its format essentially is as follows:

```
label|http://someurl.com

label text2|http://anotherurl.org

...
```

Every line contains one service link. The *label* is the text that users will see in the service link pop-up. After the pipe symbol '|' is the *URL* that the link will lead to.

In most cases, you want to provide information from the property value in the link. For example, a link to an online map service has to include the coordinates to display, and a link to a movie web site will have to include the ID of the movie. Since the exact data values are not known in advance, you use placeholders of the form *$1*, *$2*, *$3*, … in the URL. For example, the message text for a service link to IMDb could be:

```
IMDb|http://www.imdb.com/title/tt$1/
```

When SMW displays the service links for a property value, it substitutes the property value's information for these placeholders. In this IMDB example, a movie ID of Type:Number will replace $1 with the numeric value, and voilà, the service link for a movie links to its information on IMDB!

Since service links are typically perceived as "trusted resources," administrators must take care when formulating links, keeping in mind that users can accidentally or maliciously pass arbitrary URL-encoded strings to service link URLs in the place holders.

5.5.2.2 Information passed for each placeholder

The number and contents of the parameters that replace *$1*, *$2*, *$3*, … depend on the datatype of the property. For instance, a simple integer property replaces $1 with its value, whereas a geographic coordinate provides parameters for latitude, longitude, direction, and much more. In most cases, $1 is the most relevant parameter that just provides a URL-safe string version of the property value.

Type:Page

> $1: URL-encoded article name (no namespace), with underscores replaced by spaces

Type:Number (and types with custom units)

> $1: numerical value in English punctuation
>
> $2: integer version of value, in English punctuation
>
> $3: *From SMW version 1.1 onwards* unit, if any.

Type:String (but not Type:Text)

> $1: URL-encoded string

Type:URL, Type:Annotation URI and Type:Email

> $1: URL-encoded value of the URL(includes mailto: for Type:Email)

Type:Geographic coordinate

> $1: latitude integer degrees
>
> $2: longitude integer degrees
>
> $3: latitude integer minutes
>
> $4: longitude integer minutes
>
> $5: latitude integer seconds
>
> $6: longitude integer seconds,
>
> $7: latitude direction string (N or S)
>
> $8: longitude direction string (W or E)
>
> $9: latitude in decimal degrees
>
> $10: longitude in decimal degrees

$11: sign (- if south) for latitude

$12: sign (- if west) for longitude

Since geographic coordinates are so complicated, SMW includes a default message for MediaWiki:Smw service online maps — just add [[Provides service:online maps]] to any properties you have of Type:Geographic coordinate.

5.5.2.3 Display of Property:Provides_service

When adding a service links to a property with the special property "provides service", the property page's factbox should display a link to the message article for the service.

5.5.3 Extended example

To illustrate the whole process of creating and using a new service, we provide an extended example, also implemented on ontoworld.org. Articles about Semantic Web vocabularies such as FOAF contain information about the vocabulary's "namespace" and the online service Swoogle allows users to search for background information on such namespaces. Our goal thus is to add a new "Swoogle service" to the ow:Property:Namespace that is used on the vocabularies articles.

As a first step, we edit the article ow:Property:Namespace and add the line

```
As an additional service, this property provides a

[[provides service::Swoogle lookup]] of the entered namespace.
```

After saving, the factbox shows a link to the (still non-existing) service Swoogle lookup. Clicking this link, an administrator gets a new edit field, into which she enters

```
Swoogle
lookup|http://swoogle.umbc.edu/index.php?option=com_frontpage&service=diges
t&queryType=digest_ns&searchString=$1
```

The link was retrieved by using Swoogle and replacing the search string (at the end of the link) with the parameter "$1".

After those changes are saved, the new service is fully functional, and each page that uses ow:Property:Namespace will show a suitable link to Swoogle. Some articles will still show the old version, if they are retrieved from cache, but after the next edit or purge, all articles will display the links as expected.

6 Semantic Web

Although Semantic MediaWiki is designed to be used without additional technical background knowledge, it is closely related to well-known Semantic Web technologies. These technologies enable the wiki to share its knowledge with external applications by encoding it into the standard OWL/RDF format. Below is an overview of some related resources. These should be instructive for those who are familiar with semantic technologies or who wish to implement tools that use Semantic MediaWiki's RDF export.

RDF export

The explicit semantic content of Semantic MediaWiki is formally interpreted in the OWL DL ontology language, and is made available in OWL/RDF format. For further details on the exported format, see Help:RDF export.

Reusing vocabulary from external ontologies

Normally, all statements in the wiki refer to concepts and properties defined in the wiki, but it is also possible to directly use vocabulary from other sources. E.g. this wiki imports a number of FOAF elements for use within the wiki. A detailed description of how to use this feature is given in the article on vocabulary import.

Importing ontologies

Users with administrator status are allowed to import data from OWL DL ontologies into the wiki. This function is suitable for bootstrapping a wiki with existing semantic knowledge, so that articles about relevant topics are already available and include some basic annotations. Naturally, the import function cannot create sophisticated human-readable texts or import complex ontological statements that are otherwise not representable within the wiki. However, one can achieve good results in populating a wiki with a great number of articles. For further details, see Help:Ontology import.

SPARQL query service

The site can be configured to provide a SPARQL endpoint that enables expressive querying against the wiki's content. The wiki is primarily used for authoring "ABox" statements (i.e. simple facts concerning given individuals in contrast to complex schema information).

External reuse

Tools that can process OWL/RDF in a meaningful way (including many RDF tools), can also be used with Semantic MediaWiki. Help:Reuse lists a number of applications that have been tested with the wiki's output on one site or the other.

6.1 RDF export

Based on the user's semantic annotations of articles, Semantic MediaWiki generates machine-readable documents in OWL/RDF format, that can be accessed via Special:ExportRDF. Moreover, there is a maintenance script for automatically generating complete exports of all semantic data. This article explains how annotations are formally interpreted in the OWL ontology language, and how a suitable RDF serialisation is generated.

6.1.1 Using the export functionality

Users can easily access the generated RDF via the page Special:ExportRDF by entering a list of articles into the input field. The export will contain one OWL/RDF specification with various description blocks for exported elements. In addition to the requested articles, the export will also contain basic declarations for any further elements (such as mentioned instances, properties, and classes). There are two settings that further influence the set of exported articles:

- **Recursive export.** Every article usually has relations to various other articles. Normally, those other articles are just declared briefly such that tools can find further RDF specifications for them if desired. By enabling recursive export, all information about the encountered objects will be exported right away. Since this process is continued for all further objects, this option can lead to large results.

- **Backlinks.** The RDF data model is based on directed graphs. When exporting an article, one usually exports only the statements within which the corresponding element occurs as a subject, and the exported document does not include incoming links. This restricts RDF browsers, since they cannot access all elements that have some relationship to something without retrieving the whole RDF first. For this reason, one can enable the export of backlinks. All articles that have relations to any of the exported articles then will also be exported.

The server administrator can restrict the availability of the above options, and can set default values cases where no parameters can be given (see below). The reason is that the above options, especially in combination, can easily lead to the export of major parts of the wiki in RDF, which might overly impair the performance of large sites.

In addition to the form at Special:ExportRDF, one can also retrieve RDF by calling appropriate URLs directly. This is suitable for linking to RDF specifications directly. In its basic form, this is achieved by appending a (URL encoded version of an) article name to the URL of the export service. For instance, one can link to

```
http://wiki.ontoworld.org/index.php/Special:ExportRDF/ESWC2006
```

to get this RDF directly. Alternatively, the article name can also be specified as a GET parameter "page" within the URL, e.g.

```
http://wiki.ontoworld.org/index.php?title=Special:ExportRDF&page=ESWC2006
```

6.1.1.1 Additional GET parameters

In addition to title and page, ExportRDF has additional GET (query string) parameters.

- Backlinks can be enabled or disabled by setting "backlinks" to 1 or 0, respectively.
- Recursive export can be enabled or disabled by setting "recursive" to 1 or 0.

Both settings will be ignored if disabled by the administrator. If no settings are given, site-wide default values apply. For example, the ontoworld.org wiki always exports RDF with backlinks.

The default Content-Type of ExportRDF's output is application/xml (with charset=UTF-8). Content-Type of application/rdf+xml can be set by adding the "xmlmime=rdf" GET parameter; some processing tools require this RDF mimetype to process the output.

6.1.2 Exporting all data

In addition to the wiki's Special:ExportRDF function, there is also a maintenance script that allows you to export all of the wiki's semantic data at once. The script is called SMW_dumpRDF.php and can be found in SMW's maintenance directory. This directory also contains a README file that describes how to install maintenance scripts in your local MediaWiki installation.

The script SMW_dumpRDF.php can generate full exports, or it can be restricted to certain elements of the schema, e.g. to export only the category hierarchy or only the attributes with their types. Details are described in the script itself.

The script can easily be run automatically as a cronjob to generate RDF dumps on a regular basis. For ontoworld.org, the generated dumps can be obtained from http://ontoworld.org/RDF/.

6.1.3 The exported data in detail

6.1.3.1 Mapping wiki-pages to ontology elements

The export distinguishes the page in the wiki and the "thing" that the page discusses. ...

It is possible to import an external vocabulary (such as foaf or Dublin Core) into Semantic MediaWiki and associate attributes and relations in SMW with the external vocabulary, so that in RDF export the SMW attributes and relations export as properties in the external vocabulary (such as *foaf:knows* or *skos:concept*).

6.1.3.2 Categories

MediaWiki category relations are exported using existing RDF/RDFS properties. In brief:

- A category assignment in a regular article is exported as rdf:type which states "is an instance of a class". So use of MediaWiki categories is a good match for "is a" in the sense of "*San Diego* is an instance of the class Cities".
- A category assignment in a Category article is exported as rdfs:subClassOf which states "all the instances of one class are instances of another". So use of MediaWiki categories within categories is a good match for "is a" in the sense of "all instances of *Divided cities* are Cities".

There are **many** usages of MediaWiki categories that conflict with these semantics. For example, the article *Urban decay* might be in category Cities, but it is not a city. And *Category:City museums* might be in category Cities, but city museums are not cities.

6.2 External tools

A number of Semantic Web tools can immediately be used with the RDF export of Semantic MediaWiki. This page lists various tools that have been tested with SMW and explains what features they offer.

- The **Geocoding Tools for Python** provide a simple toolkit for obtaining the geographic coordinates of places in the real world. For example, you can find out the location of some building you are at by providing the buildings name. The name is looked up in various sources, including Semantic MediaWikis, and the location is returned. Further located in statements are used to infer the location of sublocations.

- **Tabulator** is an online browser for RDF documents available at http://www.w3.org/2005/10/ajaw/tab.html. Using it on external resources such as ontoworld.org requires a change in security settings of the browser, since JavaScript must be enabled to access external sites. After this is done, RDF of ontoworld can be loaded conveniently by loading the URL of some RDF export, e.g. http://wiki.ontoworld.org/index.php/Special:ExportRDF/ESWC2006. The interface is still somewhat crude, but the tool shows that dynamic interfaces for browsing RDF and RDF-enabled sites are feasible. Tabulator takes advantage of Semantic MediaWiki's feature of creating *browsable RDF* which includes "backlinks" in RDF triples as well.

- **Longwell** is a Java-based facetted browser for RDF. It includes features for browsing, querying and query refinement, Timeline support and graphical presentation of RDF graphs. It works very well with the RDF generated by Semantic MediaWiki and its faceted browsing capabilities provide some real added value for the semantic data. It is available online via http://simile.mit.edu/longwell/.

- **FOAF Explorer** is an online viewer for RDF files that use the ow:FOAF vocabulary. It can be accessed via http://xml.mfd-consult.dk/foaf/explorer/. Like most FOAF tools, FOAF Explorer is still somewhat limited and does not deal well with arbitrary

RDF that just contains some FOAF statements. It seems to work well on some pages, like http://wiki.ontoworld.org/index.php/Special:ExportRDF/User:Markus_Krötzsch.

- **Piggy Bank** is a browser extension for Firefox available at http://simile.mit.edu/piggy-bank/. Semantic MediaWiki refers to a page's RDF specification within its HTML header so that Piggy Bank can find it. In effect, Piggy Bank will collect RDF of all visited articles during browsing. Of course, the further use of this RDF, as in all applications of Piggy Bank, is left to the user.

- **KAON2** is a reasoner and ontology management API for OWL DL and SWRL. It can be used to check the validity of the exported OWL specifications. In conjunction with OWL tools, you can also transform the exported RDF into LaTeX sources that show the according DL statements (which tend to be more readable for humans, or at least for DL-aware humans).

- **OpenLink RDF Browser** is a Browser independent and Javascript based RDF Browser that is part of the OpenLink Ajax Toolkit (OAT). The deliverables includes a live demonstration that works with Semantic MediaWiki RDF Export pages.

6.3 Import vocabulary

In Semantic MediaWiki it is possible to import and reuse vocabulary that belongs to existing Semantic Web documents or standards by associating the vocabulary's elements with wiki terms. An example of this functionality is the use of the FOAF vocabulary in the ontoworld.org wiki. Although the associated terms work like any other annotation in the wiki, the RDF that is exported will directly use the elements of the FOAF specification, thus allowing users to edit FOAF files through the wiki.

6.3.1 Example: importing *foaf:knows*

Normally, concepts that can be described (and be used for description) in the wiki are *defined* by the wiki and thus are local. So the URI that RDF export uses to denote a concept is usually from a specific namespace that should be used only by the wiki. Even if you create a property that seems to come from a well-known vocabulary because you name it *foaf:knows*, it would still be exported with the URI

```
http://''your.site''/wiki/Special:URIResolver/Property-3AFoaf-3Aknows
```

which is the XML-compatible URI derived from the article URL (see Help:RDF export for details). Semantic MediaWiki's mechanism for importing vocabularies solves this problem by allowing the reuse of external vocabulary in a controlled way. After an administrator makes the external vocabulary available, any user can state that a term in the wiki matches an imported element by adding the following text to its article:

```
[[imported from::foaf:knows]]
```

The special property imported from tells SMW that the wiki's property really refers to the property *http://xmlns.com/foaf/0.1/knows* from the FOAF vocabulary.

The wiki article doesn't have to be named "Property:foaf:knows", the user could choose any name to represent the property within the wiki. (ontoworld.org uses "foaf:knows" here since the community in this wiki is expected to have some technical background; In another wiki, one might prefer a less technical name.)

In order to interpret the above "imported from" statement, the wiki needs to understand the meaning of "foaf:knows" as a shorthand for "http://xmlns.com/foaf/0.1/knows". This is the case for "foaf:knows" since the wiki administrator has made this property available for this purpose. The section on importing further vocabularies describes how you can find out which elements are available, how new elements can be added, and what the idea behind this "control mechanism" is.

6.3.2 Importing properties

Basically, all known elements can be imported as described above. When importing properties, there is a minor additional effect: the imported element will also define the datatype of the property. In this example, the property foaf:name is declared as a string merely through its import statement. **If you import a property, remove its "has type" statement.**

6.3.3 Importing further vocabularies

As explained above, not every element of some external vocabulary can be imported. The easiest way for finding out whether something is available for import is to try importing it with a statement as the above. If this does not work, a message in the factbox will inform you about the problem. Possible reasons are:

1. The namespace (e.g. "foaf:") is not available at all. The wiki does not know what it refers to, and it does not support any elements from this range.
2. The specific element (e.g. "foaf:knows") is not available in the (known) namespace "foaf:".
3. The element is known, but it cannot be imported for the specific kind of page you are trying to use it on. For example, you cannot import *foaf:knows* on a page in the wiki-namespace "Category:" since it should always be a "Property:".

In the last case, you can fix the problem by using an appropriate article for import. In the other cases, you only users with administrator privileges can add the unknown vocabulary elements. This can be done very quickly by modifying the wiki (see below), but there are also cases in which some vocabulary is not made available on purpose. Possible reasons for this are:

- The element you wish to use is not recommended for public use yet, or its correct use is still discussed in the community. An example of this is *foaf:OnlineEcommerceAccount*.
- The specification of the chosen vocabulary is too ambiguous or preliminary for current use. It might become more precise later which would affect the usage in the wiki (or render its earlier usage incorrect).
- The vocabulary is not widespread and standard, and the community of the wiki (represented by its admins) does not endorse its use at the moment. Not everyone's home-made ontology needs to be imported in a wiki.
- The specification of the vocabulary is tailored towards another ontology language than that of the wiki (OWL DL), and it is not clear how to map it correctly. Since importing some vocabulary also entails a mapping to OWL DL it must be specified how the mapping should be done. It is also questionable whether exporting such elements to OWL/RDF would be of much use.
- The imported element is part of the ontology vocabularies used for export.
    - In this case, using the element also directly could impair the sanity of the export. E.g. if *owl:Nothing* would be imported as a category into the wiki, one could easily make the whole output inconsistent by adding some article to this class, which is not desirable.
    - Alternatively, the element could already be sufficiently represented implicitly in the wiki. E.g. in order to state that something is an element of a class, one does not need to import the property *rdf:type* — it suffices to use the available category mechanism.

A quick way to find out what elements are available for some namespace abbreviation is to go to the article MediaWiki:smw_import_*foaf*, with *foaf* replaced by the namespace identifier you are interested in. The rationale behind this system is that the community can decide on a sane use of desired vocabularies, but each user still has the power to decide in which cases the available elements are used and by what name they should be represented within the wiki.

6.3.4 Making vocabularies available for import

Wiki-users with administrator status can make new elements available by simply editing a specific page for each vocabulary with a "magic" name. The page is in the Mediawiki namespace with the prefix smw_import_. For example, the page for the foaf vocabulary is named MediaWiki:smw_import_foaf. It contains something like

```
http://xmlns.com/foaf/0.1/|[http://www.foaf-project.org/ Friend Of A
Friend]

 name|Type:String

 homepage|Type:URI

 Person|Category

 knows|Type:Page
```

```
...
```

The first line tells the wiki that

- the abbreviation "foaf:" refers to "http://xmlns.com/foaf/0.1/" and that
- "Friend Of A Friend" should be given as an additional human-readable label for elements of this vocabulary (e.g. in the factbox).

After this, there is a line that declares each vocabulary element that can be reused within the wiki. For instance, "name" (referring to "foaf:name") can only be a property of Type:String. The text after the "|" declares the (unique) context in which some element can be used. Elements that can be imported as properties are declared by specifying their type with Type:*some datatype*, elements that can be imported as categories are declared by specifying the "Category" namespace identifier. (Note that in SMW 1.0 the type and namespace depends on your language setting!). Moreover, one can also declare other elements by writing anything else than the above; however, we strongly recommend you use one meaningful string; we suggest "Main", although you can use such elements in namespace other than "Property" and "Category" as well.

6.3.5 Changing import statements

It can easily happen that some existing article of the wiki should be modified to represent (another) imported concept. For example, a wiki community that already uses a category "Person" might decide to map this category to "foaf:Person" in the future. Luckily, this is not a problem. **Import statements in existing articles can be changed at any point in time without requiring additional updates in the wiki.** The exported RDF will immediately be modified accordingly. Of course, in the case of properties, the datatype should usually not be changed without updating also the articles that use this property. This is mildly related to importing since an "imported from" declaration for properties will also define the datatype.

6.4 Ontology import

Semantic MediaWiki has a feature to import ontologies into a Semantic MediaWiki installation. The ontologies have to follow certain formats, in order to be useful. Further down you will find a description of an alternative way of importing data from ontologies (or, actually, other formats as well).

6.4.1 Ontology format

The ontology elements – i.e. classes, properties and individuals -- should all have labels. The labels will be used to name the relations, the categories, and the article pages, and also to create the appropriate annotations -- i.e. typed links or categorization links -- on the article pages. The mapping of the import naturally follows the mapping of the export, so it looks like this:

OWL Construct	Semantic MediaWiki
Class	Category
Datatype property	Property
Object property	Property also *(??)*
Class instantiation	Page categorization (e.g. [[Category:X]])
Subclass of	Category subcategorization (e.g. [[Category:X]] on a category page)
Individual	Article (in Main namespace)
Instantiated datatype property	Attribute annotation (e.g. [[X:=Y]])
Instantiated object property	Typed link (e.g. [[X::Y]])

Note that the ontology needs to be in OWL DL in RDF-serialization, not just general RDF or RDFS, and that all properties and classes have to be defined as such in order to be recognized. Only explicit statements get imported, i.e. no reasoning occurs. So even if you can infer from the ontology that Adam is a man, he will not be put into the Category:Man unless such a triple is in the ontology, and Man is defined as an OWLClass. If you want to import implicit statements, you have to make them explicit first. You can use any reasoner that allows you to materialize such statements.

Note also that all further constructs from OWL DL, like inverse relations, complement classes, class union, etc., will not be imported into the ontology. If you want to use more complex ontologies with the Semantic MediaWiki, check out the publication Reusing Ontological Background Knowledge in Semantic Wikis.

6.4.2 How it works

On the page Special:Import ontology you can upload the ontology file. The special page is only available to users with admin privileges. After you have chosen the ontology file the system parses it (using RAP, thanks!), you will be presented with a list of all importable statements, i.e. especially statements that are not within the wiki already (though this display is a bit buggy, sorry for that). Here you can choose every statement to import, and you can enter a small text to be imported alongside the import (for example a template that will resolve to a message telling the user that this information was imported from a particular ontology).

After you have chosen the appropriate statements and set all other options, click on the import button at the far end of the page, and wait. A few moments later, the statements should have been imported (check Recent changes).

Note that this part is still somewhat buggy. You may want to try smaller portions of the ontology first, or even single statements, to see if it works as you want it to.

6.4.3 Alternative import

As an alternative to this experimental feature, you can use pre-processing tools to annotate wiki pages with the wiki text for SMW properties, then import those pages using MediaWiki import tool(s).

A more robust way to import ontologies, is to use a framework like the Python Wikipedia Bot. It should work with other wikis as well, not just with Wikipedia, but you will have to create a new family file in order to get access to your wiki. In this case you are not constrained to using OWL DL compliant ontologies.

For example, on the Ontoworld wiki we imported the delegates list from the ESWC2006 ontology. We sketch the program in the following. It uses the rdflib library for the RDF parsing, and it uses the Wikipedia Bot framework to work with Wikipedia. It creates a template out of the RDF. It could also create sentences with typed links inside (see towards the end of the code for an example), or check the output of the page first if the triple to be added is already included (and thus may be skipped).

```
from rdflib import Graph, URIRef, Literal, Namespace, RDF

import wikipedia, login, category



family = "ontoworld" # note that you need to setup the appropriate family
file



i = Graph()



i.bind("foaf", "http://xmlns.com/foaf/0.1/")

RDF = Namespace("http://www.w3.org/1999/02/22-rdf-syntax-ns#")

RDFS = Namespace("http://www.w3.org/2000/01/rdf-schema#")
```

```
FOAF = Namespace("http://xmlns.com/foaf/0.1/")

i.load("eswc.rdf")



ow = wikipedia.Site('en')

login.LoginManager('password', False, ow)



unchanged = list()   # in order to safe those that already have a page



# iterates through everything that has the type Person

# (note, only explicit assertions -- rdflib does not do reasoning here!)

for s in i.subjects(RDF["type"], FOAF["Person"]):

        for n in i.objects(s, FOAF["name"]):   # reads the name

            p = wikipedia.Page(ow,n)           # gets the page with that
name

            if p.exists():

                unchanged.append(n)

            else: # create the instantiated template

                h = '{{Person|'   '\n'

                h  = ' Name='   n



                for hp in i.objects(s, FOAF["workplaceHomepage"]):

                    h  = '|'   '\n'

                    hp = hp[7:]

                    h  = ' Homepage='   hp

                    if len(hp)>23: # if the name of the page is too long,
                        h  = '|'   '\n'

                        if hp.find("/"): # make something shorter

                            hp = hp[0:hp.find("/")]

                        h  = ' Homepage label= at '   hp


                for hp in i.objects(s, RDFS["seeAlso"]):

                    h  = '|'   '\n'

                    h  = ' FOAF='   hp

                h  = '\n'   '}}' # end Person template


                # write a sentence

                h  = '\n'   "''''"   n   "''' attended the [[delegate
at::ESWC2006]]."


                # add a category

                h  = '\n'   '\n'   '[[Category:Person]]'

                print n   ' changed'

                p.put(h, 'Added from ontology')


wikipedia.stopme()

print unchanged
```

As you have the full power of Python available, you can basically parse any machine-readable document and process it any way you like. As of 2006, and as long as the ontology import is still not perfect, this is the recommended way to import data into the ontology (especially since it allows you much more freedom in stating the facts and reusing templates than the ontology import ever will).

7 GFDL licence v1.2

November 2002 Copyright (C) 2000,2001,2002 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others. This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software. We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law. A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language. A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them. The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none. The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words. A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque". Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only. The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text. A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition. The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3. You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects. If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages. If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public. It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version: A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version. N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section. O. Preserve any Warranty Disclaimers. If the Modified Version includes new frontmatter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles. You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard. You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but previous publisher that added the old one. The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers. The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work. In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects. You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document. If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail. If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/. Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page: Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License". If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this: with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST. If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation. If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.