

Entry Service Tutorial

Michael Milonov¹
© Fusionsoft

Entry Service is a product that provides centralized access to distributed data. Data are represented as united hierarchy of objects with arbitrary structures. Each object has stable and unique string identifier in Entry Service. The [XQuery/XPath](#)-like language can be used over this object's hierarchy. Entry Service provides common access to objects managed by different data storages (XML files, RBMS and etc.) in the scope of local or global networks.

Entry Service has treelike structure describing data and metadata, including entities, attributes, data types and relationship between them from the point of view of a certain application domain. Entry Service is closely related to [Ontology](#) concept model as conceptualization specification for a knowledge domain, but has essential differences.

¹ Please, report all discovered errors and inaccuracies, suggestions and requests to info@fusionsoft-online.com

CONTENTS

Terminology	2
Deployment of local version of Entry Service	2
Problem to solve	3
The Example schema.....	3
Types description.....	3
Resources description.....	5
User entries description	7
Creating project using Eclipse.....	9
System log in	10
Access to objects from Example schema	11
Creating user's objects	11
Editing/removing user's objects	12
Saving to XML.....	14
Backward navigation.....	14
Temporary Entries	15
Copying the branch of Entries.....	15
Entry Editor. The utility to review and administer Entry Service.....	16
The start and look at a glance	16
Navigating Entries tree	17
Creating instances of constraints using Entry Editor.....	19
Creation and removing user's Entries using Entry Editor.....	20
Editing values, types and names of Entries	21
Saving changes	22

Terminology

Entry represents the base structural unit that allows storing data in Entry Service. Each entry has a unique string name. If you know the name you can get access to the entry you need any time. Entry Service has treelike structure and all entry names starting from root entry witch has the name “/”.

Each entry can store only one *value*.

Every entry is of some *type* and stores data corresponding to some data type. One could register entry instances in their types. It allows using backward navigation from types to instances.

Entry can refer to another entry. Such entry is called a pointer or reference. One could register pointer in the pointed entry instance. It allows using backward navigation from instance to pointers.

Deployment of local version of Entry Service

Entry Service requires jdk-1_5_0_06 or higher and the number of third-party libraries:

asm-all-2.2.3.jar

cglib-2.2_beta1.jar

jta.jar

Libraries are distributed with Entry Service.

The Entry Service distributive has the following folder structure:

EntryService – the root project's folder, entryservice.jar is placed here.

EntryService/jars – folder with third-party libraries

EntryService/javadoc – document's folder

EntryService/tutorial – tutorial’s folder
EntryService/es_schemas – folder with example schemas
EntryService/config – folder with configuration files
EntryService/EntryEditor – folder contains utility that allows reviewing and editing Entries tree.
EntryService/examples – the folder with examples of using Entry Service.

Problem to solve

Suppose we want to make simple project management system for managing people, tasks and projects. To simplify a task we will get the ready ontology where entities, attributes, links between them exist and semantically complete description of all system’s components exists also. Using this system, we will learn principles of Entry Service work from the point of java developer view who develops functions, interfaces and business logic of project management system.

The Example schema

The folder *Entry Service/es_schemas* contains the structure of projects management system, the files *EntryService.xml* and *Example.xml*.

EntryService.xml is common for all project, it contains the description of all types, non-specific from the point of concrete task view.

Example.xml contains the description of types, entities, attributes and links between them, specific for the project management system that we use as an example.

Types description

The description of types used in project management system one can see below:

```

<within name="//Example/Types">
  <type _name="Resource"
        tag="resource"/>

  <type name="Person Resource"
        tag="person-resource"
        java-class="com.fusionsoft.entry.security.GranteeImpl">
    <base-types>
      <base-type-ref _value="//Security/Types/Account"/>
      <base-type-ref value="./Resource"/>
    </base-types>

    <variable _name="#first_name"
              type="String"
              tag="first-name"/>

    <variable name="#last name"
              type="String"
              tag="last-name"/>

    <variable name="#e-mail"
              type="String"
              tag="e-mail"/>
  </type>

  <type _name="Task"
        tag="task">
    <variable name="Task Name"
              type="String"
              is-mandatory="true"
              tag="task-name"/>

    <array name="Assignment"
           type="./Assignment"
           tag="assignments"/>

    <variable name="Start Date"
  
```

```

                type="Date"
                tag="start-date"/>
        <variable          _name="Duration"
                        type="Integer"
                        tag="duration"/>
    </nesting          type="./Task"/>
</type>
<type    name="Assignment"
        tag="assignment">
    <variable          name="Resource"
                    shortcut-type="./Resource"
                    is-mandatory="true"
                    tag="resource-ref"/>
    <variable          name="Share"
                    type="Float"
                    tag="share"/>
</type>
<type    name="Project"
        tag="project">
    <base-types>
        <base-type-ref _value="./Task"/>
    </base-types>
    <variable          name="Calendar"
                    type="./Calendar"
                    tag="calendar"/>
</type>
<type    name="Calendar"
        tag="calendar">
    <array          name="Work Days"
                    type="./Work Day"
                    tag="work-days"/>
</type>
<type    name="Work Day"
        tag="work-day">
    <variable          _name="Day"
                    type="Date"
                    tag="day"/>
    <array          name="Duration"
                    type="Integer"
                    tag="duration"/>
</type>
</within>

```

The description of the type “Resource”

```

<type    name="Resource"
        tag="resource"/>

```

“Resource” is a base type for the “Person Resource” type:

```

<type    _name="Person Resource"
        tag="person-resource"
        java-class="com.fusionsoft.entry.security.GranteeImpl">
    <base-types>
        <base-type-ref _value="//Security/Types/Account"/>
        <base-type-ref _value="./Resource"/>
    </base-types>
    <variable          name="#first name"
                    type="String"
                    tag="first-name"/>
    <variable          name="#last name"
                    type="String"
                    tag="last-name"/>
    <variable          _name="#e-mail"
                    type="String"
                    tag="e-mail"/>

```

</type>

A type contains some constructions inside. They are named constraints. According to the content of <base-types> that describes the ancestors of the type, the type “Person Resource” has two ancestors: //Security/Types/Account and //Example/Types/Resource. It means that the type “Person Resource” inherits all constraint from ancestor types and also has its own:

- java-class="com.fusionsoft.entry.security.GranteeImpl" means that this type can return the instance of the java class com.fusionsoft.entry.security.GranteeImpl, which is necessary to log into system and check privileges;
- <variable _name="#first_name" type="String" tag="first-name"/> means that instance of the type Person Resource can have a string first name. The last name and e-mail are described in the same way.

The type “Task” is necessary to describe project’s tasks. It has the number of constrains:

- "Task Name", is-mandatory="true" means that “Task Name” is obligatory constraint.
- "Assignment" – array of assignments.
- "Start Date".
- "Duration".
- <nesting type="./Task"/> - task could contain subtasks.

The type “Assignment” describes executors of project’s tasks. Type has constraints:

- "Resource" is mandatory constraint. It means that assignment should have executor obligatory. This constraint is shortcut that points to some person.
- “Share”, describes the person’s share in this assignment (for example 0,5 or 1).

The type “Project” is descendant of type “Task”. It has all constraints whose are peculiar to “Task” (see the type “Task”) and one its own constraint:

- "Calendar"

The type “Calendar” includes constraint:

- "Work Days" – array of work days.

The type “Work Day” has constraints:

- "Day"
- "Duration"

Resources description

Human resources and roles are described by system’s ontology:

```
<within _name="//Example/Resources">
  <person-resource      _name="maxx"
                        first-name="Maxx"
                        last-name="U"
                        e-mail="maxx@somewhere.ru">
    <!-- "maxx" digested -->
    <password      body="mLu/tZH8zjPuEoIXW/Z1qiQj5+sbKhJQYY0buA0leLo="
                  method="SHA-256"/>
  </person-resource>

  <person-resource      name="mike"
                        first-name="Mike"
                        last-name="M"
                        e-mail="mike@somewhere.ru">
    <!-- "mike" digested -->
    <password      body="ZLTQ9HyTziPRV+aKWHZzVig9ybY8RZ1FOODjmpzpkubk="
```

```

                                method="SHA-256"/>
</person-resource>

<person-resource      _name="yuri"
                                first-name="Yuri"
                                last-name="V"
                                e-mail="yuri@somewhere.ru">
    <!-- "yuri" digested -->
    <password      body="wwkZHCBKcSye/zihQLpTJnUFga5stNbdhz+oQz1c/3o="
                                method="SHA-256"/>
</person-resource>

<person-resource      name="ovch"
                                first-name="Vladimir"
                                last-name="O"
                                e-mail="ovch@somewhere.ru">
    <!-- "ovch" digested -->
    <password      body="dnZl3rLgiTl0GS+1XlIZzjssK6Vvrk+KQcZ2VRJ5Qis="
                                method="SHA-256"/>
</person-resource>

<role      name="All Resources">
  <grantees>
    <grantee-ref      value="./maxx"/>
    <grantee-ref      value="./mike"/>
    <grantee-ref      _value="./yuri"/>
    <grantee-ref      _value="./ovch"/>
  </grantees>
</role>

<role      name="Administrators"
            _owner="//Example/Resources/ovch">
  <grantees>
    <grantee-ref      value="./ovch"/>
  </grantees>
</role>
</within>

```

Let's see the one of the resources:

```

<person-resource      name="maxx"
                                first-name="Maxx"
                                last-name="U"
                                e-mail="maxx@somewhere.ru">
    <!-- "maxx" digested -->
    <password      body="mLu/tZH8zjPuEoIXW/ZlqiQj5+sbKhJQYY0buA01eLo="
                                method="SHA-256"/>
</person-resource>

```

each person has first name, last name, e-mail and password that is necessary to log in. passwords are stored in XML files as hash.

Project management system has two roles "All Resources" and "Administrators":

```

<role      name="All Resources">
  <grantees>
    <grantee-ref      _value="./maxx"/>
    <grantee-ref      _value="./mike"/>
    <grantee-ref      value="./yuri"/>
    <grantee-ref      value="./ovch"/>
  </grantees>
</role>

<role      _name="Administrators"
            owner="//Example/Resources/ovch">
  <grantees>
    <grantee-ref      value="./ovch"/>
  </grantees>
</role>

```

Putting the finishing touches to project management system, branches "Types" and "Resources" have permissions:

```

<within      name="//Example">
  <entry      name="Types"
            owner="//Example/Resources/Administrators">
    <_permissions>
      <permission      grantee-ref="//Example/Resources/All Resources"
                        actions="select"

```

```

                                is-propagated="true"/>
        </ permissions>
    </entry>

    <entry name="Resources"
        owner="//Example/Resources/Administrators">
        < permissions>
            <permission grantee-ref="//Example/Resources/All Resources"
                actions="select"
                is-propagated="true"/>
        </ permissions>
    </entry>
</within>

```

User entries description

When all types had been described, one could make the ontology of project management system:

```

<within _name="//Example/Projects">
    <project _name="Semantic Browser"
        owner="//Example/Resources/ovch">
        <task name="1"
            task-name="GUI">
            <assignments>
                <assignment resource-ref="//Example/Resources/maxx"
                    share="1"/>
            </assignments>

            <task name="1"
                task-name="Loading a form from Entry Service and
                    showing it"/>

            <task name="2"
                task-name="Loading a master-detail form from Entry
                    Service and showing it"/>

            <task _name="3"
                task-name="Looking for and loading an abitrary form
                    from Entry Service and showing it"/>

            < permissions>
                <permission grantee-ref="//Example/Resources/maxx"
                    actions="all"
                    is-propagated="true"/>
            </ permissions>
        </task>

        <task _name="2"
            task-name="Entry service">
            <assignments>
                <assignment resource-ref="//Example/Resources/mike"
                    share="1"/>
            </assignments>

            <task _name="1"
                task-name="Loading Entry Service from XML"/>

            <task name="2"
                task-name="Editing Entry Service over XML"/>

            <task name="3"
                task-name="Managing temporary entries in Entry
                    Service"/>

            <_permissions>
                <permission grantee-ref="//Example/Resources/mike"
                    actions="all"
                    is-propagated="true"/>
            </_permissions>
        </task>

        <task name="3"
            task-name="Data sources">
            <assignments>
                <assignment resource-ref="//Example/Resources/yuri"
                    share="1"/>
            </assignments>

```

```

        <task    name="1"
                task-name="A relational table flat data source"/>

        <task    name="2"
                task-name="Filtration of a flat data source"/>

        < permissions>
            <permission    grantee-ref="//Example/Resources/yuri"
                            actions="all"
                            is-propagated="true"/>
        </ permissions>
    </task>

    <_permissions>
        <permission    grantee-ref="//Example/Resources/All Resources"
                            actions="select"
                            is-propagated="true"/>
    </_permissions>
</project>
</within>

```

As you can see from ontology description:

```

<project    name="Semantic Browser"
            owner="//Example/Resources/ovch">

```

Project has name “Semantic Browser” and the owner of the one is resource //Example/Resources/ovch.

In the scope of “Semantic Browser” three tasks exists:

- GUI
- Entry Service
- Data Sources

From the description of “Graphical user interface”:

```

<task    _name="1"
            task-name="GUI">
    <assignments>
        <assignment    resource-ref="//Example/Resources/maxx"
                        share="1"/>
    </assignments>

    <task    _name="1"
            task-name="Loading a form from Entry Service and
                        showing it"/>

    <task    _name="2"
            task-name="Loading a master-detail form from Entry
                        Service and showing it"/>

    <task    name="3"
            task-name="Looking for and loading an abitrary form
                        from Entry Service and showing it"/>

    < permissions>
        <permission    grantee-ref="//Example/Resources/maxx"
                            actions="all"
                            is-propagated="true"/>
    </_permissions>
</task>

```

is evident that it assigned to //Example/Resources/maxx and maxx implements this task oneself. Task includes 3 subtasks:

- Loading a form from Entry Service and showing it.
- Loading a master-detail form from Entry Service and showing it.
- Looking for and loading an arbitrary form from Entry Service and showing it.

From the <_permissions> description is evident that resource //Example/Resources/maxx has all privileges on this task.

Second task “Entry Service” has the same structure as the first one:

```

<task    _name="2"
            task-name="Entry service">

```

```

<assignments>
  <assignment    resource-ref="//Example/Resources/mike"
                share="1"/>
</assignments>

<task    name="1"
        task-name="Loading Entry Service from XML"/>

<task    _name="2"
        task-name="Editing Entry Service over XML"/>

<task    name="3"
        task-name="Managing temporary entries in Entry
                Service"/>

< permissions>
  <permission    grantee-ref="//Example/Resources/mike"
                actions="all"
                is-propagated="true"/>
</_permissions>

</task>

```

and consists of three subtasks:

- Loading Entry Service from XML
- Editing Entry Service over XML
- Managing temporary entries in Entry Service

The task “Entry Service” is assigned to resource //Example/Resources/mike that has all privileges on it.

The third task “Data source” has the structure that you can see below:

```

<task    name="3"
        task-name="Data sources">
  <assignments>
    <assignment    resource-ref="//Example/Resources/yuri"
                  share="1"/>
  </assignments>

  <task    name="1"
        task-name="A relational table flat data source"/>

  <task    _name="2"
        task-name="Filtration of a flat data source"/>

  < permissions>
    <permission    grantee-ref="//Example/Resources/yuri"
                  actions="all"
                  is-propagated="true"/>
  </ permissions>

</task>

```

It consists of two subtasks:

- A relational table flat data source
- Filtration of a flat data source

The task “Data source” is appointed to resource //Example/Resources/yuri that has all rights on it.

Creating project using Eclipse

Let’s create a new project using Eclipse, attach all necessary libraries and make some tests. Suppose our project has name “Project Management”. Copy directories es_schemas and config to the root project directory. Also is necessary to attach following libraries: entryservice.jar, asm-all-2.2.3.jar, cglib-2.2_beta1.jar, jta.jar. Then create a package “examples” and file TestProjects.java inside it. Then set Virtual Machine arguments (VM arguments) -Xms16m -Xmx128m. It is necessary for successful start of Entry Service.

System log in

There are two ways to log in:

- calling a dialog where user name and password will be asked
- using silent login, by writing user name and password in the code of program

Let's try the first way, the program's code will look like:

```
package examples;

import com.fusionsoft.entry.security.*;
import com.sun.security.auth.callback.DialogCallbackHandler;

public class TestProject {

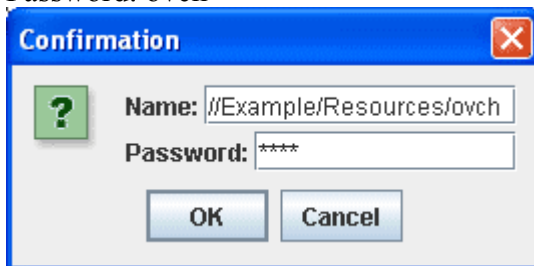
    /**
     * @param args
     */
    public static void main(String[] args) {
        try {
            if (SessionManager.login(new DialogCallbackHandler()) !=
                null) {

            }
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Type name and password when you will be asked about them, for example:

Name: //Example/Resources/ovch

Password: ovch



If authorization is passed successfully, initialization of Entry Service is started. Now Entry Service only makes initialization and then finish working because doesn't have any tasks. If you have mistaken when typing name or password, the exception will be raised and shown in console.

The second way is more convenient to debug applications and we will use it in our examples. The code will look like:

```
package examples;

import com.fusionsoft.entry.security.*;

public class TestProject {

    /**
     * @param args
     */
    public static void main(String[] args) {
        try {
            if (SessionManager.login(
                new PredefinedAccountHandler(
                    "//Example/Resources/ovch", "ovch")) != null) {

            }
        }
    }
}
```

```

    }
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

All our examples will be placed inside following block of code:

```

if (SessionManager.login(
    new PredefinedAccountHandler(
        "//Example/Resources/ovch", "ovch")) != null) {

    //Текст примера
}

```

Access to objects from Example schema

As an example, let's out to console the list of all subtasks of first task (GUI):

```

Entry task1_entry =
EntryUtils.getByUniqueName ("//Example/Projects/Semantic Browser/1");
for (Iterator<? extends Entry> subtask_iter = task1_entry.getNestedEntries(
    EntryTypeUtils.getByUniqueName ("//Example/Types/Task"));
    subtask_iter.hasNext();) {
    Entry subtask = subtask_iter.next();
    System.out.println(subtask.getNestedEntry("TaskName").
        getValue().getString());
}

```

The out should look like here:

```

Loading a form from Entry Service and showing it
Loading a master-detail form from Entry Service and showing it
Looking for and loading an arbitrary form from Entry Service and showing it

```

In our example we have made following things:

- Got the Entry of first task: `Entry task1_entry = EntryUtils.getByUniqueName ("//Example/Projects/Semantic Browser/1");`
- Made the iterator and got nested Entries of first task `Entry` those have type `//Example/Types/Task`: `subtask_iter = task1_entry.getNestedEntries(EntryTypeUtils.getByUniqueName ("//Example/Types/Task"));`
- Because the type `Task` has the name (`TaskName`), which is represented as nested entry `"TaskName"`, we show the name on the console: `System.out.println(subtask.getNestedEntry("TaskName").getValue().getString());`

Creating user's objects

Suppose the chief of the project, the person `//Example/Resources/ovch` wants to add current day (data and duration) in the Calendar. Let's suppose that it is first day of the project.

```

Entry project_entry =
    EntryUtils.getByUniqueName ("//Example/Projects/Semantic Browser");
Entry calendar = project_entry.getOrCreateNestedEntry("Calendar");
Entry work_days = calendar.getOrCreateNestedEntry("Work Days");
Entry work_day = work_days.createNestedEntry("1");
work_day.createNestedEntry("Day").getValue().setDate(new Date());
work_day.createNestedEntry("Duration").getValue().setInteger(8);

```

```

System.out.println(EntryUtils.getEntryByUniqueName (
    "//Example/Projects/Semantic Browser/Calendar/Work Days/1/Day").
    getValue().getString());
System.out.println(EntryUtils.getEntryByUniqueName (
    "//Example/Projects/Semantic Browser/Calendar/Work Days/1/Duration").
    getValue().getString());

```

First we get Entry “Semantic Browser”: //Example/Projects/Semantic Browser. Then we create new nested entry or get nested entry “Calendar” if it already exists. Then we create or get existing entry with name “Word Days”. “Work Days” is the array of Entries of “//Example/Types/Work Day” type. Then we create the element of array with name “1” and set data and duration. On can call Entries by names and we do it, showing their values on the console.

It is clear from the example that we didn’t mentioned types of entries which we have created. So what type newly created Entries have? We can find answer in ontology that describes constraints of type. The first entry that we have created was the Entry “Calendar”. The parent Entry “Semantic Browser” has the type //Example/Types/Project where constraint “Calendar” is described:

```

<variable _name="Calendar"
  type="./Calendar"
  tag="calendar"/>

```

Entry Service has two very important and convenient constraints: constraint that has type //Entry Service/Types/Variable (tag variable) and constraint that has type //Entry Service/Types/Array (tag array). They always have fixed names, which are determined by type ontology. So when we create nested Entry “Calendar” using functions getOrCreateNestedEntry or createNestedEntry it is enough to use Entry name only. Entry Service could find constraint by name in type ontology and give the right type to newly created Entry.

The same principle is correct for elements of array. Creating Entry work_day Entry Service has found that it is an element of array and setting a right type “//Example/Types/Work Day”.

Creating any other Entries those are not Variables, Arrays or elements of array one should point out exact type. In exact type is not pointed, Entry will have default type: //Entry Service/Types/Entry:

```

Entry project_entry =
    EntryUtils.getEntryByUniqueName ("//Example/Projects/Semantic Browser");

Entry e1 = project_entry.createNestedEntry("bla-bla",
EntryTypeUtils.getEntryTypeByUniqueName ("//Entry Service/Types/String"));
System.out.println(e1.getType().getEntry().getUniqueName());

Entry e2 = project_entry.createNestedEntry("bla-bla2");
System.out.println(e2.getType().getEntry().getUniqueName());

```

При этом Entry e1 будет иметь тип //Entry Service/Types/String, а e2 – тип по умолчанию, //Entry Service/Types/Entry.

Entry e1 will have type //Entry Service/Types/String and Entry e2 will have default type //Entry Service/Types/Entry.

Editing/removing user’s objects

Suppose following example: in the task 1 (GUI) we will change assignment for subtask 3 to another person and set beginning date and duration. Then we will change that assignment to another person, change the share value and delete the beginning date.

```

Entry task1_subtask3_entry =
    EntryUtils.getEntryByUniqueName ("//Example/Projects/Semantic Browser/1/3");
Entry assignment = task1_subtask3_entry.getOrCreateNestedEntry("Assignment").
    createNestedEntry("1");

```

```

assignment.createNestedEntry("Resource").setPointedEntry(EntryUtils.
    getEntryByUniqueName("//Example/Resources/yuri"));
assignment.createNestedEntry("Share").getValue().setFloat(1);
task1_subtask3_entry.getOrCreateNestedEntry("Start Date").getValue().
    setDate(new Date());
task1_subtask3_entry.getOrCreateNestedEntry("Duration").getValue().
    setInteger(100);

System.out.println(EntryUtils.getEntryByUniqueName (
    "//Example/Projects/Semantic Browser/1/3/Assignment/1/Resource").
    getPointedEntry().getUniqueName());

System.out.println(EntryUtils.getEntryByUniqueName (
    "//Example/Projects/Semantic Browser/1/3/Start Date").
    getValue().getString());
System.out.println(EntryUtils.getEntryByUniqueName (
    "//Example/Projects/Semantic Browser/1/3/Duration").
    getValue().getString());

EntryUtils.getEntryByUniqueName (
    "//Example/Projects/Semantic Browser/1/3/Assignment/1/Resource").
    setPointedEntry(EntryUtils.getEntryByUniqueName (
        "//Example/Resources/maxx"));
EntryUtils.getEntryByUniqueName (
    "//Example/Projects/Semantic Browser/1/3/Assignment/1/Share").
    getValue().setFloat(0.5);
EntryUtils.getEntryByUniqueName (
    "//Example/Projects/Semantic Browser/1/3/Duration").removeAll();

```

Please take a look at setPointedEntry and getPointedEntry functions. Entry

“//Example/Projects/Semantic Browser/1/3/Assignment/1/Resource” is a pointer because it links to another Entry. In this case one should use getPointedEntry instead of getValue().setString(). Function getPointedEntry() returns the Entry which the pointer links to. As the extension of current example, let's add the second person to subtask 3, set share value for him and then take out the assignment's list to console:

```

Entry person = task1_subtask3_entry.getOrCreateNestedEntry("Assignment").
    createNestedEntry(null);
System.out.println(person.getUniqueName());
person.createNestedEntry("Resource").setPointedEntry(EntryUtils.
    getEntryByUniqueName("//Example/Resources/mike"));
person.createNestedEntry("Share").getValue().setFloat(0.5);

for (Iterator<? extends Entry> iter =
    task1_subtask3_entry.getNestedEntry("Assignment").
    getNestedEntries(EntryTypeUtils.
    getEntryTypeByUniqueName("//Example/Types/Assignment"));
    iter.hasNext();) {
    Entry assign = iter.next();
    System.out.println("Assignment No:" + assign.getName());
    System.out.println("Resource: " +
        assign.getNestedEntry("Resource").getPointedEntry().getUniqueName());

    System.out.println("Share: " +
        assign.getNestedEntry("Share").getValue().getString());
}

```

Pay your attention to the creation of new element of array “Assignment”

createNestedEntry(null). The unnamed entry is created. But it conflicts with one of the main postulates of Entry Service that reads that each Entry has a unique name. So if a name is not shown, the Entry gets a default name, if exactly it gets a number. If an Entry with name “1” already exists, a new Entry gets name “2”. The full unique name of the Entry is //Example/Projects/Semantic

Browser/1/3/Assignment/2. It is convenient feature when creating elements of array.

Creating iterator, we get all elements of array with type

“//Example/Types/Assignment” and put their name as well as values of nested Entries

Resource and Share. Console looks like here:

```
Assignment No:1
Resource: //Example/Resources/maxx
Share: 0.5
Assignment No:2
Resource: //Example/Resources/mike
Share: 0.5
```

Saving to XML

All changes that we had made in our examples didn't save in XML files and was lost after program completion. To save changes in XML files on should execute following command:

```
EntryService.commit()
```

After that all changes will be saved in XML files.

Backward navigation

Backward navigation is necessary to go from the Entry to pointers those point to this Entry.

Suppose we had made all examples above and saved changes in XML files. In our examples we created pointers to persons when made some assignments. Such pointers were registered in the Entries of persons so we can see what Entries link to them. Let's see what Entries link to persons //Example/Resources/mike and //Example/Resources/maxx:

```
Entry person = EntryUtils.getEntryByUniqueName ("//Example/Resources/mike");
System.out.println("Person: "+person.getUniqueName()+ " has pointers:");
for(Iterator<? extends Entry> entry_iter = person.getPointers();
    entry_iter.hasNext();) {
    Entry pointer = entry_iter.next();
    System.out.println(pointer.getUniqueName());
}

person = EntryUtils.getEntryByUniqueName ("//Example/Resources/maxx");
System.out.println("Person: "+person.getUniqueName()+ " has pointers:");
for(Iterator<? extends Entry> entry_iter = person.getPointers();
    entry_iter.hasNext();) {
    Entry pointer = entry_iter.next();
    System.out.println(pointer.getUniqueName());
}
```

Console out looks like here:

```
Person: //Example/Resources/mike has pointers:
//Example/Projects/Semantic Browser/2/Assignment/1/Resource
//Example/Projects/Semantic Browser/2/#permissions/1/1
//Example/Resources/All Resources/#grantees/2
//Example/Projects/Semantic Browser/1/3/Assignment/2/Resource
Person: //Example/Resources/maxx has pointers:
//Example/Projects/Semantic Browser/1/Assignment/1/Resource
//Example/Projects/Semantic Browser/1/#permissions/1/1
//Example/Resources/All Resources/#grantees/1
```

Temporary Entries

Temporary Entries exist in addition to regular Entries. The life time of temporary Entries is appointed by work time of Entry Service. When Entry Service is finishing working all temporary Entries are destroying. Anyway such Entries could be useful to cache some calculations results, store some temporary values, counters and etc.

As an example let's create temporary Entry in the task 1 (GUI) that fixes the start date of Entry Service:

```
Entry task1_entry =
EntryUtils.getEntryByUniqueName ("//Example/Projects/Semantic Browser/1");
Entry temp_root = task1_entry.createNestedEntry ("_");
System.out.println(temp_root.getUniqueName() + " is " +
    String.valueOf(temp_root.isTemporary()) + "temporary");
temp_root.createNestedEntry("StartDate",
    EntryTypeUtils.getEntryTypeByUniqueName (
        "//Entry Service/Types/Date"));
getValue().setDate(new Date());
```

The key moment here is creation of Entry with name “_”. Entry Service makes this Entry temporary. Then all nested Entry starting from “_” are temporary Entries and they are not stored in the data storage.

Copying the branch of Entries

Entry Service provides three ways to copy the branch of Entries:

- Copying missing Entries to one branch from another with synchronization of types and values of Entries those have concurrent names. Accomplished by copyFrom function.
- The exact copy of the branch. All Entries those are missing in the source branch will be deleted. Accomplished by copyAs function.
- The creation of version of source branch with possibility to navigate from version to source branch and back.

Let's see all ways of copying. We will use the branch `//Example/Projects/Semantic Browser/1` as source and will copy it to the temporary Entry:

```
Entry task1_entry =
EntryUtils.getEntryByUniqueName ("//Example/Projects/Semantic Browser/1");
Entry temp_root =
    EntryUtils.getEntryByUniqueName ("//Example/Projects/Semantic Browser").
        createNestedEntry ("_");

Entry version = temp_root.createNestedEntry("version");
Entry copy = temp_root.createNestedEntry("copy");
Entry copy_as = temp_root.createNestedEntry("copy_as");
copy_as.createNestedEntry("additional");

copy.copyFrom(task1_entry);
version.createVersion(task1_entry, "GUI");
copy_as.setAs(task1_entry);

System.out.println(copy.getUniqueName() + " has type: " +
    copy.getType().getEntry().getUniqueName());
System.out.println(copy_as.getUniqueName() + " has type: " +
    copy_as.getType().getEntry().getUniqueName());
System.out.println(version.getNestedEntry("GUI").getUniqueName() +
    " is version of: " + version.getNestedEntry("GUI").
    getVersionedEntry().getUniqueName());
System.out.println(task1_entry.getUniqueName() + " has following versions:");
for(Iterator<? extends Entry> ver_iter = task1_entry.getVersions();
    ver_iter.hasNext();) {
    System.out.println(ver_iter.next().getUniqueName());
```

```
}
```

The console output:

```
//Example/Projects/Semantic Browser/_/copy has type: //Example/Types/Task
//Example/Projects/Semantic Browser/_/copy_as has type: //Example/Types/Task
//Example/Projects/Semantic Browser/_/version/GUI is version of:
//Example/Projects/Semantic Browser/1
//Example/Projects/Semantic Browser/1 has following versions:
//Example/Projects/Semantic Browser/_/version/GUI
```

We have made three temporary Entries: copy, copy_as and version and copied the source branch to them using different copying functions; copy and copy_as are exact copies of `//Example/Projects/Semantic Browser/1`; Entry version contains the version of source branch starting from Entry with name “GUI”. One can navigate from version to original branch and back using functions `getVersionedEntry` and `getVersions`.

Entry Editor. The utility to review and administer Entry Service

Entry Editor is a part of Entry Service distribution. It is utility with graphical interface that allows reviewing and administering the tree of Entries. Entry Editor allows:

- Review the structure of Entries;
- Create new Entries;
- Remove existing Entries;
- Change type and value of Entries;
- Rename Entries;
- Save changes to storage

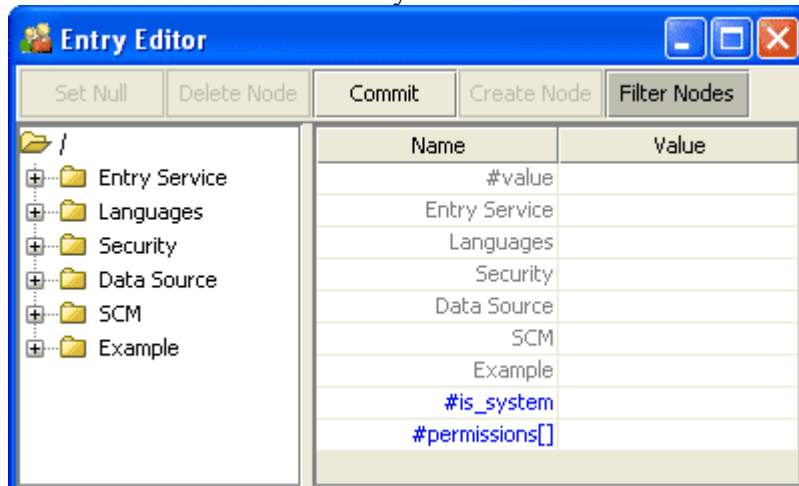
The start and look at a glance

To start Entry Editor use file `entryeditor.bat` from `EntryEditor` folder. You can customize the path to folder with examples xml schemas by editing file `config/application.properties`. By default `es_schemas` from Entry Service distributive are used.

When Entry Editor is starting one should log in. One should enter the name of person and password, for example `//Example/Resource/ovch` and password: `ovch`. One can use another person's names:

```
//Example/Resource/mike, password: mike
//Example/Resource/yuri, password: yuri
//Example/Resource/maxx, password: maxx.
```

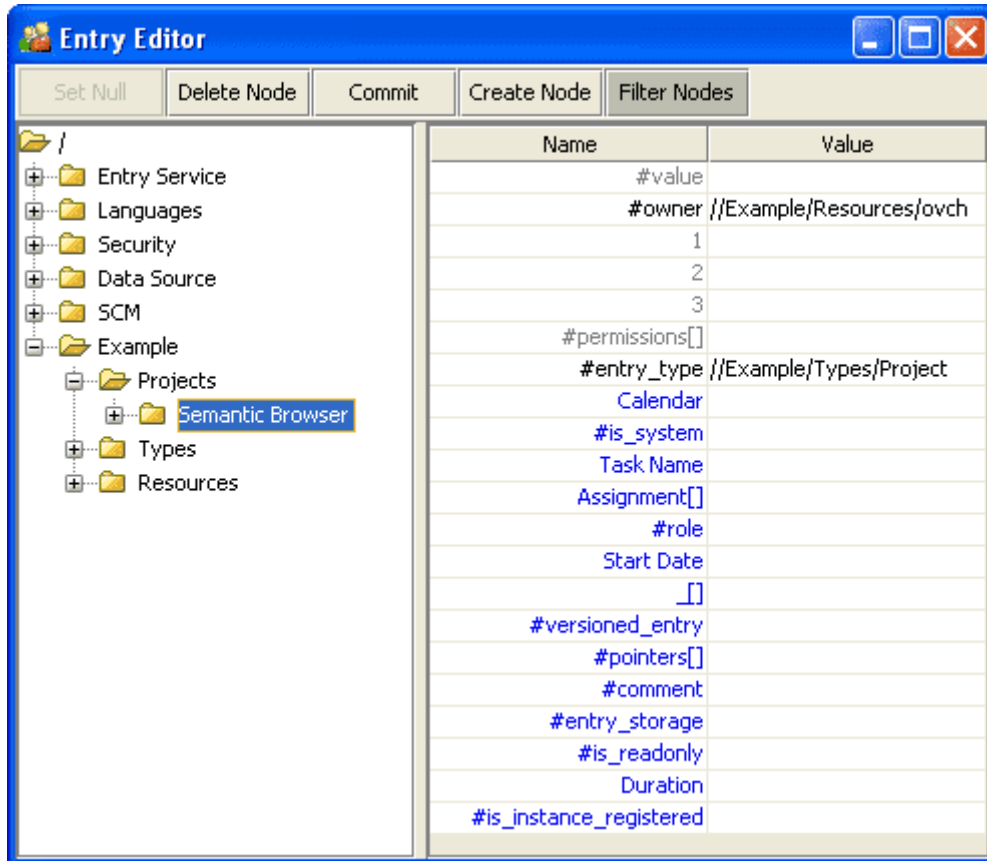
After successful initialization you can see the window of Entry Editor:



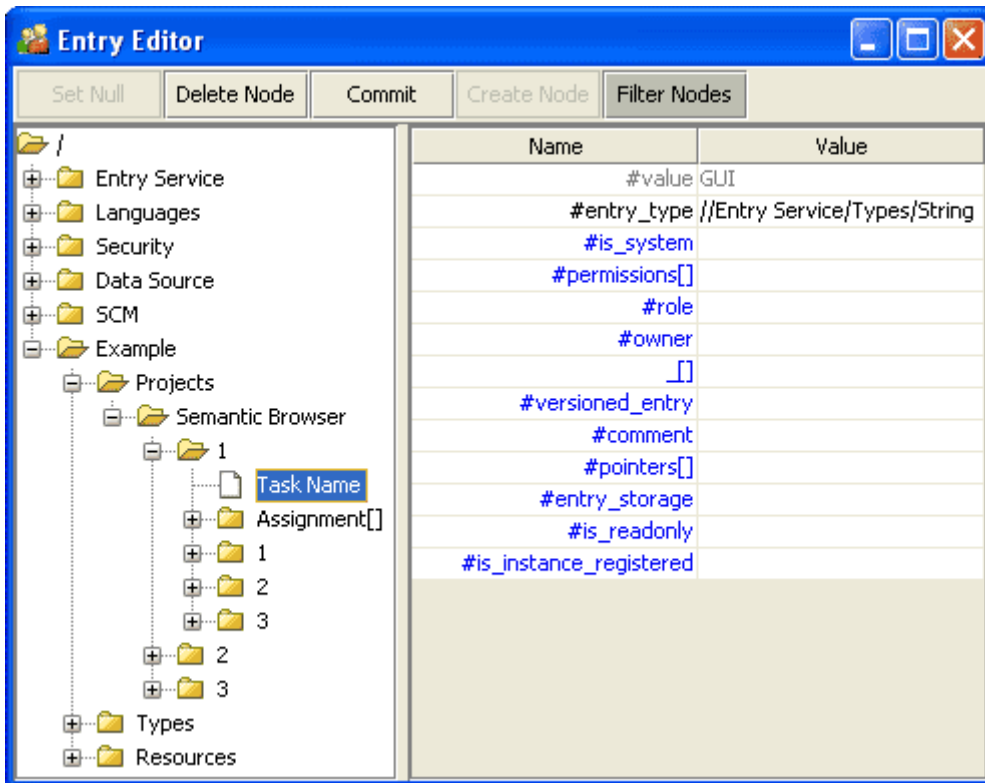
In the left part of the application on can see the structure of Entries tree. Right part shows Entries those are nested to Entry, which is selected in the left part of application as well as all constraints (usually start from #) those are peculiar to selected Entry type.

Navigating Entries tree

Let’s open the folder Example/Project/Semantic Browser and select the Entry “Semantic Browser”:

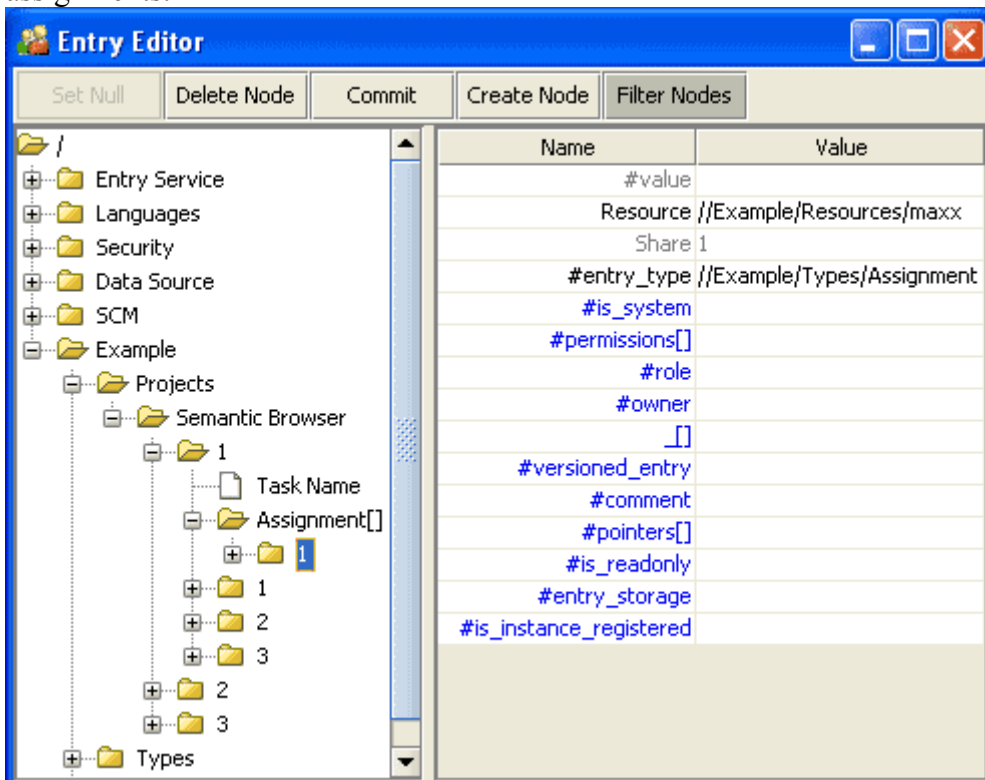


In the right part of application on could see names of familiar constraints (we saw them in example ontology): Calendar, Task Name, Duration, Start Date, Assignment. On can see nested Entries of project’s tasks: 1, 2, 3. There are some other constraints those came from ancestors of type //Example/Types/Project. Continuing navigation, let’s go to Entry //Example/Projects/Semantic Browser/1/Task Name:



In the right part on can to see the value (#value=GUI) and Entry type (//Entry Service/Types/String). Existing Entries are marked by gray and black colors. Existing pointers are marked by black color. Non-existent constrains have blue color.

Going to Assignment array (arrays are marked by sign []) on can to see the list of task's assignments:

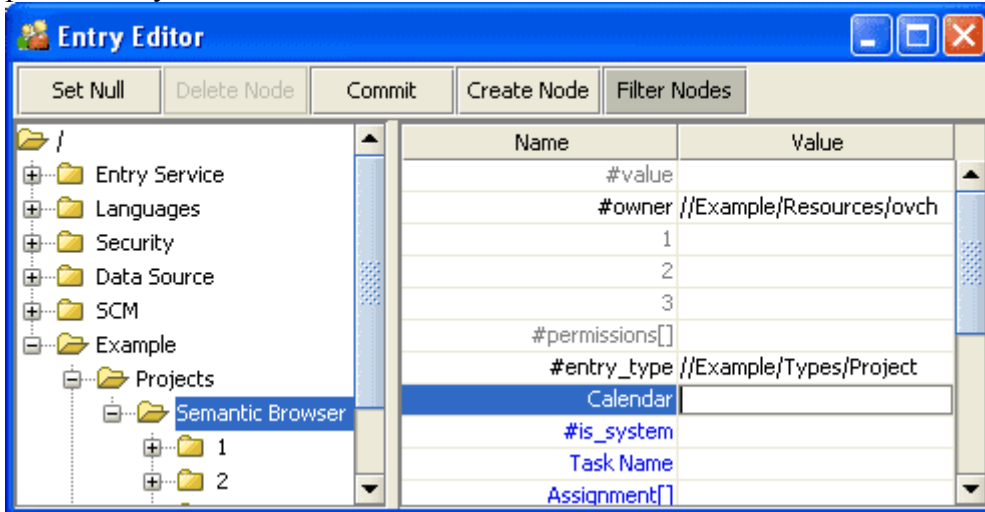


If the button "Filter Nodes" is switched on system Entries (with names started from #) are not shown in the three. On can to show all Entries by switching off "Filter Nodes" button.

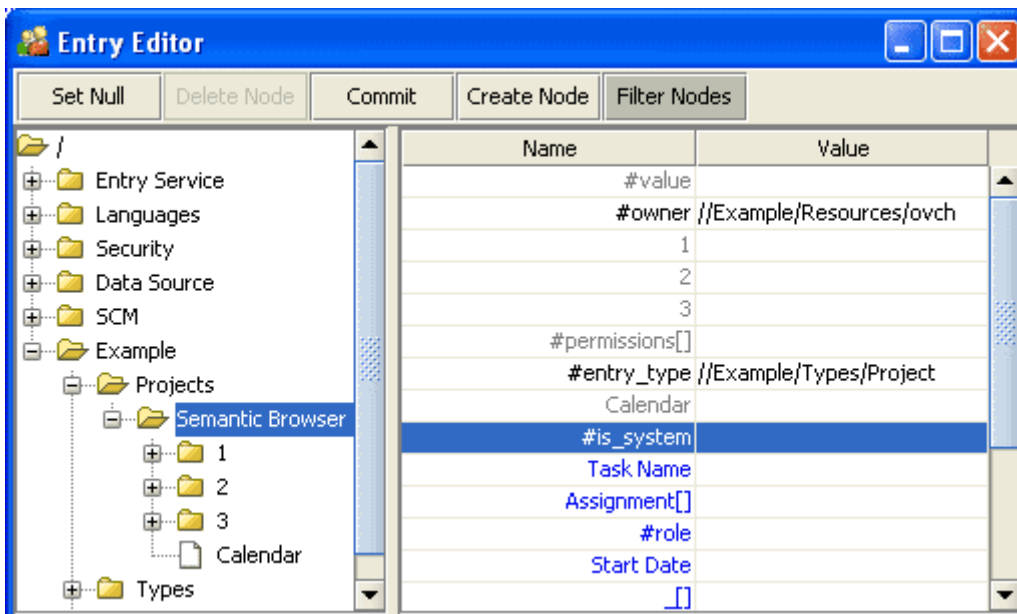
Creating instances of constraints using Entry Editor

Let's decide the task of Calendar creation, so we simply do the same task as in chapter [Creating user's objects](#). We will create a Calendar and add current day to it.

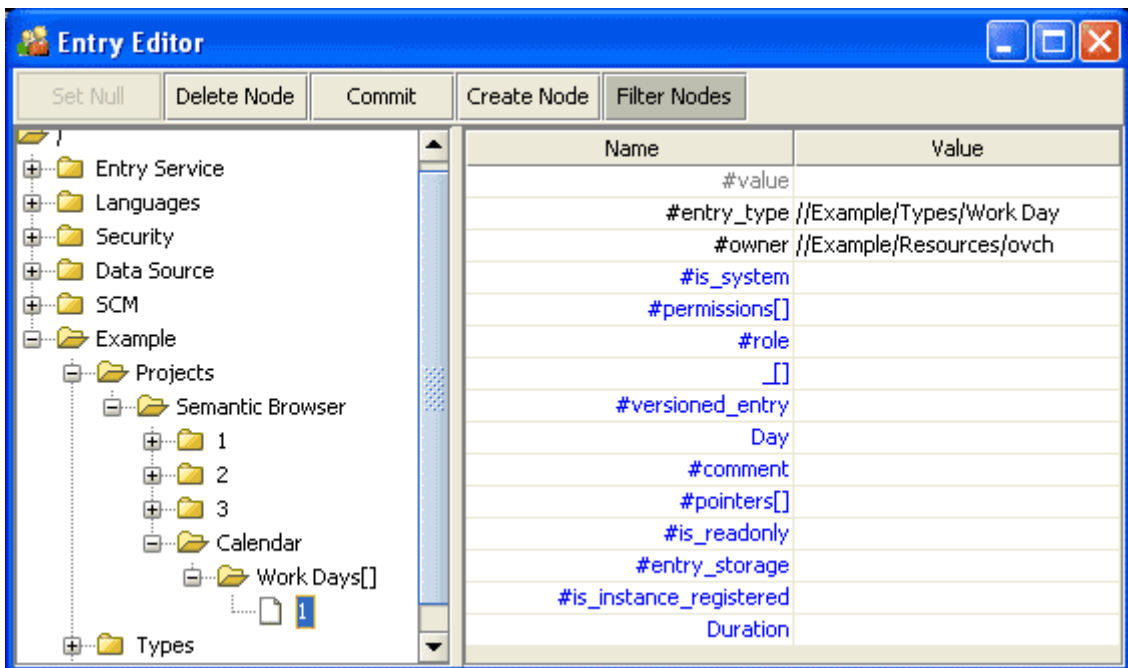
On need select Entry "Semantic Browser" in the left part and constrain "Calendar" in the right part of Entry Editor:



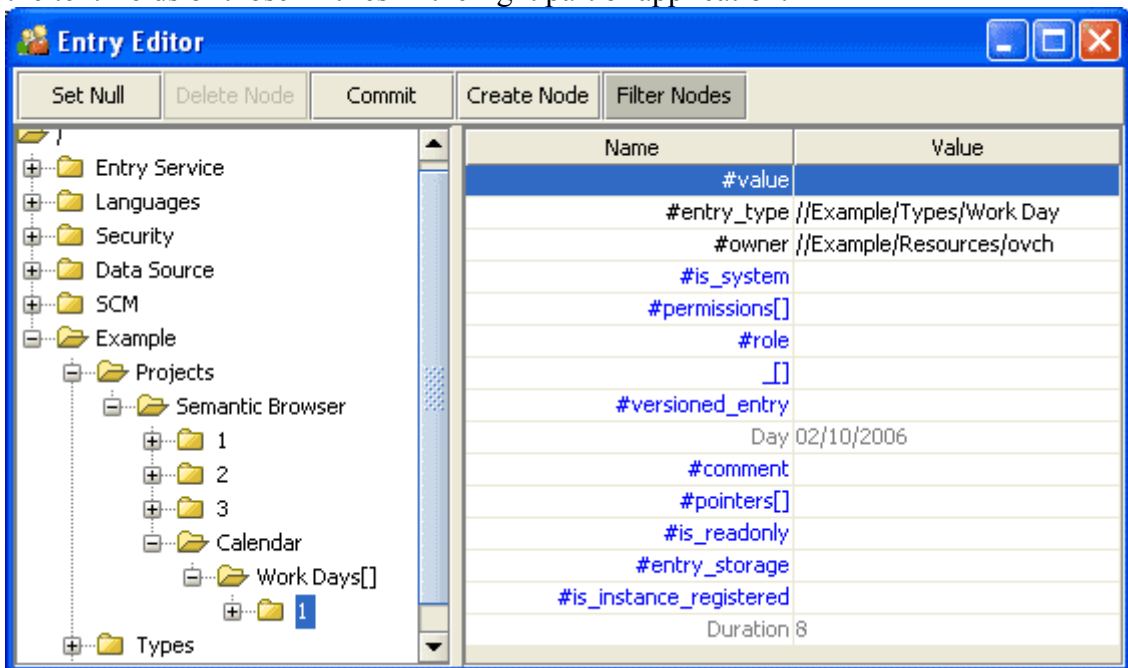
Push "Enter" key in the empty text field of Entry "Calendar" and Entry will be created and appear in the tree:



Using the same way we will create WorkDays array in Entry "Calendar". To create the element of array on need to select the newly created Entry "WorkDays" and push "Create Node" button in the Entry Editor's toolbar. The new element of array will be created with name "1".



Let's create constraints Day and Duration for Entry "1". One can do it simply by typing values in the text fields of those Entries in the right part of application:

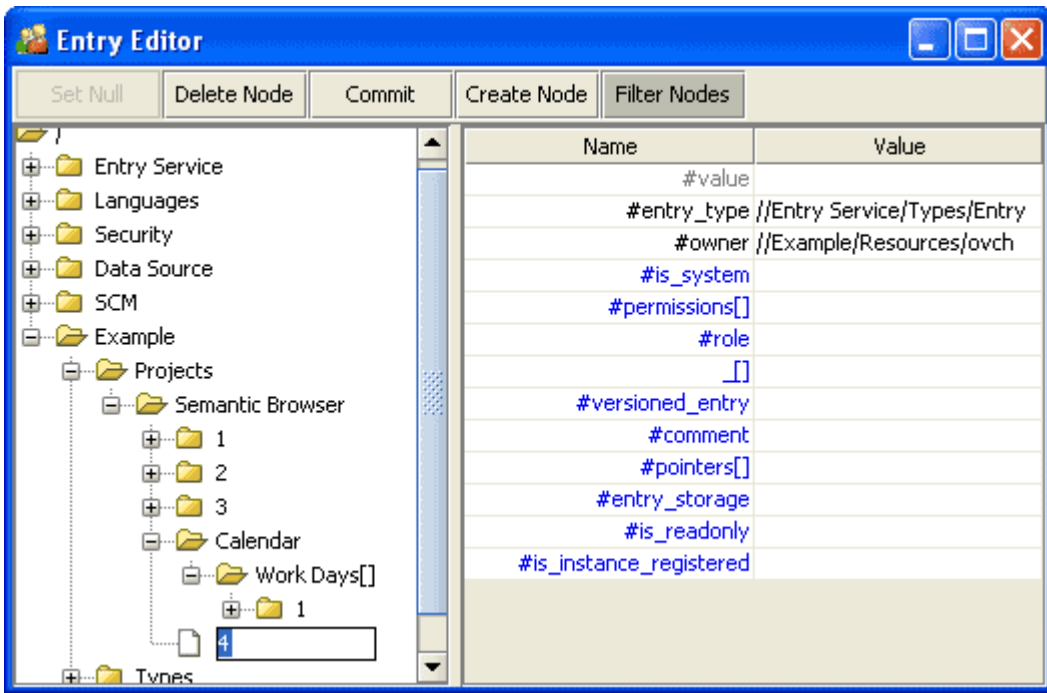


So we have completed the addition of work day to the project's calendar using Entry Editor.

Creation and removing user's Entries using Entry Editor

We have been already familiar with Entries creation using Entry Editor, but there are some moments those I want to clarify. As was shown in previous example, one could create type constraints using the right panel of Entry Editor. When we were creating element of array the button "Create Node" was used. There is some nuance here: if element of massive is created, new Entry will have type which is peculiar to current array; but all another Entries will have default type //Entry Service/Types/Entry.

Let's create new nested Entry within "//Example/Projects/Semantic Browser", using "Create Node" button:

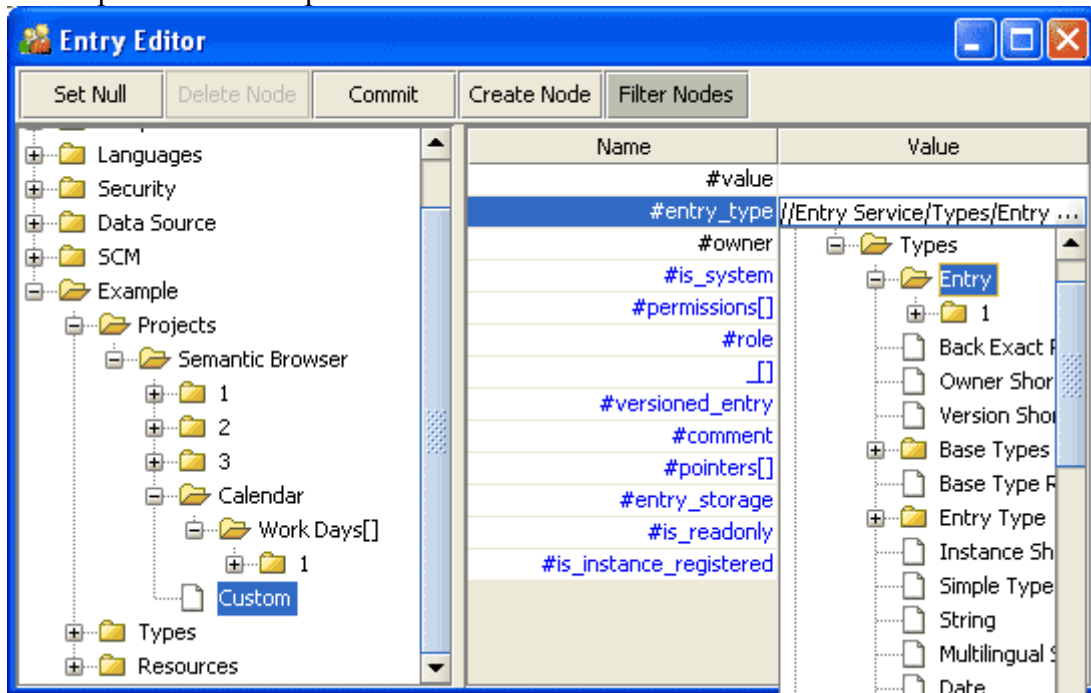


Entry is created with default name (number) and has default type //Entry Service/Types/Entry. Let's create some number of such Entries. To delete Entry use the button "Delete Node".

Editing values, types and names of Entries

Entry Editor allows editing properties of existing Entries. As an example change the name from "4" to "Custom". Do the left click on the Entry name in Entries tree. Text edit field should appear. Change the Entry name using this field.

To change Entry type go to the right panel, select #entry_type field, select appropriate type from the drop-down list and push Enter:



Value could be edited by typing some string in the field #value.

Saving changes

To save all changes on should push “Commit” button on the program toolbar. Changes will be stored in storage.