

---

# RDBToOnto Development Guide

Version 1.2 beta

## Adding New Readers and Converters

---

**Farid Cerbah (Dassault Aviation)**

farid.cerbah@dassault-aviation.fr

**Abstract.**

This guide addresses the extension capabilities offered by RDBToOnto. A framework is provided to allow the implementation and integration of new readers and converters. The description of these capabilities given in this report relies on a fully implemented sample that includes a reader and a converter. The report explains how to extend RDBToOnto configuration and user interface to handle the new components with their specific global options and local constraints.

**Keyword list:** Ontology learning, relational databases, OWL

<b>Document Id.</b>	TAO/2008/D7.2a2/v1.2
<b>Project</b>	TAO IST-2004-026460
<b>Web links</b>	<a href="http://www.tao-project.eu/researchanddevelopment/demosanddownloads/RDBToOnto.html">http://www.tao-project.eu/researchanddevelopment/demosanddownloads/RDBToOnto.html</a>

## Contents

<b>1</b>	<b>RDBToOnto extension capabilities</b>	<b>2</b>
<b>2</b>	<b>A running example</b>	<b>2</b>
<b>3</b>	<b>Configuration files</b>	<b>3</b>
<b>4</b>	<b>Implementing new database readers</b>	<b>3</b>
<b>5</b>	<b>Implementing new converters</b>	<b>5</b>
5.1	Main class of the converter . . . . .	5
5.2	Adding new global options . . . . .	6
5.2.1	Definition of global options . . . . .	6
5.2.2	Initialization of the global constraints . . . . .	6
5.2.3	Saving of global options . . . . .	8
5.2.4	Extending the user interface to handle the new global options . . . . .	9
5.3	Adding new types of local constraints . . . . .	10
5.3.1	Constructors . . . . .	10
5.3.2	Reading and saving the local constraints . . . . .	10
5.3.3	Other methods to be specialized . . . . .	11
5.3.4	Extending the user interface to handle the new local constraints . . . . .	12

## 1 RDBToOnto extension capabilities

RDBToOnto is a highly configurable tool that can be used to derive fine-tuned populated ontologies from relational databases. Further details on its functionalities are given in the user guide.

Though RDBToOnto implements a complete ontology learning process, it is worth noting that it is designed as a platform that allow the integration of new method implementations. This guide is specifically focused on the extension capabilities.

The tool can be extended by adding:

- **New database readers.** To feed the learning process with data, some database readers are included in the core system while allowing the integration of new ones to cover other types of databases (or any other source of structured data, such as XML data files).
- **New converters,** i.e. implementations of new learning/conversion methods. A number of development means are included in the platform to ease the implementation and integration of new methods together with specific options, local constraints and dedicated user interface.

Figure 2 at the end of this report provides an overview of classes and methods that may need to be overridden depending on the extension to be applied.

## 2 A running example

A fully implemented example is provided in directory `dev/samples/AroundOneTable`.

We refer to this example throughout this document. The two main extension capabilities are illustrated:

- **Adding a new database reader:** The sample includes the implementation of a reader that shows how to exploit XML exports from MS Access.
- **Adding a new converter:** The `AroundOneTable` converter included in this sample shows how to implement a new transformation method with its specific global options and local constraints.

`AroundOneTable` produces an ontology from a source table. The only tables that are considered are the source table and those which are directly linked to it through foreign key relationships. The source table is a user input defined as a global option.

The way new types of local constraints can be added is also illustrated. A new local constraint type is added to allow the inclusion of comments on the classes derived from tables.

User interface extensions are implemented for the new global option and local constraint.

### 3 Configuration files

Parameterization of the tool and definition of projects are performed through XML files, and more particularly “configuration files” as defined in *Apache commons configuration* package<sup>1</sup>. In an RDBToOnto session, the settings are issued from two configuration files:

- The RDBToOnto configuration file
- The current project file.

Some parameter values assigned in the RDBToOnto configuration can be overridden by values assigned in the current project file.

In the Java code:

- The RDBToOnto configuration is represented by the static class member `RDBToOnto._R2OConf` which is an instance of class `XMLConfiguration`.
- The current project descriptor is represented by the static class member `RDBToOnto._PrjConf`, which is also an instance of class `XMLConfiguration`.

New defined components should be declared in the RDBToOnto configuration file:

- Database readers are declared in the list of available readers (`<availableReaders>` element).
- Converters are declared in the list of available converters (`<availableConverters>` element).

For both types of components, the name of the implementing class should be given.

### 4 Implementing new database readers

Acquisition of input data in RDBToOnto is performed through the mediation of a product independent representation of relational data. Readers dedicated to specific types of databases should be implemented. The function of a reader is to abstract away from the specific encoding formats of database systems in order to provide the schema definition and table data into a “neutral” form, to be considered as the reference input for all processing tasks in RDBToOnto.

For this purpose, a database model is included in the platform. It is depicted in figure 1. The result of the reader is an instance of class `Database`. It should be noted that even though the reader is called at the start of the process, all the table data are not necessarily loaded from the start. The `Database` object can be partially filled by the reader and the data can be loaded in the course of the transformation process<sup>2</sup>.

<sup>1</sup><http://commons.apache.org/configuration/userguide-1.2/overview.html>.

<sup>2</sup>By invoking method `ensureDataLoaded(TableDef tDef)`, which creates and fills in the `table` object corresponding to the `TableDef` object (table definition from the schema) given as argument. The method has no effect if the table data is already loaded.

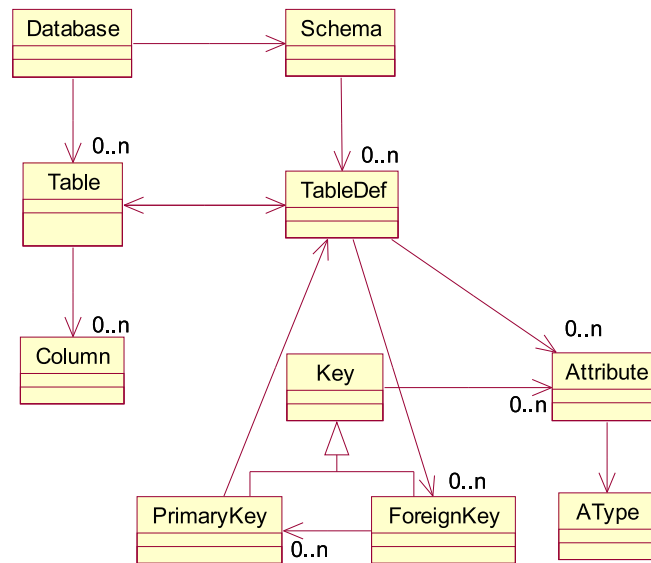


Figure 1: Simplified view of RDBToOnto database model

New readers should be defined in package `rdbtoonto.dbmodel` and derived from class `DBReader`. The `Database` instance is created by the base constructor `DBReader(String loc)` (set to `_Database` class member).

To further fill in this target object, the following methods should be overridden:

- `readSchema()`: to get the schema definition and create the resulting `DBSchema` instance (to initialize `_Database._DBSchema`).
- `readTableData(TableDef tdef)`: to get the data of a given table. The effect of the method should be the creation and filling of the corresponding `Table` instance and update of the `Database` instance.

To this end, the implementation should include a code part that might look like:

```

...
Table table = new Table(tdef.Name);
_Database.addTable(table);
tdef.setTable(table);
table.setTableDef(tdef);
...

```

The full implementation of the `XMLAccessReader` is given in `AroundOneTable` sample (java classes are in directory `dev/samples/AroundOneTable/src/rdbtoonto/dbmodel`).

This reader is intended to exploit output from data export utility given in directory `dev/samples/AroundOneTable/DBExporter` (look at document `DBExporter-README.pdf` for use instructions).

**Note**

When creating a new project, the selection of the reader is carried out on the basis of the "reader/database path name extension and prefixes associations defined in the RDBToOnto configuration file. For example, the MySQL reader is associated in the configuration file with the prefix "jdbc:mysql" while the MS Access reader is associated with "mdb" file extension.

If several readers are assigned to the same extension, the first one in the list is selected (choice of another reader is possible by changing the order of the available readers).

Readers can also be defined in the configuration file without specifying particular extensions. Selection of such readers should be done manually through the interface before invoking creation of a new project.

## 5 Implementing new converters

A converter is characterized by:

- A transitioning method
- Global options
- Local constraints
- A dedicated user interface to manage the specific global options and local constraints.

Depending on the extensions introduced in the new converter, some methods should be specialized. Figure 2 gives a view of all methods that might be specialized with short indications on when this might be needed. Detailed descriptions and examples are given in the the following sections.

### 5.1 Main class of the converter

RDBToOnto handles a typical process where some steps can be adapted through invocation of an "external" converter.

The typical process is:

1. Database optimization (optional)
2. Loading of the data needed for the identification of the ontology model (i.e. definitions of classes and properties)
3. Conversion (ontology model identification and optionally ontology population)
4. Saving of the generated ontology

Steps 2 and 3 can be adapted. This is done by overriding the methods `readNeededData()` and `convert()` from `RDBToOntoConverter` class.

The main class that implements the new converter should be defined in the package `rdbtoonto.converters` and be derived from `RDBToOntoConverter` base class.

The `readNeededData()` method loads the data needed from the start of the conversion step. Basic behavior of the `readNeededData()` implemented in `RDBToOntoConverter` class is restricted to database schema loading. However, more information might be needed by some methods. For instance, a converter might need both the database schema and table data to identify the class and property definitions. To meet this requirement, the method could be overridden in the main class of the converter implementation.

However, note that, for sake of performance, it might be more convenient to load the needed data in the course of the process. This is what is done in the RTAXON converter implementation. The class and property mining method exploits the table data along with the schema but the data are only loaded when required (using the method `ensureDataLoaded(...)`).

## 5.2 Adding new global options

### 5.2.1 Definition of global options

- **In configuration files**

Global options are defined and initialized in the conversion part of the XML configuration files.

**Example**

Adding a boolean global option `withInstances` indicating whether the generated ontology should be populated.

Within `RDBToOnto` configuration file or a project file, initialization of the option can be:

```
<conversion withInstances=' 'yes' '  
...  
</conversion>
```

(possible values for boolean options are `yes` and `no`).

- **In the Java code**

In the converter implementation, global options are defined as static class members.

**Example**

The `withInstances` boolean option would be defined as:

```
static boolean _WithInstances;
```

### 5.2.2 Initialization of the global constraints

Methods are provided in `RDBToOnto` class to manipulate the global options, especially for initialization.

Some of the highly useful methods are: `booleanInit`, `addBoolProperty`, `initProp` (see javadoc documentation for detailed description).

**Example**

```
_WithInstances = booleanInit(_R2OConf,
    ``conversion[@withInstances]``, ``no``);
```

This method call tries to initialize the option from the RDBToOnto configuration. If nothing is found in this source, the option is set to default value `no`.

When adding a new converter, some initialization methods should be overridden in the converter main class:

- `initConvConf`: Reading of options default values from the RDBToOnto configuration. The new implementation should start with a call to `RDBToOntoConverter.initConvConf()`.

**Example**

```
public class AroundOneTable extends RDBToOntoConverter {
    public static boolean _WithInstances;
    public static String _SourceTabName;

    public AroundOneTable() {
        super();
    }

    public static boolean initConvConf() {
        RDBToOntoConverter.initConvConf();
        _WithInstances = RDBToOnto.booleanInit(
            RDBToOnto._R2OConf,
            "conversion[@withInstances]",
            true);

        return true;
    }
}
```

- `initNewProject`: To reset the option values when creating a new project. The new implementation should start with a call to `RDBToOntoConverter.initNewProject()`.

**Example**

```
public class AroundOneTable extends RDBToOntoConverter {
    ...
    public static boolean initNewProject() {
        boolean res = RDBToOntoConverter.initNewProject();
        if (!res) return false;
        _SourceTabName = "-none-";
        return true;
    }
}
```

- `initConvProject`: to read option values from an existing project. The new implementation should start with a call to `RDBToOntoConverter.initCommonConvProject()`.

**Example**

```
public class AroundOneTable extends RDBToOntoConverter {
    ...

    public static boolean initConvProject() {
        boolean res = RDBToOntoConverter.initCommonConvProject();
        if (!res) return false;
        _WithInstances = RDBToOnto.booleanInit(
            RDBToOnto._PrjConf,
            "conversion[@withInstances]",
            _WithInstances);
        _SourceTabName = RDBToOnto.initProp(
            RDBToOnto._PrjConf,
            "conversion.sourceTable",
            "-none-");
        readAllLCSs();
        return true;
    }
}
```

Note that `initConvProject` also performs reading of local constraints through a call to `readAllLCSs()` (see 5.3).

**5.2.3 Saving of global options**

The method `saveConvPartOfProject()` should be overridden. The new implementation should start with a call to `saveCommonConvPart()`.

**Example**

```
public class AroundOneTable extends RDBToOntoConverter {
    ...
    public static void saveConvPartOfProject() {
        saveCommonConvPart();
        RDBToOnto._PrjConf.addProperty(
            "conversion.sourceTable", _SourceTabName);
        saveAllLCSs();
    }
}
```

Note that `saveConvPartProject` also performs saving of local constraints through a call to `saveAllLCSs()` (see 5.3).

### 5.2.4 Extending the user interface to handle the new global options

To display and manipulate the added global options through the interface, the concerned part of the interface should be adapted.

The class that implements the required extensions should bear a specific name which is the concatenation of the converter name and the suffix `Gui_GOptPanel`.

The class should be defined in package `rdbtoonto.gui` and be derived from `BaseGlobalOptionPanel` class.

#### Example

Partial definition of the global option Gui section for the `AroundOneTable` converter.

```
public class AroundOneTableGui_GOptPanel extends BaseGlobalOptionPanel
{
    private static JComboBox _TableSelector;
    private static JCheckBox _InstSwitch;

    public AroundOneTableGui_GOptPanel() {
        super();
        GridBagConstraints c = new GridBagConstraints();
        ...

        c.anchor = GridBagConstraints.WEST;
        _InstSwitch = new JCheckBox("Instances");
        _InstSwitch.setToolTipText("Populate the ontology");
        c.gridx = 0;
        add(_InstSwitch,c);
        ...
    }
}
```

The following methods should be overridden:

- `displayInitValues()`: display of the option values in the interface
- `getValuesFromGui()`: getting the values from the interface
- `setProjectEnabled(boolean)`: a method allowing to enable/disable display of project-related items.

Each implementation of the methods should start with a call to the super class method.

**Example**

Partial definition of the global option `Gui` section for the `AroundOneTable` converter.

```
public void displayInitValues() {
    super.displayInitValues();
    _InstSwitch.setSelected(AroundOneTable._WithInstances);
    _TableSelector.addItem("-none-");
    R20Gui.initWithTableNames(_TableSelector);
}
```

See the example code for further details.

**5.3 Adding new types of local constraints**

To add new types of local constraints, a specific class is required for the extended local constraint set. This class should be derived from the base class `LocalConstraintSet`.

Typically, a boolean attribute is assigned to each local constraint to be able to dynamically activate/deactivate it.

**Example**

Adding a comment to the generated classes.

```
public class AroundOneTableLCS extends LocalConstraintSet {
    public String _Comment;
    public boolean _CommAllowed;
    ...
}
```

**5.3.1 Constructors**

At least, the two following constructors should be included in this class:

- Constructor with table name as argument
- Constructor with a `LocalConstraintSet` as argument (a kind of partial copy constructor).

**5.3.2 Reading and saving the local constraints**

To take into account the new constraints in the reading and saving of constraint sets, the following methods from the converter class should be overridden:

- `readLCS()`: The new implementation should start with a call to `super.readLCS` method to create a local constraint set with the common constraints. Then, the specific

constraints are initialized.

### Example

```
public AroundOneTableLCS readLCS(String iter) {
    LocalConstraintSet cd = super.readLCS(iter);
    AroundOneTableLCS tcd = new AroundOneTableLCS(cd);
    String comm = RDBToOnto._PrjConf.getString(
        "conversion.lcs" + iter + ".comment");
    if (comm == null) {
        comm = "-none-";
    }
    tcd._Comment = comm;
    return tcd;
}
```

The argument `iter` is an iterator over the list of constraint sets. It is initialized by the calling method `readAllLCSs()` (there is no need to redefine this high-level method).

- `saveLCS()`: The new implementation should start with a call to `super.saveLCS` method to save the common constraints. Then, the specific constraints are saved.

```
public void saveLCS(LocalConstraintSet cd) {
    super.saveLCS(cd);
    AroundOneTableLCS aotcd;
    if (cd instanceof AroundOneTableLCS) {
        aotcd = (AroundOneTableLCS)cd;
    }
    else {
        aotcd = new AroundOneTableLCS(cd);
    }
    if (!aotcd._Comment.equals("-none-")) {
        RDBToOnto._PrjConf.addProperty(
            "conversion.lcs.comment", aotcd._Comment);
    }
}
```

Note that the object which is passed as argument is a `LocalConstraintSet` but not necessarily an `AroundOneTableLCS`. The constraint might have been created by another converter which has its specific local constraints.

### 5.3.3 Other methods to be specialized

Additionally, the following methods should be overridden:

- `toString()`: This method is called to display the list of the local constraint sets within `RDBToOnto` main view. The new implementation should start with the call `super.toStringOpen()`. Then, serialization of the new constraints can be added.

**Example**

```

public String toString() {
    String str = super.toStringOpen();
    if (!_Comment.equals("-none-")) {
        String comm;
        if (_Comment.length() <= 20) {
            comm = _Comment;
        }
        else {
            comm = _Comment.substring(0,20) + " ...";
        }
        str += ", <b>Comment: </b><i>" + comm + "</i>" + "]"";
    }
    return str;
}

```

(In this implementation, only the first twenty characters of the comment are considered).

- `allActive()`: To test if all the defined constraints are active. The method from super class should be called as in the following example:

```

public boolean allActive() {
    return (super.allActive() &&
            (_CommAllowed || (_Comment.equals("-none-"))));
}

```

**5.3.4 Extending the user interface to handle the new local constraints**

To display and manipulate the added local constraints through the interface, the local constraint set editor should be extended.

The class that implements the new editor should bear a specific name which is the concatenation of the converter name and the suffix `Gui_LCSEditPanel`.

The class should be defined in package `rdbtoonto.gui` and be derived from `BaseLCSEditPanel` class.

**Example**

Partial definition of the local constraint editor for the `AroundOneTable` converter.

```

public class AroundOneTableGui_LCSEditPanel extends BaseLCSEditPanel {
    private static JTextArea _CommentField;
    private static JCheckBox _CommSW;
    public AroundOneTableGui_LCSEditPanel(LocalConstraintSet cd) {
        super(cd);
        GridBagConstraints c = new GridBagConstraints();
        ...
    }
}

```

```

        _CommSW = new JCheckBox("Comment");
        c.gridx = 0;
        c.gridy=4;
        add(_CommSW,c);
        c.gridx = 1;
        _CommentField = new JTextArea();
        ...
        add(_CommentField,c);
        ...
    }
}

```

The following methods should be overridden:

- `displayInitValues()`: displays from the current local constraints set (in `_LCS` field) of the option values.
- `getValuesFromGui()`: gets the values from the interface.
- `fillSlots()`: When a table is selected in the local constraint set view, this method sets the fields in the corresponding view with default values.

Each implementation of the methods should start with a call to the super class method.

We give below the implementation for the method `displayInitValues()` from the `AroundOneTable` converter. See the source code for the implementation of `getValuesFromGui()` and `fillSlots()` methods.

### Example

```

public void displayInitValues() {
    super.displayInitValues();
    if (_LCS != null) {
        AroundOneTableLCS aotcd;
        if (_LCS instanceof AroundOneTableLCS)
            aotcd = (AroundOneTableLCS)_LCS;
        else {
            aotcd = new AroundOneTableLCS(_LCS);
        }
        _CommSW.setSelected(aotcd._CommAllowed);
        _CommentField.setText(aotcd._Comment);
        _CommSW.setSelected(aotcd._CommAllowed);
    }
}

```

See the sample code for further details.

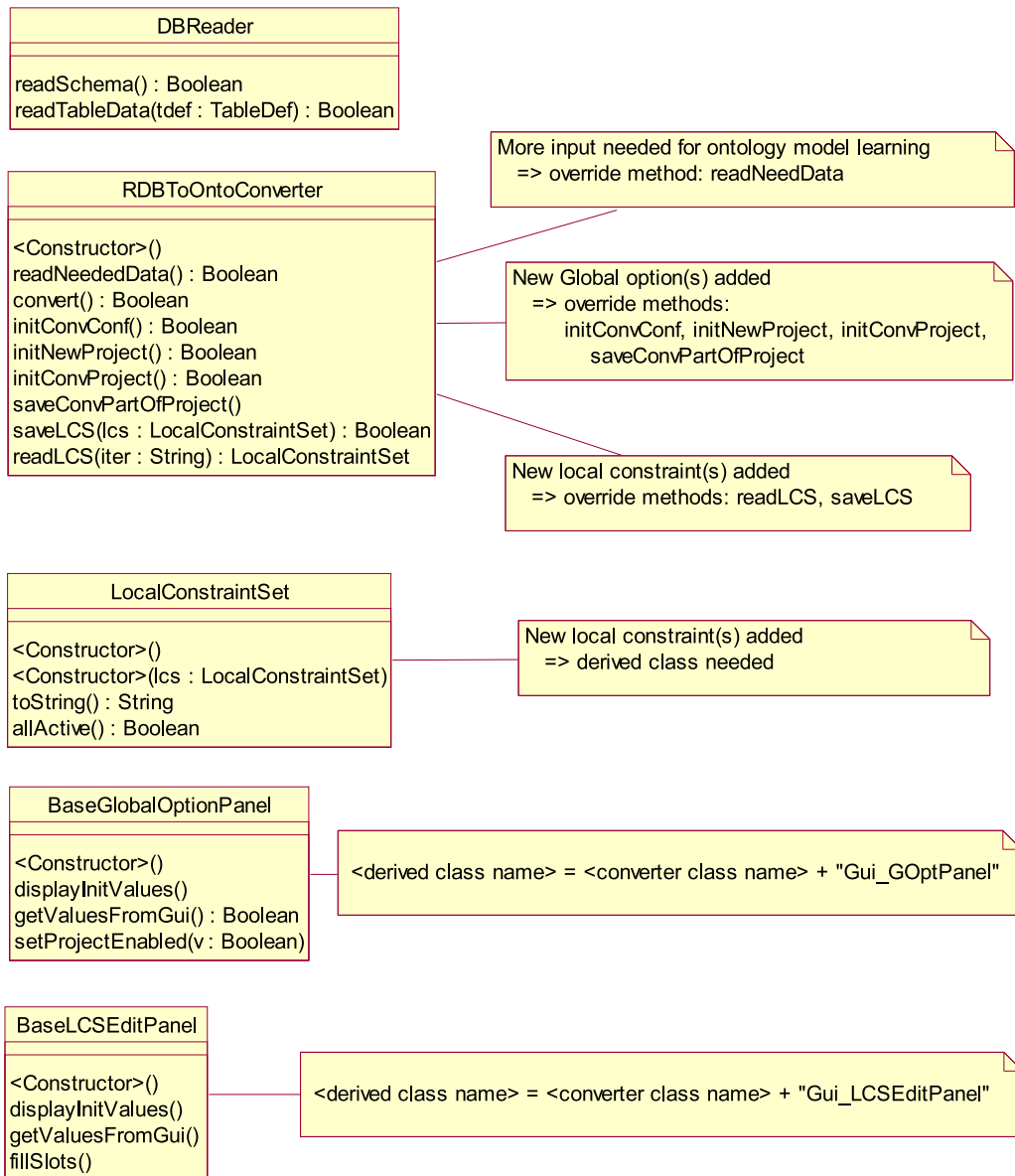


Figure 2: Methods that may be specialized when adding new readers and converters.