

Scriptella ETL Reference Documentation

1.0

by **Fyodor Kupolov**

NOTICE: Work in progress

Table of contents

1 Introduction.....	4
1.1 When to use.....	4
2 System Requirements.....	5
3 Installation.....	5
4 Scripts Syntax.....	6
4.1 <properties>.....	6
4.2 <connection>.....	7
4.3 <script>.....	8
4.4 <query>.....	8
5 Expressions and Variables Substitution.....	9
5.1 Implicit Variables.....	10
5.2 Adding references to files from SQL scripts.....	10
5.3 Examples.....	11
6 Command Line Execution.....	11
6.1 Command Line Options.....	11
7 Ant Integration.....	12
7.1 System requirements.....	12
7.2 Installation.....	12
7.3 "etl" Task.....	12
7.4 <etl-template> Task.....	13
8 In-Process Java Integration.....	14
9 JMX Monitoring and Management.....	15
9.1 Attributes.....	15
9.2 Operations.....	16
10 JDBC Adapters.....	16
10.1 Autodiscovery of JDBC drivers.....	16
11 Non-relational Datasources Interoperability.....	17
11.1 Accessing directories based on LDAP.....	17

11.2 Working with CSV and text data.....	17
11.3 Working with XML data.....	17
11.4 Using Java Code.....	17
11.5 Interaction with Scripting Languages.....	18
11.6 Producing Reports with Velocity.....	18
11.7 URL schemes supported by Scriptella.....	18
12 Examples.....	19
13 Best Practices.....	20
13.1 Using Scriptella as a Database Schema Evolution Tool.....	20

1. Introduction

Scriptella is a Java-based ETL and scripts execution tool. The primary scripting language is a plain old SQL executed by the JDBC bridge. At the same time other non-JDBC providers could be easily added allowing to mix SQL scripts with other scripting languages.

The following diagram is an overview of the Scriptella execution model:

A typical ETL process collects data from one or many datasources and loads it into other datasources optionally performing necessary transformations. To accomplish these tasks Scriptella provides 3 basic elements:

- **Connection.** Represents a connection to a datasource like a database, directory service, XML file etc.
- **Script.** Contains code to execute, i.e. SQL, JavaScript, JEXL or even a DSL (Domain Specific Language).
- **Query.** Contains code written in DSL to query a datasource. Queries can contain nested queries and scripts.

One of the primary Scriptella features is ability to link queries and scripts written in different languages while sharing the same variables (e.g. database columns, XML or LDAP attributes or RegEx groups). To use this power you have to understand how queries work in Scriptella. For the sake of simplicity queries can be considered as "for each" statements which loop through the set of rows (JDBC ResultSet, LDAP entries, XML DOM elements etc.) and execute nested elements passing attributes/columns of each row as variables.

1.1. When to use

There is no silver bullet and you have to pick a right tool aligned with your goals. Here is a list of scenarios when Scriptella may be a very helpful tool:

- Storing database SQL scripts. Scriptella allows to easily execute SQL scripts without dependencies on database vendor SQL tools. Moreover you can reference BLOB content, e.g. files, and use other JDBC features in your scripts.
- Working with several datasources. For example gathering statistics from the one database and store it inside the other one
- If you prefer to write


```
SELECT * FROM Account WHERE login=?name
```

 instead of using a "Smart Query Designer GUI"
- Test data generation. You don't have to store huge random generated test files - produce test data during script execution.
- Migrations. Scriptella supports multiple migrations styles including Ruby-on-Rails-like

schema evolution and data migration.

Of course there are situations when you'd probably try to find a better tool:

- Database replication, i.e. one-to-one copy of the database content. There are plenty of tools doing this job easier.

2. System Requirements

Scriptella requires JDK or JRE version 5.0 or later to operate.

The memory requirements are dependant on your script content and referenced connection providers. For example if in-process database connections are used inside a script we recommend to use at least 512Mb of RAM.

3. Installation

- [Download](#) Scriptella binary distribution.
- Unpack it and add a <SCRIPTELLA_DIR>/bin to a system PATH variable.
Use `set PATH=%PATH%;SCRIPTELLA_DIR\bin` for Windows and `export PATH=${PATH}:SCRIPTELLA_DIR/bin` for Unix.
- Check if JRE has been installed correctly by running `java -version`. Alternatively `JAVA_HOME` may be used to specify JDK location.
- *Optional step:* Put JDBC drivers required by your scripts to <SCRIPTELLA_DIR>/lib directory or directly specify classpath attribute in script connection elements.

The binary distribution of Scriptella has the following directory layout:

```
+--+
|  +--- bin - launch scripts
|  |
|  +--- docs - documentation
|  |   +--- api - API Documentation (in Javadoc format)
|  |   |
|  |   +--- dtd - Scriptella DTD file and documentation in DTDDoc format.
|  +--- lib - scriptella boot class path libraries loaded by launch scripts.
```

Among other libraries the Scriptella distribution contains the following jars:

- `scriptella-core.jar` - Scriptella core classes required to operate
- `scriptella-drivers.jar` - Built in drivers and adapters for different datasources
- `scriptella-tools.jar` - Misc tools, Ant and command line launchers
- `scriptella.jar` - All-In-One jar containing all dependencies except the driver specific. You have to load additional driver dependencies in a connection classpath

attribute or by putting the jars into lib folder.

For convenience the Java source files are stored inside `scriptella-src-ide.zip` file. This file can be used inside IDE to attach sources for `scriptella.jar`

4. Scripts Syntax

Note:

For details about all XML elements consult a [Scriptella DTD Documentation](#)

The following XML stanza briefly covers main Scriptella XML elements:

```
<!DOCTYPE etl SYSTEM "http://scriptella.javaforge.com/dtd/etl.dtd">
<etl>
  <description></description>
  <properties>
    <include href="script.properties"/>
    driver=org.jdcDriver
  </properties>
  <connection id="con1" driver="$driver" url="{url}" user="$user"
password="12345678">
    driver.property=value
  </connection>
  <connection id="con2" url="jdbc:hsqldb:file:db" user="sa" password=""/>
  <script connection-id="out">
    <include href="dbschema.sql"/>
  </script>
  <query connection-id="in">
    SELECT * from Bug
    <script connection-id="out">
      INSERT INTO Bug VALUES (?ID, ?priority, ?summary, ?status);
    </script>
  </query>
</etl>
```

`<etl>` is a root element of all Scriptella files.

4.1. <properties>

In this element define properties to be substituted in other script elements. This concept is similar to Ant.

`<include>` element is used to insert external content. This concept is almost identical to XInclude

Note:

The values of properties defined in this element are overridden by system properties or other external properties depending on

execution environment, e.g. Ant properties.

4.2. <connection>

Elements of this type define connections to different datasources to work with. A set of required attributes depends on driver. For example JDBC drivers require url, user and password attributes. Additionally you may specify catalog/schema attributes.

id attribute is required if you declare more than one connections inside your script. The value of this attribute is used to reference the connection in scripts and queries.

url attribute specifies connection URL. URL format is driver specific, e.g. `jdbc:` for JDBC drivers. Other drivers like CSV or text accept absolute URLs or file locations relative to an ETL file.

classpath optional attribute is used to specify additional classpath to load driver libraries. Paths are colon/semicolon separated and resolved relative to a directory where ETL file resides. Absolute URLs are also supported, e.g. `ftp://server/jconnect.jar;http://server2/classes12.zip` Alternatively you can add drivers jars to scriptella boot classpath (`SCRIPTELLA_DIR/lib` directory or in Ant taskdef classpath). Boot classpath libraries take precedence over libraries specified in connection classpath attribute.

driver optional attribute specifies driver class name or an alias. The default value is "auto", in this case a target driver is guessed from an url attribute if possible. See [Drivers matrix](#) for the list of driver aliases included into Scriptella distribution.

lazy-init optional attribute specifies connection initialization strategy. `true` if connection initialization should be deferred until it is actually used. The default value of `false` means that connection is initialized on ETL startup. Lazy connection initialization may be useful in the following cases:

- A script or a query operating with this connection has conditional **if** attribute, in this case greedy initialization may be redundant
- Other connections create database users or produce files required for this connection to be initialized properly.

Use element's content to set connection properties. Several drivers also accept connection properties in a URL string, e.g. `jdbc:url:db;param=value`

Example

```
<!-- Declare a connection to Oracle database using sys user-->
<connection driver="oracle" url="jdbc:oracle:thin:@localhost:1521:DB"
user="sys as sysdba" password="password">
```

```
#Connection properties
  plsql=true
</connection>
```

4.3. <script>

Elements of this type contain executable content. The script language is connection specific. This element supports the following optional attributes:

connection-id attribute specifies a connection identifier.

new-tx attribute specifies transaction semantics. If new-tx=true a dedicated connection is used instead of the global one.

if attribute is used for conditional execution. The element content is executed only if the result of this attribute value evaluation is true. For example

```
<script if="rownum gt 1 and name!=null and name.length() gt 0">
```

is evaluated only if rownum variable has value greater than 1 and the name variable is not empty.

Example

```

        <!--A script executed in a separate
connection(new-tx=true)
        if drop_tables property has been set to true. -->
        <connection id="in" ...>
        ....
        <script connection-id="in" new-tx="true"
if="drop_tables">
            DROP TABLE Table1;
            DROP TABLE Table2;
        </script>
```

4.4. <query>

Elements of this type contain query expressions. The query language is connection specific. This element supports the following optional attributes:

connection-id attribute specifies a connection identifier. By default connection ID is inherited for child elements, but may be overridden.

if attribute is used for conditional execution. The element content is executed only if the result of this attribute value evaluation is true. For example

```
<query if="migrate_users">
```

is evaluated only if migrate_users variable has value of true, 1, on or yes.

Example

```

        <!--Declare connections -->;
        <connection id="in" .../>;
        <connection id="out" .../>;
        ...
        <!--Query selects users and executes nested elements if
migrate_users is on-->;
        <query connection-id="in" if="migrate_users">
            SELECT * FROM Users;
            <!--For each row sends LDIF entry to a directory
server-->
                <script connection-id="out">
                    dn: uid=$user_name,ou=people,dc=scriptella
                    objectClass: inetOrgPerson
                    uid: $user_name
                    cn: $user_name
                </script>
            </query>

```

FIXME (ejboy):

Describe all elements

5. Expressions and Variables Substitution

Binding variables syntax varies between drivers. JDBC drivers support variables substitution and expression evaluation based on the following syntax:

- `$Property_Name` - inserts a value of the specified property or variable. `$` prefixed expressions are substituted in all parts except comments.
- `${expression}` - inserts a value of the expression evaluation. [JEXL](#) syntax is used for expressions. `$property` and `${property}` return the same value, but the first one is faster, because JEXL expression engine is not used to evaluate the expression.
- `?Property_Name` or `?{expression}` are used *only in SQL queries and scripts* to set prepared statement parameters. `?` prefixed expressions are not substituted inside quotes, comments and outside of query/script elements.

The properties syntax described above is allowed in the following places:

- Attributes of ETL `<connection>` element,
- `if` attribute of `<query>/<script>` elements,
- SQL scripts/queries processed by the JDBC-based providers

Due to several JEXL limitations, Scriptella provides a helper variable available inside JEXL expressions. The [etl](#) variable provides several utility objects:

- `etl.date` - Methods for parsing and formatting dates.
Example: `${etl.date.today('YYYYMMdd')}` - formats the current date using the specified pattern. See [JavaDoc](#) for more details.
- `etl.text` - Methods for working with text.
Example: `${etl.text.isNull(a)}` - returns empty string if a=null. See [JavaDoc](#) for more details.

Other providers may not support properties/expressions substitution or use alternative syntax, e.g. in Janino use `get(name)` to get variable's value. For details on variables/expressions syntax consult the driver's specific [Javadoc](#). The basic support for properties substitution is provided by [PropertiesSubstitutor](#) core class which is based on a `${}` syntax and used by drivers unless other rules are described.

Note:

In SQL scripts and queries try to minimize usages of \$, use ? instead. In this case prepared statements may be cached thus increasing the performance of hot spot sections evaluated multiple times. Use \$ for direct substitution only if you have problems passing prepared statement parameters via ?, ?{ } syntax.

5.1. Implicit Variables

Scriptella supports the following implicit variables in scripts and queries:

Variable	Description	Scope
<code>rownum</code>	Represents row number of query row set. Starts with 1.	Nested elements of Query

Additionally SQL queries produce a set of implicit variables corresponding to selected column names. These variables are available in nested elements. Columns may also be referenced using a result set column number, e.g. ?1, ?2 ,...

Typically non-SQL queries also produce virtual row sets and expose variables available in nested elements. Consult driver specific Javadoc for more details.

5.2. Adding references to files from SQL scripts

The JDBC bridge supports referencing file content via `?{file <Expression>}` and `?{textfile <Expression>}` syntax. Where Expression is any valid JEXL expression, e.g. 'file.dat' or protocol+path

The `file` prefix is used to reference binary files inserted as BLOB and `textfile` references text files inserted as CLOBs. File references are set as prepared statement parameters of stream type, so LOBs of virtually any size supported by the database driver can

be referenced. Please note that uploading large **text** files may create a temporary file on disk.

Note:

Music Store example demonstrates usage of file references

5.3. Examples

Execute a SQL query over a table specified by a TABLE property and inserts the first row into TABLE2

```
<connection driver="hsqldb"/> <!--JDBC Bridge Driver-->
<query>
  SELECT V1, V2, V3 from $TABLE;
  <script if="rownum=1">
    INSERT INTO $TABLE2 VALUES (?V1, ?{V2+V3});
  </script>
</query>
```

6. Command Line Execution

If you followed the [Installation](#) instructions running Scriptella is simple - just type scriptella to execute a etl.xml in the current directory.

The Scriptella launching script loads all jars from SCRIPTELLA_DIR/lib directory and adds them to executed script(s) class path.

Alternatively a standard java launcher may be used to run a script: java -jar scriptella.jar [options] [file1] [file2] ... [fileN]. It is assumed that the scriptella.jar file from the binary distribution is available in the current directory.

Important

Java launcher does NOT load jars from lib directory so you would have to specify additional class path elements in script file connection declarations or use java -Xbootclasspath/a:path/to/driver1.jar;path/to/driver2.jar -jar scriptella.jar to overcome -jar option limitation.

6.1. Command Line Options

The scriptella launcher has the following invocation syntax:

```
scriptella [options] [file1] [file2] ... [fileN]
```

File names/paths are specified after options and separated by spaces.

-help, -h	display help
-debug, -d	print debugging information

-nostat	Suppress statistics collecting. Improves performance.
-quiet, -q	be extra quiet
-version, -v	print version
-template, -t	creates an etl.xml template file in the current directory. See the list of supported templates .

FIXME (ejboy):

Describe command line tools

7. Ant Integration

7.1. System requirements

Ant 1.6 or later is recommended.

7.2. Installation

Use scriptella.jar from the binary distribution. This file contains all necessary classes and resources for integration with Ant.

In order to use scriptella as Ant task you will need the following taskdef declaration:

```
<taskdef resource="antscriptella.properties"
classpath="/path/to/scriptella.jar"/>
```

You may also add paths to database drivers used inside the script:

```
<taskdef resource="antscriptella.properties"
  <classpath>
    <pathelement location="/path/to/scriptella.jar">
    <pathelement location="lib/hsqldb.jar">
    <pathelement location="lib/jconn2.jar">
  </classpath>
</taskdef>
```

These drivers will be available on the boot classpath. In this case you don't have to specify classpath in <connection> elements.

7.3. "etl" Task

7.3.1. Parameters

Attribute	Description	Required
-----------	-------------	----------

file	The script file to execute.	Yes, unless a nested <fileset> element is used.
inheritAll	If true, pass Ant project properties to the Script Executor.	No, default value is true.
debug	If true, print debugging information.	No, default value is false.
nostat	If true turns off statistics collecting.	No, default value is false.
quiet	If true, be extra quiet.	No, default value is false.

7.3.2. Nested Elements

The task supports nested [<fileset>](#) element.

7.3.3. Examples

Executes etl.xml file in the current directory:

```
<etl/>
```

Executes name.etl.xml file in the current directory:

```
<etl file="name" />;
<!--Or explicitly specifying the full name-->
<etl file="name.etl.xml" >
```

Executes all .etl.xml files in db directory:

```
<etl>
  <fileset dir="db" includes="*.etl.xml"/>;
</etl>
```

7.4. <etl-template> Task

7.4.1. Parameters

Attribute	Description	Required
name	ETL template name.	No, default ETL template is generated.
inheritAll	If true, pass all properties to Scriptella. Defaults to true.	No
debug	If true print debugging information.	No, default value is false.

quiet	If true be extra quiet.	No, default value is false.
-------	-------------------------	-----------------------------

7.4.2. Nested Elements

The task has no nested elements.

7.4.3. Supported Templates

- Default(no name) - Produces a simple ETL template for a quick start.
- [DataMigrator](#) - Produces an ETL template for transferring data between tables of different databases.

7.4.4. Examples

Produces default ETL template named etl.xml in the current directory:

```
<etl-template/>
```

Produces data migration template named etl.xml in the current directory:

```
<property file="etl.properties"/>
<!--
Ant properties
  driver,class,user,password
must be set before calling "DataMigrator" etl-template
-->
<etl-template name="DataMigrator"/>
```

8. In-Process Java Integration

Although Ant and command line execution are the most typical usage scenarios, it may be helpful to invoke Scriptella directly from Java code. Typical use cases for in-process integration:

- Preparing datasources for testing. Example: create the database before a test case, and cleanup on test completion.
- Creating/upgrading database schema on application startup. Example: creating a database on web application initialization, very helpful in user friendly, zero-deployment or demo applications.
- Custom migration solutions. Example: Client uploads a CSV file to a J2EE server and then Scriptella managed ETL script imports CSV data into several database tables.

The invocation from java code is simple:

- Make sure scriptella.jar is available on classpath.

- Use `EtlExecutor.newExecutor` to execute an ETL file. See [EtlExecutor Javadoc](#) for more details.

Note:

Typically you will need a [Spring Driver](#) or a [JNDI Driver](#) to integrate with application deployment environment (Spring, J2EE or other).

9. JMX Monitoring and Management

Scriptella supports JMX by registering a dedicated MBean for each ETL operation. ETL mbeans use the following naming convention:

```
scriptella:type=etl,url=<ETL_XML_FILE_URL>[ ,n=<COLLISION_ID> ]
```

Note:

The COLLISION_ID is appended only if files with identical URLs are executed simultaneously.

ETL MBeans are registered automatically when Scriptella is launched from the command line or Ant. If `EtlExecutor` is invoked directly from Java code, set `jmxEnabled` property to `true` by calling `etlExecutor.setJmxEnabled(true)` before executing the ETL.

By default the mbeans are registered in a platform default MBean server and the Java Monitoring and Management Console (`jconsole.exe`) can be used for remote control over ETL tasks.

Note:

Since Java SE 6 it is allowed to monitor local JVM without additional configuration changes. Just run `jconsole.exe` and choose a required JVM
For Java SE 5 -`dcom.sun.management.jmxremote` system property enables the JMX agent for local access.
See also [Monitoring and Management Using JMX](#) for more details.

9.1. Attributes

The following read-only attributes are available for monitoring:

- `ExecutedStatementsCount` - number of executed statements by all connections of the ETL task.
- `StartDate` - date/time when ETL was started.
- `Throughput` - the throughput (number of statements per second) of the managed ETL task.

9.2. Operations

Currently only cancellation is supported via JMX. The `cancel` operation terminates ETL and tries to roll back any changes made during the ETL execution.

10. JDBC Adapters

Scriptella is bundled with a set of adapters for popular JDBC drivers. It is allowed to use any JDBC vendor driver, but Scriptella adapters provide the following benefits:

- Simplified naming. No need to specify a full driver's class name, alias may be used, e.g. `driver="hslqdb"`, `"oracle"` etc.
- Scriptella adapters are preconfigured for optimal performance, syntax parsing rules and other execution options, e.g. BLOB handling for Oracle. In most cases you can specify a generic JDBC property, but using adapters simplifies switching between drivers and even a target platform e.g. Java/.Net
- Different JDBC drivers can be used to access popular databases such as Microsoft SQL Server or Sybase ASE/ASA. Each driver has an individual class name. Example list of driver names for Sybase ASA/ASE: `com.sybase.jdbc3.jdbc.SybDriver`, `com.sybase.jdbc2.jdbc.SybDriver`, `com.sybase.jdbc.jdbc.SybDriver`, `net.sourceforge.jtds.jdbc.Driver`. With Scriptella you simply specify `"jdbc"` as a name of the driver and an available driver is loaded from classpath.

See [JDBC Bridge Drivers Matrix](#) for the complete list of supported adapters.

Note:

Scriptella driver's package short name is an alias for the driver, e.g. `h2` is an alias for `scriptella.driver.h2.Driver`

10.1. Autodiscovery of JDBC drivers

Scriptella provides auto-discovery feature based on the connection URL. If driver name is not specified the driver will be recognized based on the specified URL. Internally a driver named `"auto"` is used to select a target driver implementation. Example:

```
<connection url="jdbc:sybase:Tds:host:2048/database" />
```

The `"auto"` driver will load the `"sybase"` driver, because the URL belongs to Sybase.

Note:

Autodiscovery serves for convenience purposes and works only with JDBC drivers and URL schemes supported by Scriptella, in other cases driver's name has to be specified manually. See [JDBC autodiscovery driver JavaDoc](#) for autodiscovery implementation details.

11. Non-relational Datasources Interoperability

See [Non-JDBC Drivers Matrix](#) for the complete list of drivers.

Additionally various JDBC bridge drivers can be used to work with specific datasources like LDAP directories, CSV files, XML files or object databases.

11.1. Accessing directories based on LDAP

The feature of Scriptella is a possibility to write transformations in a language suitable for the datasource you operate, so we've added a [built-in support](#) for LDIF scripts and RFC 2254 search filter queries.

Alternatively you can use the freely available [JDBC-LDAP Bridge Driver](#) (JDBC-LDAP) to access information held in directories. This driver uses SQL-like syntax for scripts and queries. See [Getting Started Documentation](#). A similar driver is offered by Novell. See [LDAP JDBC Driver](#) for more details.

11.2. Working with CSV and text data

The [built-in CSV driver](#) allows to easily query and generate CSV files.

The [built-in Text driver](#) provides a generic way to handle text data.

HSQLDB [text tables](#) feature can be used to work with CSV files.

11.3. Working with XML data

The [built-in XPath driver](#) allows querying XML files with XPath expressions.

[Text driver](#) can be used to output XML files.

11.4. Using Java Code

Although Scriptella was designed for JDBC/SQL datasources under some circumstances it is useful to embed or invoke Java from Scriptella file. The [Janino Provider](#) is bundled with Scriptella distribution and provides a Java bridge. This driver also exposes a set of properties and methods for [<script>](#) and [<query>](#) elements.

Scriptella has no equivalent of Ant Task API, although we provide two alternatives to integrate Java solutions:

- Writing a Service Provider Implementation (Scriptella Driver). This is the most powerful

API, but it is harder to implement.

- Calling compiled code from Janino scripting elements. Just specify your jar as an additional class-path entry for Janino connection and you can work with any class from this jar. This approach is easy-to-use and do not add dependencies on Scriptella API to your classes/tasks.

Other Java compilers or interpreters can also be plugged in via custom Scriptella Drivers.

11.5. Interaction with Scripting Languages

Java SE 6 includes [JSR 223: Scripting for the Java™ Platform](#) API. This is a framework by which Java Applications can "host" script engines. Sun's implementation of Java SE 6 includes an example script engine based on [Mozilla Rhino:JavaScript for Java](#).

Scriptella supports JSR 223 compatible scripting languages via [javax.script Bridge](#) driver.

Note:

Java SE 6 or higher is required to work with JSR-223 scripts.

11.6. Producing Reports with Velocity

Velocity is a Java-based template engine which can be used to generate reports. Most other ETL tools provides custom reporting services, but Scriptella relies on proven open source solutions instead of reinventing the wheel. The built-in [Velocity Driver](#) allows using velocity templates in <script> elements. Typically velocity report is generated in several steps:

1. Printing a header
2. Query data from datasource(s) and produce a report body.
3. Printing a footer.

Primes example demonstrates how to produce a simple report using Velocity.

Note:

Velocity Driver does not support query element. This limitation may be eliminated in future, but currently we do not see Velocity strengths for querying the data. JDBC, JEXL or Janino drivers provide more powerful and easy-to-use solutions for dynamic queries.

11.7. URL schemes supported by Scriptella

Scriptella allows specifying URLs to externally located resources. The following protocols are supported:

- http, ftp, file, jar and other protocols supported by JRE

Examples:

```

        <connection driver="xpath"
url="http://snippets.dzone.com/rss/tag/scriptella"/>
        <connection driver="csv"
url="http://finance.google.com/finance/historical?q=JAVA&output=csv"/>
        <connection driver="text"
url="ftp://ftphostname/report.txt"/>
    
```

12. Examples

Download Scriptella examples from [Downloads](#) page. The examples are ready to run, although you may have to download additional third party jars, which cannot be redistributed, e.g. jdbcldap.jar etc. These dependencies are specified in lib/readme.txt file.

Ant	This example demonstrates integration with ant.
CSV	Example of working with CSV data using HSQLDB CSV tables feature, Scriptella built-in CSV or Text driver.
DB Upgrade	This example shows how Scriptella can be used as a schema evolution tool.
LDAP	Example of migration script to exchange data between a database and LDAP directory. This example shows 2 ways to access LDAP directories from Scriptella: built-in LDAP driver and JDBC-LDAP Driver from OctetString.
Mail	This example shows how to produce and send E-Mails from Scriptella.
Music Store	This script create a sample database of music track. This sample shows how to reference BLOB content in external files and use JDBC escaping syntax to pass dates independently from the DB locale/date/time settings.
ODBC	Produces a HTML view of the MS Access Northwind Categories table. This example demonstrates working with ODBC datasources (Northwind.mdb) and producing a report with images.
Primes	Fills the database with prime numbers and produces HTML/CSV reports. This example

	demonstrates a combined usage of Velocity, JEXL, CSV Driver and SQL.
XML	This example shows how to work with XML data using XPath expressions.

13. Best Practices

This chapter describes Scriptella usage best practices. The following list provides general recommendations for Scriptella ETL scripts:

- Do not hardcode connection properties and other configuration settings e.g. driver names, urls, mode flags etc. Use a property file like `etl.properties` to store ETL configuration parameters
- Use built-in Scriptella adapters for JDBC drivers if possible.
- Use `<dialect>` for database specific SQL parts. Use properties to configure data types similar to domains, example:

```
CREATE TABLE User (
  ID $INTEGER,
  LOGIN $SMALL_STRING
);
--Values for $INTEGER AND $SMALL_STRING are
--stored in a mapping property file e.g. type-mapping.properties
--and easily reconfigurable for another DB vendor
```

- Use JDBC escaping to specify time and date literals, function calls and other parameters independently from DB vendor. Examples:
Calls without a return value: `{ call procedure_name (argument1, argument2, ...) }`
Calls with a return value: `{ ? = call procedure_name (argument1, argument2, ...) }`
Timestamp Literals: `{ ts 'yyyy-mm-dd hh:mm:ss.f...' }`
- Use `.etl.xml` naming convention for ETL scripts. In this case it would be easy to recognize scriptella files, moreover you wont have to specify `etl.xml` extension to run ETL files. Example: typing `scriptella init` executes `init.etl.xml` automatically if no file `init` was found.
- For large bulk-insert files use `<include>` element to avoid loading a whole file into memory.

13.1. Using Scriptella as a Database Schema Evolution Tool

dbupgrade example shows how to build a simple database upgrade(downgrade) script. The following steps are required to create a simple database upgrade framework:

- If no DB present create an new database and fill it with up-to-date initial script
- If old DB is present apply a set of incremental updates to migrate from version X to Y.

Scriptella provides enough features to create a simple database upgrade framework, but the primary one is the ability to work with several datasources. This means you can change database vendors between builds and synchronize several datasources, e.g. LDAP-Database.

Scriptella is lightweight and can be easily added to a set of application libraries during deployment. In this case you can integrate your schema upgrade solution with application startup procedure typically written as a set of webapp context listeners or initializer servlets.