convertigo
MASHUP YOUR BUSINESS

Starting with Convertigo Mashup Sequencer

Quick Guide

CEMS 5.1.2

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Getting Familiar with Convertigo Mashup Sequencer

This chapter presents the purpose, concepts and objects of **Convertigo Mashup Sequencer (CMS)**, one of Convertigo Enterprise Mashup Studio's components. It also describes the four views of the Eclipse-based CMS Studio and their elements (tabs, icons, menu).

1.1 CMS Purpose

The purpose of Convertigo Mashup Sequencer (CMS) is to "orchestrate" different transactions used by Convertigo to access application sources such as:

- Web Services;
- RSS syndication feed;
- Legacy screens (BULL, IBM);
- HTML pages;
- SQL databases.

Any kind of Convertigo project (Web integration, Legacy integration, etc.) can be considered as a "source" in the context of a sequence.

Convertigo Mashup Sequencer allows you to build server-level mashups designed to:

1 *Call* as many of the above mentioned services as required.

2 *Call* services using input data collected from other services.

3 *Combine* them into one unique response by aggregating, or leaving transaction outputs "as is".

4 *Store* the result into a database, if need be.

Response times can be optimized by setting sequence branches so as to call transactions in parallel.

Convertigo Mashup Sequencer can call either transactions or other sequences. It is

therefore possible to set up complex projects while taking advantage of modularity capability.

CMS encompasses a large number of applications:

- *Mashing-up* several applications at server-level.
- *Creating* XML responses complying with user-defined XML Schema Definition (XSD).
- *Creating* databases from elements collected on the Web (*Data Mashup*).
- *Filling in* Web forms using database data (AutoComplete).

1.2 CMS Concepts and Objects

This section presents the general concept and notions of the application. It then relates notions to real CMS objects, which are defined in detail in a dedicated sub-section.

- General Concept
- CMS Objects
- Definitions

1.2.1 General Concept

CMS orchestrates transactions based on all possible Convertigo ***connectors*** (HTML, Javelin, SQL, etc.). These ***connectors*** are defined at project level (for example, HTML connectors for Web integration projects, or Javelin connectors for Legacy integration projects). Once data has been retrieved from application sources by the transactions, CMS achieves the transactions orchestration by processing data through the execution of actions (or *steps*) defined as part of a ***sequence***.

Two types of sequences can be created in CMS:

1 Sequences orchestrating transactions so as to collect data, arrange ("mix" as required by the user) collected data and produce an arranged XML output.

For an example of this type of orchestration, see "First sequence description" on page 2-3.

2 Sequences orchestrating transactions so as to collect data, arrange ("mix" as required by the user) collected data and insert them in database.

For an example of this type of orchestration, see "Second sequence description" on page 2-5.

In a CMS sequence, a first step can consist in preparing data from sequence inputs and then calling a transaction using this prepared data as input (for example calling a Web integration transaction such as `searchGoogle` with sequence input data as keyword, refer to the *"Starting with Convertigo Web Integrator" Quick Guide*).

Following steps then:

- process the transaction XML response by arranging its content, checking the presence of specific information, extracting information, using extracted information as input variables for calling other transactions or to generate XML content, etc.,
- generate XML elements (simple, complex, concatenated, etc.) in a specific XML structure (possibly matching a pre-defined schema),
- if need be (depending on the type of orchestration chosen), execute SQL queries via SQL transactions set when developing the sequence.

For example, after a transaction's XML response has been generated and knowing the response XML schema, CMS can check the presence of a specific XML node within the XML output using an *if/then/else* conditional step. Depending on whether the node is present or not, two sets of actions can then be triggered (the checked node *is* / *is not* present).

1.2.2 CMS Objects

The following table describes CMS objects:

Table 1 - 1: CMS objects

Object[a]	Description
Connector	Defines the SQL environement accessed by CMS in the context of a sequencing project.
Transaction	Defines SQL requests to be executed.
Variable	Transaction input variable.
Sequence	Defines the sequence of actions (steps) and the order and conditions of their execution.
Step	Actions to be executed / conditions to be checked as part of a sequence.
Variable	Sequence input variable.

a. *Objects are indented in the column according to their indentation level in the Project tree of the Projects View (see Figure 1 - 1)*

All of these objects are set when developing a Mashup Sequencing project. Once the project has been created, these objects appear in the **Projects View** of the CMS Studio (*"CMS Studio"* on page 1 - 9):

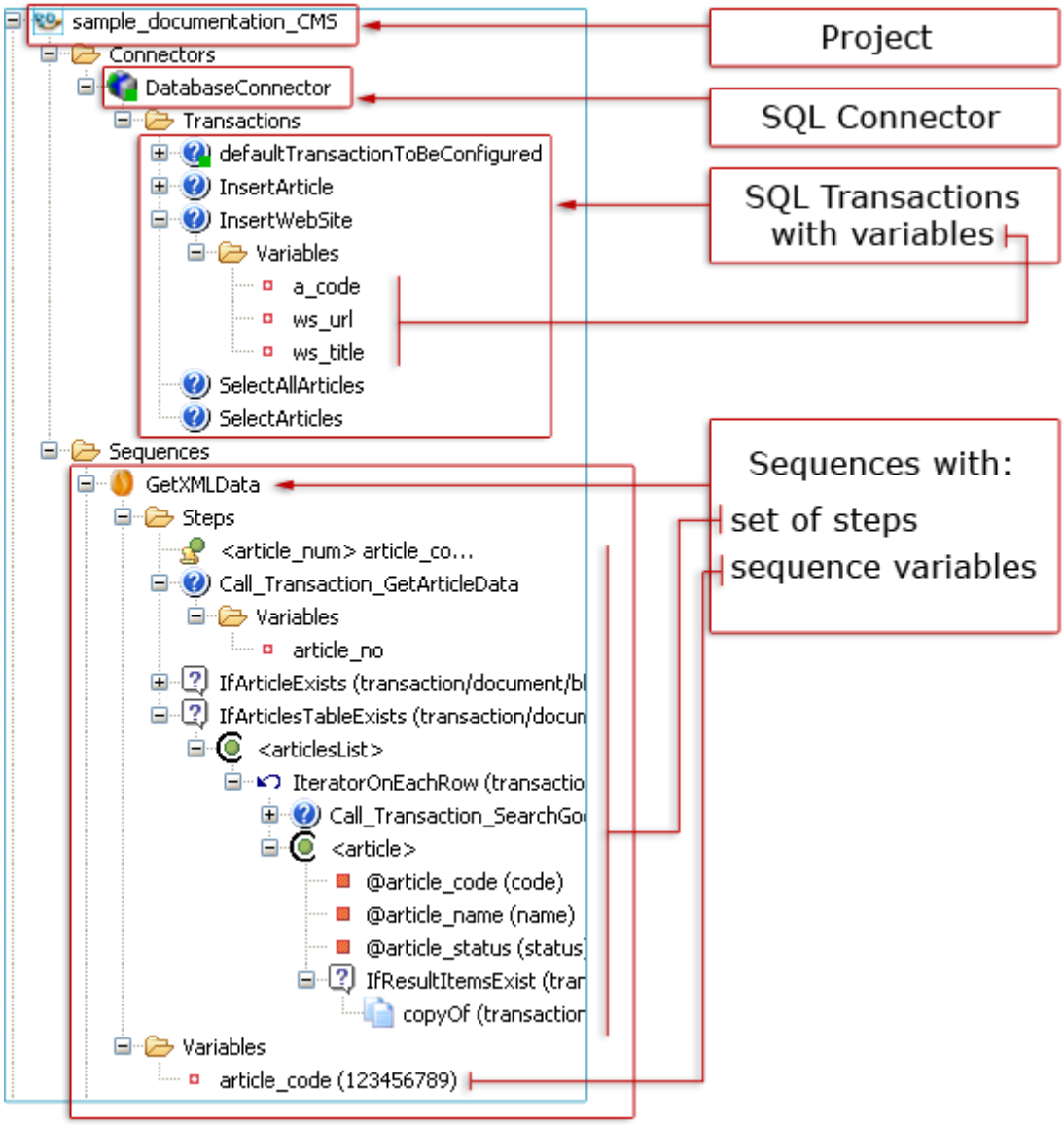


Figure 1 - 1: CMS objects

To get started with your first Mashup Sequencing project, see *"My First Convertigo Mashup Sequencer Project" on page 2-1*

1.2.3 Definitions

CONNECTOR

A connector defines the type of application or data source CMS is able to connect to.

Any type of connector (HTML, legacy, etc.) can be added to a sequencer project, but only one is specific to CMS: the SQL connector, which enables CMS to connect to anumber of SQL databases.

Once a new project has been created, the corresponding connector appears in the *Connectors* folder of the project, in the **Projects View** of the CMS Studio (*"CMS Studio"* on page 1 - 9):

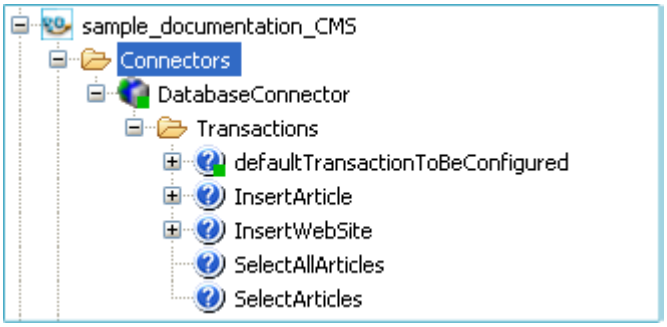


Figure 1 - 2: Connectors folder

Connector parameters appear in the **Properties View** when selecting a connector. The table below describes these parameters:

Table 1 - 2: SQL connector parameters

Parameters	Description
Comment	Defines any comment typed in by the programmer. May be used for retro documentation.
Database url	Defines the database URL, the syntax of which depends on the selected driver class.
Driver	Defines the JDBC driver class to be used for the connection to the database. The driver is selected in a drop-down menu containing the following drivers: • `sun.jdbc.odbc.JdbcOdbcDriver` (*JDBC-ODBC* bridge for accessing ODBC databases, for example *Microsoft Access*) • `com.ibm.as400.access.AS400JDBCDriver` (*IBM AS400* database) • `com.mysql.jdbc.Driver` (*MySQL* database) • `net.sourceforge.jtds.jdbc.Driver` (*Microsoft SQL Server* database) • `org.hsqldb.jdbcDriver` (*HSQLDB* database, one is included in the Studio for demos and samples)
Max. connections	The Convertigo SQL connector uses a connection pool to access the target database. This property defines the maximum number of parallel connections allowed to this connector to connect to the target database.
User name	Defines the user name used for the database connection.
User password	Defines the user password used for the database connection.

TRANSACTION

In the context of CMS, two types of transactions can be called from a sequence:

- transactions defined outside a CMS project as part of other Convertigo projects, such as `searchGoogle` for the sample CWI project (refer to the *"Starting with Convertigo Web Integrator" Quick Guide*) or `GetArticleData` for the sample CLI project (refer to the *"Starting with Convertigo Legacy Integrator" Quick Guide*). These transactions are not described here.
- SQL transactions specifically defined for CMS projects, sub-objects of SQL connector.

An SQL transaction is always associated with an SQL query defined when creating the transaction:

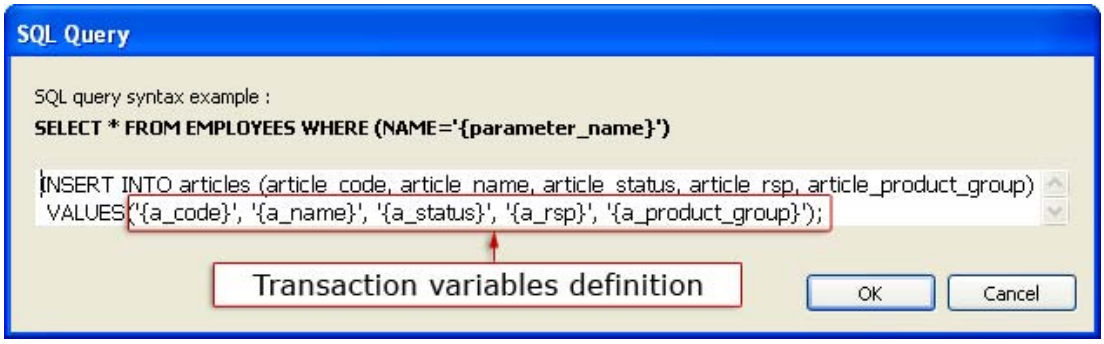


Figure 1 - 3: SQL transaction query

If the query is written with variable values (defined using a Convertigo-specific syntax), transaction variables are automatically generated when validating the query (see Figure 1 - 3 and Figure 1 - 4:).

The SQL query is then executed on call of the transaction. The database is then updated accordingly.

Once a transaction has been defined, it appears together with its variables in the *Transactions* folder of the connector, in the **Projects View** of the CMS Studio (*"CMS Studio"* on page 1 - 9):

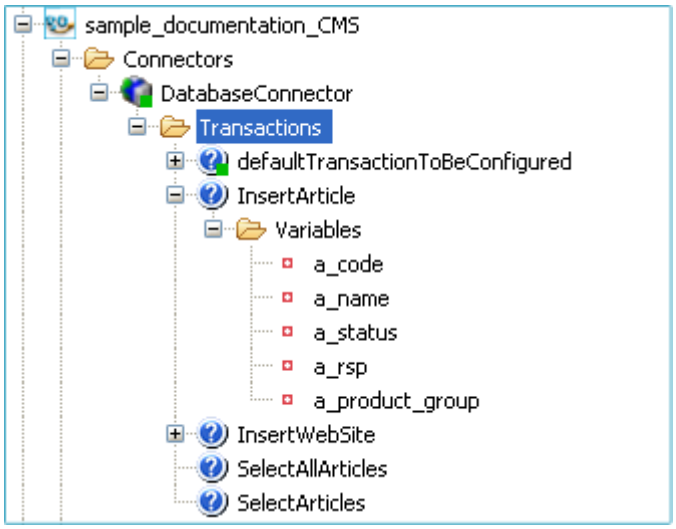


Figure 1 - 4: Transactions folder (transaction and automatically generated variables)

SEQUENCE

A sequence defines a series of actions (*steps*), the order in which they are executed and conditions of execution. A sequence follows a logical process meant at achieving a specific goal.

Any sequence can be triggered as a Web Service in SOAP or REST protocol. A sequence can be set as part of the project containing the transaction to be orchestrated or as part of any other project. For example, a sequencer project can contain sequences orchestrating exclusively transactions from other projects.

Sequences can return XML data combined from the orchestrated transactions to the Web Service caller. Any XML output structure can be defined using appropriate steps.

"Blind" sequences are also possible - they do not return any data to the caller, but can for

example insert data in databases using the defined SQL connector, or insert data into forms (thanks to CWI transactions for example).

Once a sequence has been defined, it appears together with its variables in the *Sequences* folder of the project, in the **Projects View** of the CMS Studio (*"CMS Studio"* on page 1 - 9):

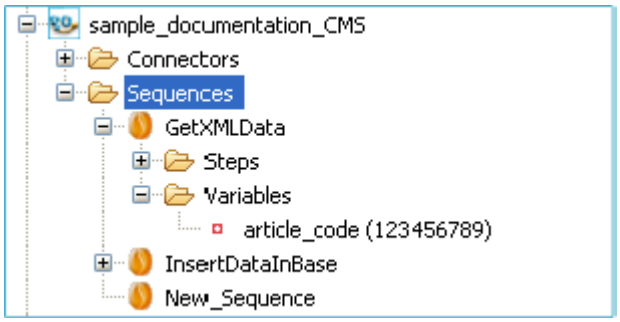


Figure 1 - 5: Sequences folder

STEP

As mentioned before, CMS executes sequences which in turn use a flow of steps to achieve the transaction orchestration.

Steps are the building blocks of sequences. Each step:

- has its own functionality;
- can use data from sources;
- produces XML data which can in turn be used as sources by other steps.

The different types of steps share common properties:

Table 1 - 3: Step common properties

Property	Description
Comment	Basic comment field.
Ouput	If set to `true`, the XML possibly generated by this step is appended to the sequence response. If set to `false`, the possibly generated XML is not added to the sequence XML output. In both cases, the generated XML can be used as source by another step.
Is active	If set to `false`, the step is idle and appears in red in the sequence. Useful for testing purpose.

Once steps have been defined for a sequence, they appear in the *Steps* folder of the sequence, in the **Projects View** of the CMS Studio (*"CMS Studio"* on page 1 - 9):

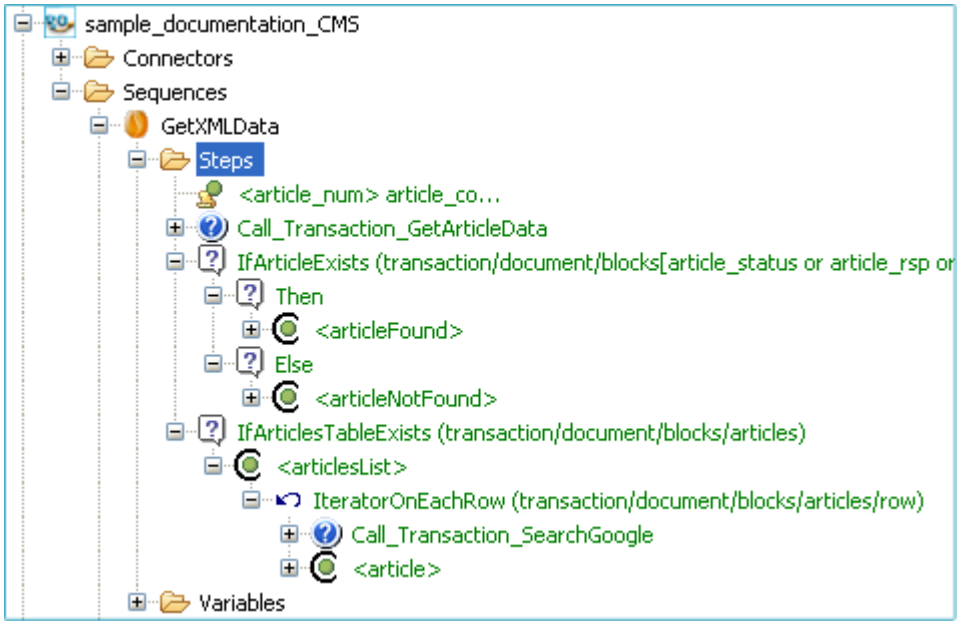


Figure 1 - 6: Steps folder

Steps are based on *sources*. A source can be defined as a path defining the value of a given XML node to be used for the step.

A source is defined using a **Step Source** wizard (see *"Step Source Wizard"* on page 1-15).

VARIABLE

Transactions and sequences are based on variables.

Variables are created either automatically (for example when creating an SQL transaction with a variable-based query, see Figure 1 - 3) or using a **New Variable** wizard.

Once they have been created, variables appear in the transaction / sequence object *Variables* sub-folder in the **Projects View** of the CMS Studio (*"CMS Studio"* on page 1 - 9):

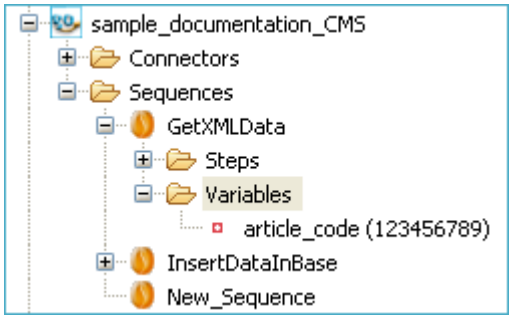


Figure 1 - 7: Variables folder

The table below describes the main variable properties, which can be accessed via the **Properties View** (see *"Properties View"* on page 1-12):

Table 1 - 4: Variable properties

Property	Description
Description	The description value is automatically inserted as comment in the SOAP WSDL.

Table 1 - 4: Variable properties (...)

Property	Description
Default value	Used if no value is provided when the transaction / sequence is called. Useful for testing purpose or for variables not supposed to change. When set, default values appear in brackets to the right of variable names (see Figure 1 - 7).
WSDL exposed	If `true`, variable definitions are inserted in the WSDL.
Multivaluated	If `true`, the variable is an array of values.
Personalizable	If `true`, the transaction / sequence variable will be used as a personalizable preference field while generating a widget for Mashup Composer.
Cached Key	If true the value of this variable will be used as a cache key to determine if the sequence response must be pulled from the cache or not.

1.3 CMS Studio

This section describes the CMS Studio, which comprises four distinct panes: the **Projects View**, the **Connector / Sequence View**, the **Properties View** and the **Console view**.

- General Presentation
- View Description

1.3.1 General Presentation

Based on an Eclipse platform, the CMS Studio is divided into four views that all include:

- one or several tabs, located either on the upper or on the lower part of the view;
- icons;
- three standard Eclipse buttons (**Menu** , **Minimize** and **Maximize**)

The four views are illustrated and named in the figure below:

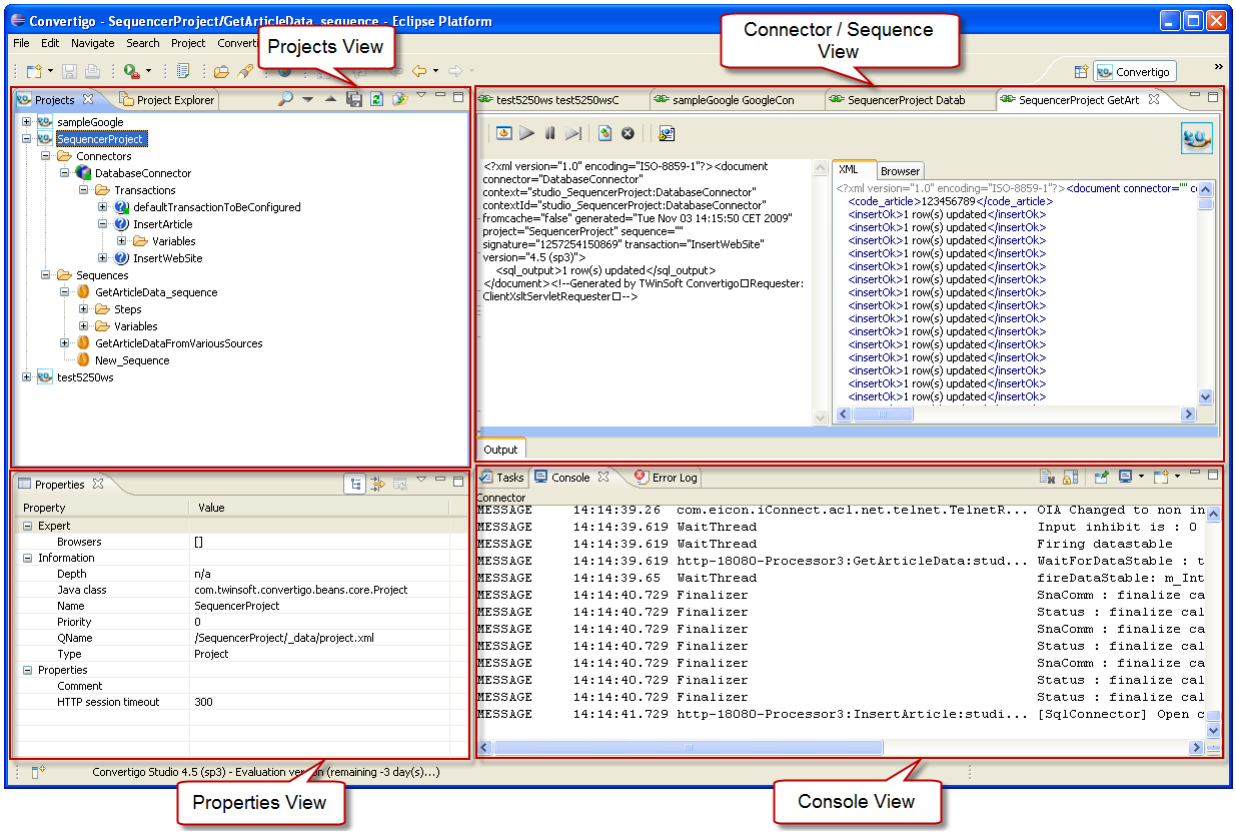


Figure 1 - 8: CMS Studio

The following table describes the views of the Convertigo Mashup Sequencer studio:

Table 1 - 5: CMS views

View Name	Description
Projects View	Displays in a tree structure all CMS projects and their related objects: connectors, transactions and sequences.
Properties View	Displays the properties of the CMS object selected in the **Projects View**. Used to modify a given object's properties.
Connector / Sequence View	In the context of CMS, the **Connector / Sequence View** displays information related to: • **each connector** in each **Connector View** - when a connector is opened by double-clicking on the object in the **Projects View**, a *<ProjectName> <ConnectorName>* tab is opened and contains the **Connector View**. In the case of a SQL connector, this view displays ongoing SQL transactions. • **each sequence in each Sequence View** - when a sequence is opened by double-clicking on the object in the **Projects View**, a *<ProjectName> <SequenceName>* tab displays sequence-specific information such as the XML generated thoughout the sequence, errors, etc.
Console View	Standard Eclipse console. Displays information about running tasks, errors, etc. Different consoles are available (standard output, Engine, Studio, etc.).

1.3.2 View Description

All four views contain common standard Eclipse buttons in the upper right corner of the view:

Table 1 - 6: Standard Eclipse buttons

Button	Icon	Description
Menu		Opens a drop-down menu with a content identical to the view toolbar.
Minimize		Minimizes the current view without closing it. The view is still visible in the form of an icon that can be clicked to be enlarged again.
Maximize		Maximizes the current view to full screen. Maximizing a view can also be achieved by double clicking the upper bar of the view.

The following section describes the four panes of the CMS Studio in further detail.

Projects View

The **Projects View** displays all information (objects, folders and files) associated with a project. Information appears in distinct tree structures depending on the selected tab.

In addition to project tree structures and standard Eclipse buttons (see Table 1 - 6 on page 1-11), the **Projects View** includes tabs and icons:

Table 1 - 7: Projects View tabs and icons

View Element		Description
Tabs	Project	Convertigo view. Contains all projects and related objects. Objects are presented in a tree structure and contained in folders.
	Project Explorer	Standard Eclipse view. Contains all projects and related files. Files are represented in a tree structure.
Icons		*Find* icon. Search project tree for a specific object
		Decrease priority icon. Available only if an extraction rule is selected. Decreases its priority, thus changing extraction rules execution order for this screen class and its children classes.
Icons		*Increase priority* icon. Available only if an extraction rule is selected. Increases its priority, thus changing extraction rules execution order for this screen class and its children classes.
		*Save* icon. Available only when at least one object has been modified. Saves all the project. All objects will be saved, and project xml file will be updated. **Remember to save projects on a regular basis.**
		*Refresh* icon. Available only if a project is selected. Refreshes Projects View by reloading project content from the hard drive. Identical to closing and re-opening project.
		*Import* icon. Imports a new project (in CAR or XML format) in the Studio.

In the **Projects View**, objects are formatted depending on their status:

- plain text - unchanged object;
- bold - object changed and unsaved;

- red - disabled object;
- orange - not executable object (child of a disabled object).

The figure below shows the different types of formatting in the **Projects View**.

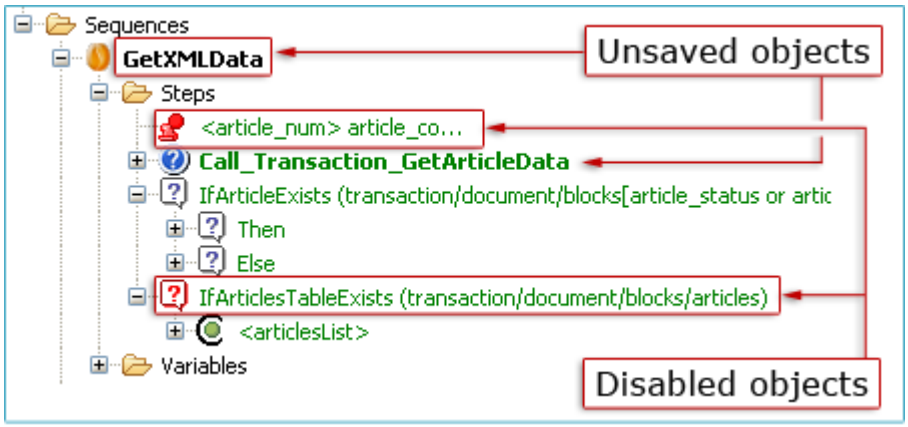


Figure 1 - 9: Formatting of unsaved, inherited and disabled objects in the Projects View

Once an object has been selected in the **Projects View**, its properties appear in the **Properties View**.

Properties View

The **Properties View** displays the properties of the object selected in the **Projects View**.

Properties depend on the type of the selected object. They are presented by categories (*Expert*, *Configuration*, *Information*, *Selection*, *Properties*, etc. - categories can be hidden by clicking on) that can be either expanded or collapsed by clicking respectively on or .

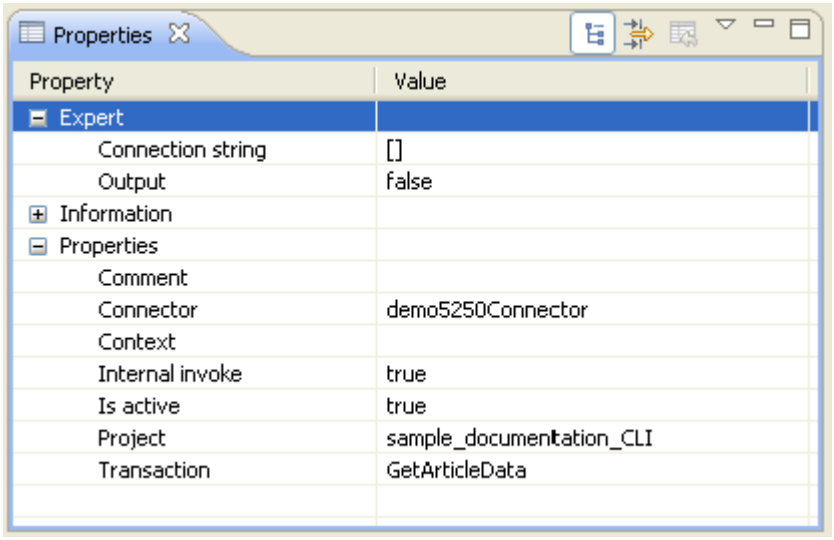


Figure 1 - 10: Properties View

In addition to object properties and standard Eclipse buttons (see Table 1 - 6 on page 1-11), the **Properties View** includes one tab (*Properties*) and icons:

Table 1 - 8: Properties View icons

Icon	Description
	*Show Categories* icon. Displays or hides property categories. When categories are hidden, properties appear in alphabetical order.
	*Show Advanced Properties* icon. Displays or hides advanced properties, if relevant. Not used for Convertigo objects properties.
	*Restore Default Value* icon. Available only when a property has been modified. Restores it to default value. Not used for Convertigo objects properties.

To edit properties

1 Left-click on the property value in the **Properties View**.

- the value is highlighted with white background; edit it by typing a new value in the field:

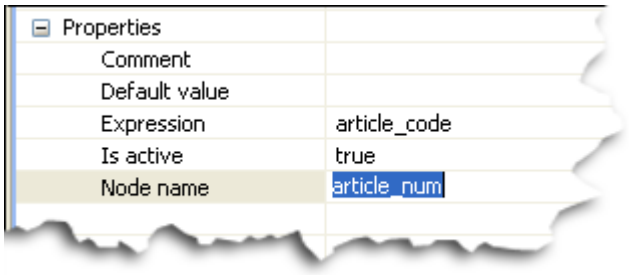


Figure 1 - 11: Text property field edition

- the value is highlighted with clear blue background; edit it by typing a new javascript expression in the field:

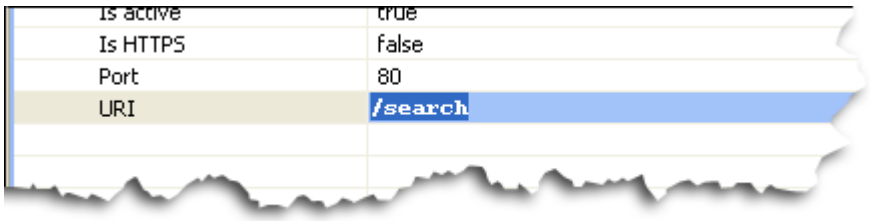


Figure 1 - 12: Javascript expression field edition

- the value is higlighted then a  symbol appears on the right of the field:

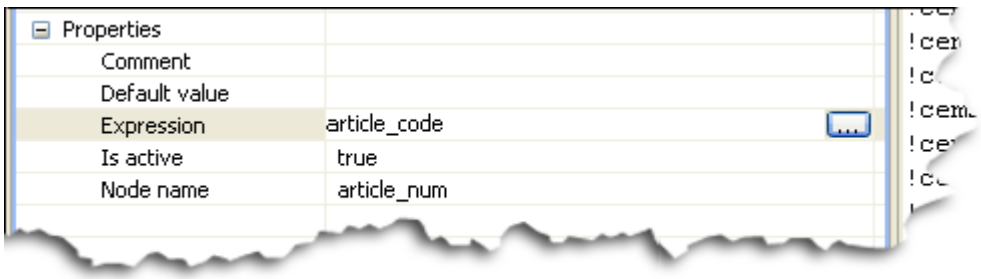


Figure 1 - 13: Text property edition via edition window

Click on  . Depending on the property type:

A. A Javascript expression window or an edition window opens:

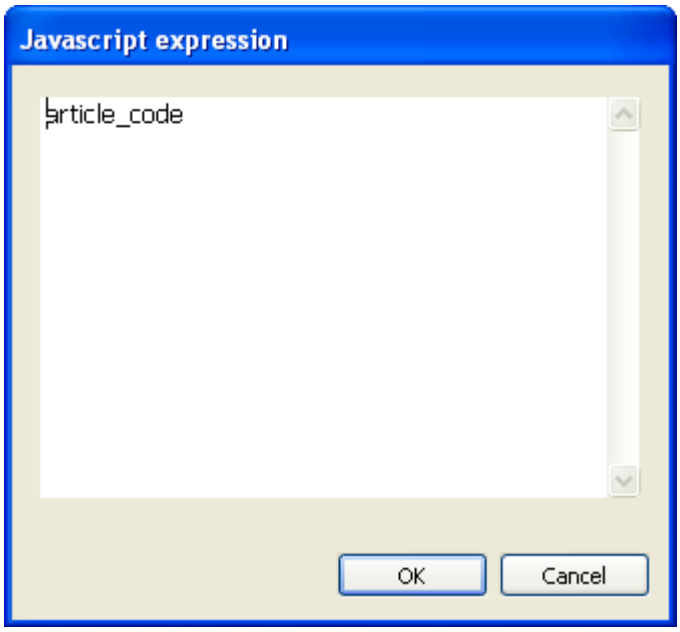


Figure 1 - 14: Javascript expression window

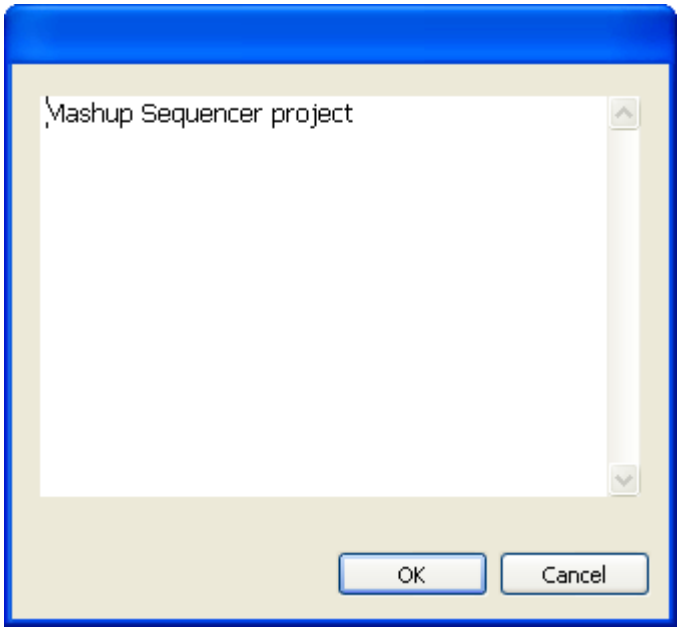


Figure 1 - 15: Edition window

1. Enter the new javascript expression or the text comment in the window.
2. Click on **OK**.

B. A configuration window opens (in this example the **Xml Attributes to include** window for setting attributes in transaction ouput XML):

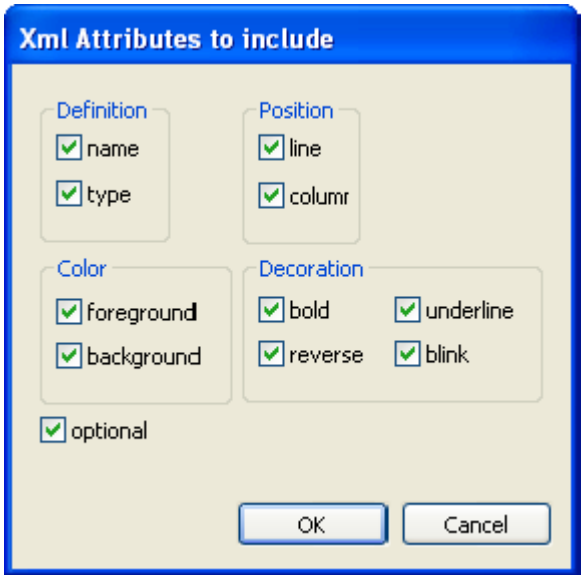


Figure 1 - 16: Configuration window

1 Set the parameters in the different fields and menus.

2 Click on **OK**.

- the value is higlighted and a symbol appears on the right of the field:

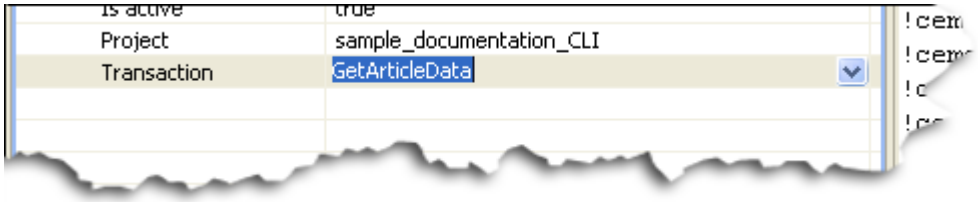


Figure 1 - 17: Text property edition via drop-down menu

1 Click on . A drop-down menu appears:

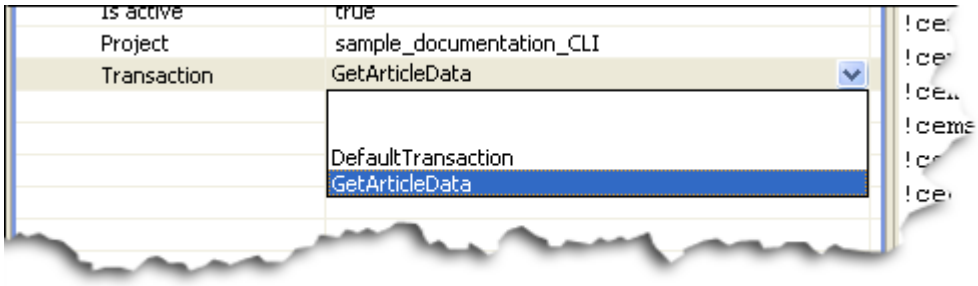


Figure 1 - 18: Selecting a value in the drop-down menu

2 Select a value in the drop-down menu.

3 Press `Enter`.

Step Source Wizard

Step sources are set using a **Step Source** wizard, which opens automatically when clicking on the button on steps source-type properties.

The **Step Source** wizard allows you to:

- select a step,
- access its XML output schema,
- select, within this XML output schema, a given XML node, the value of which will serve as a basis for the actions to be performed by the step.

The figure below describes the **Step Source** wizard:

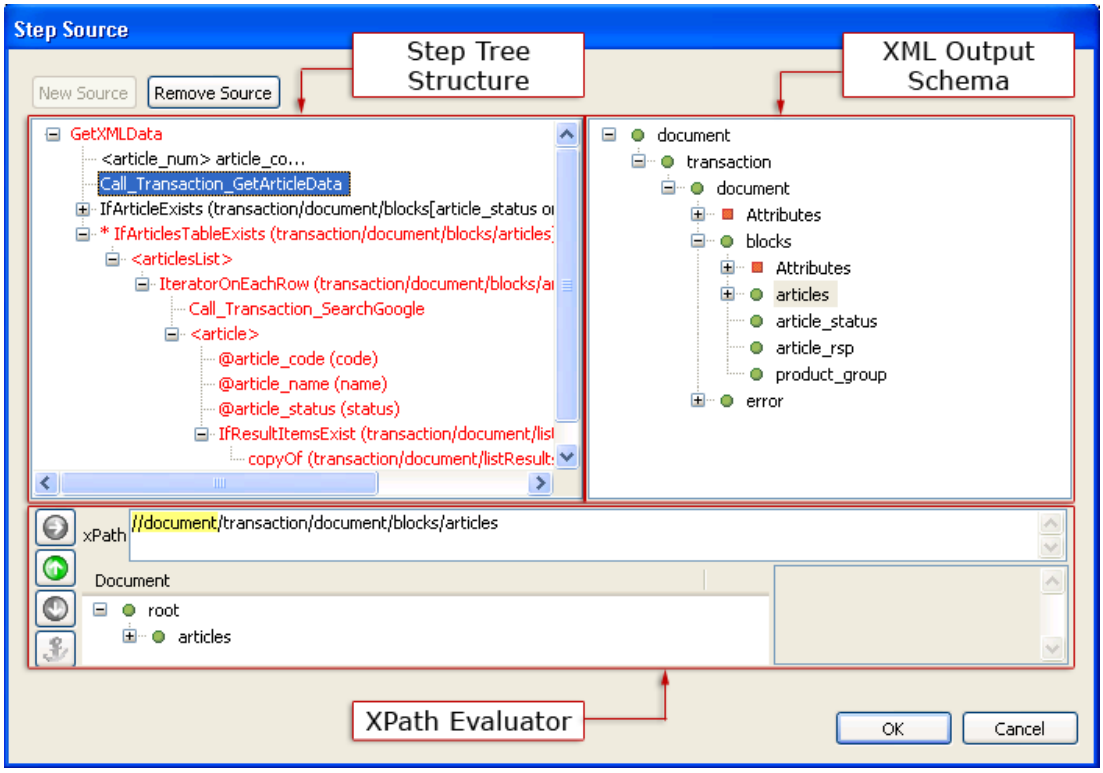


Figure 1 - 19: Step Source wizard

The table below describes the three panes of the **Step Source** wizard**:**

Table 1 - 9: Step Source wizard description

Pane	Description
Steps Tree Structure	Displays the steps of the current sequence. The currently parametered step appears with a * in front of its name. Non reachable steps appear in red (they can not be selected as source in the step currently parametered). This is the case for steps situated after (in terms of sequence order) the step you are configuring the source for.
XML Output Schema	When a step is selected in the **Steps Tree Structure** pane, the XML output structure (Schema) of the step is displayed in this pane.
XPath Evaluator	Displays the XPath expression to nodes selected in the **XML Output Schema** as well as result nodes of the generated XPath execution on the **XML Output Schema**. Contains an **xPath** field, a **Document** tree structure and several icons (see Table *"XPath Evaluator elements"* on page 1-17). You can also manually add predicates to the XPath expression such as `[contains(., 'this text')]` or `[position() > 3]`. For more information on XPaths, see http://www.w3schools.com/XPath/.

The table below describes the fields and icons of the **XPath Evaluator**:

Table 1 - 10: XPath Evaluator elements

XPath Evaluator Elements		Description
Field	xPath	Each time an item is selected in the **XML Output Schema** pane, the source wizard automatically generates the XPath to address the element in this field.
	Document	Displays the result of the generated XPath applied on the schema DOM.
Icons		*Evaluate XPath* icon. Evaluates the XPath expression specified in the *xPath* field on the schema **DOM**. The result is displayed in the *Document* field.
		*Backward XPath History* icon. Displays in the *xPath* field the previous XPath stored in the XPath history.
		*Forward XPath History* icon. Displays in the *xPath* field the next XPath stored in the XPath history.
		*Set Anchor* icon. Applicable after an XPath has been generated for a given node. Fixes this XPath and highlights it in yellow. Used to develop complex XPath from an *anchored* XPath.

When setting up a step property, to define a new source

1 Click on **New Source**.

The **Steps Tree Structure**, **XML Output Schema** and **XPath Evaluator** panes become active.

2 Select the step providing the source data in the **Steps Tree Structure**.

3 Select the source data in the associated **XML Output Schema**.

The XPath to the source is automatically generated.

4 Click **OK**.

This process can be illustrated as follows:

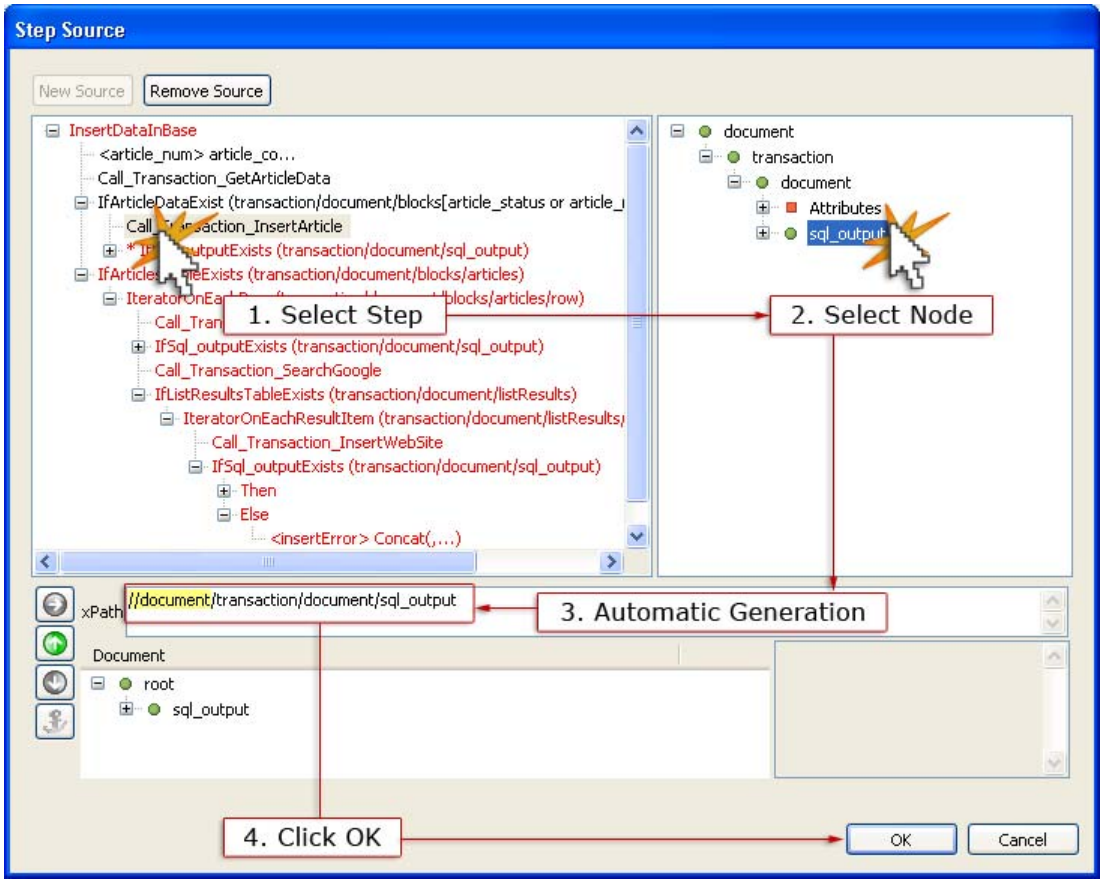


Figure 1 - 20: Setting a step source using the Step Source wizard

Connector / Sequence View

The **Connector** (respectively **Sequence**) **View** displays information specific to the connector (respectively sequence) set for the project.

The table below describes these tabs:

Table 1 - 11: Connector / Sequence View

View	Description
Connector View	A *<ProjectName> <ConnectorName>* tab opens when double-clicking the *<ConnectorName>* connector set for a CMS project. Displays ongoing transactions, the type of which depends on the connector type.
Sequence View	A *<ProjectName> <SequenceName>* tab opens when double-clicking the *<SequenceName>* sequence set for a CMS project. Displays the XML output generated by the sequence depending on steps set as generating XML.

The following sub-sections describe both views.

SQL Connector View

The view described in this sub-section is the SQL connector view; it is specific to the CMS studio.

Different types of connector views are used and mentioned (for example HTML connectors, Javelin connectors) in this CMS Quick Guide, because the sequence project set as part of this Quick Guide (see "Mashup Sequencing Project" on page 2-3) uses various Convertigo projects (CWI, CLI) including project-specific connector types.
For more information on specific connector views, refer to the related Quick Guide.

This view, displayed in a tab called after the name of the project and connector, contains information related to the SQL connector and its transactions. It opens when double-clicking the connector name in the **Projects View**.

The left part (table) of the view displays ongoing transaction result records in real time.

The XML output generated by SQL transactions appears in the **XML** tab in the right part of the view.

In the following figure, several rows have been selected in a table thanks to a transaction. Their content is displayed in the left pane:

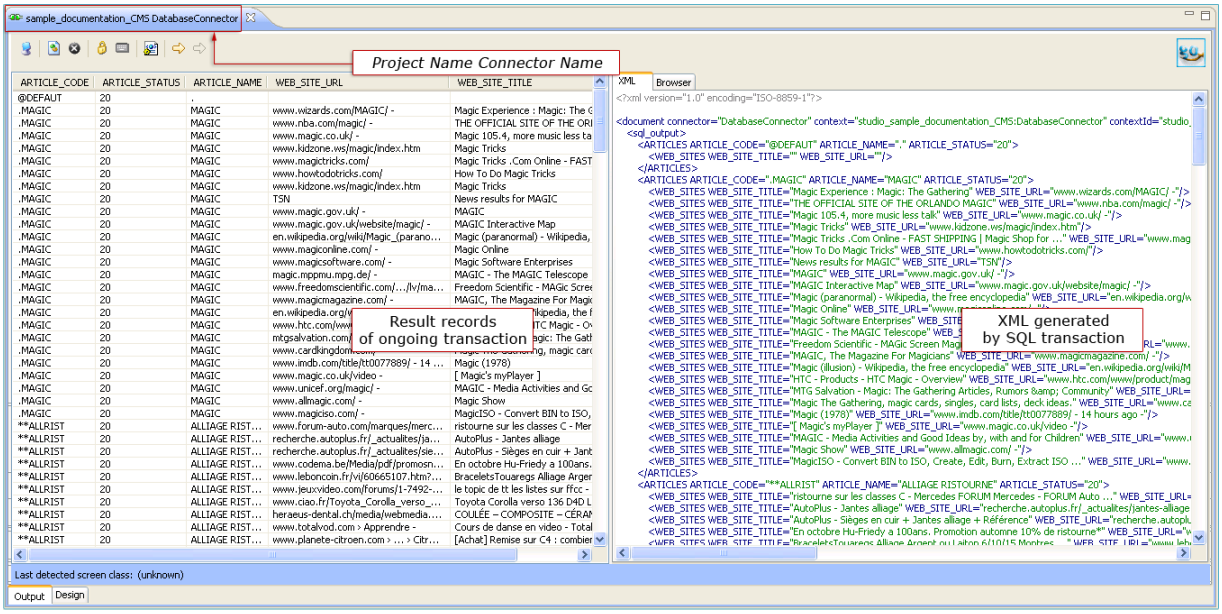


Figure 1 - 21: SQL Connector View

Sequence View

This view, displayed in a tab called after the name of the project and sequence, contains information related to the sequence execution. It opens when double-clicking the sequence in the **Projects View**.

The window in the left part of the view displays in real time information related to processed steps (for example, the transaction output returned to the sequence when a *Call Transaction* step is executed in the sequence).

The XML output generated by the sequence appears in the **XML** tab in the right part of the view.

In the following figure:

- a *Call Transaction* step has been executed - the transaction output returned to the sequence appears in the left part of the tab.
- several steps that generate XML have been executed - the output XML of the sequence is displayed in the right part:

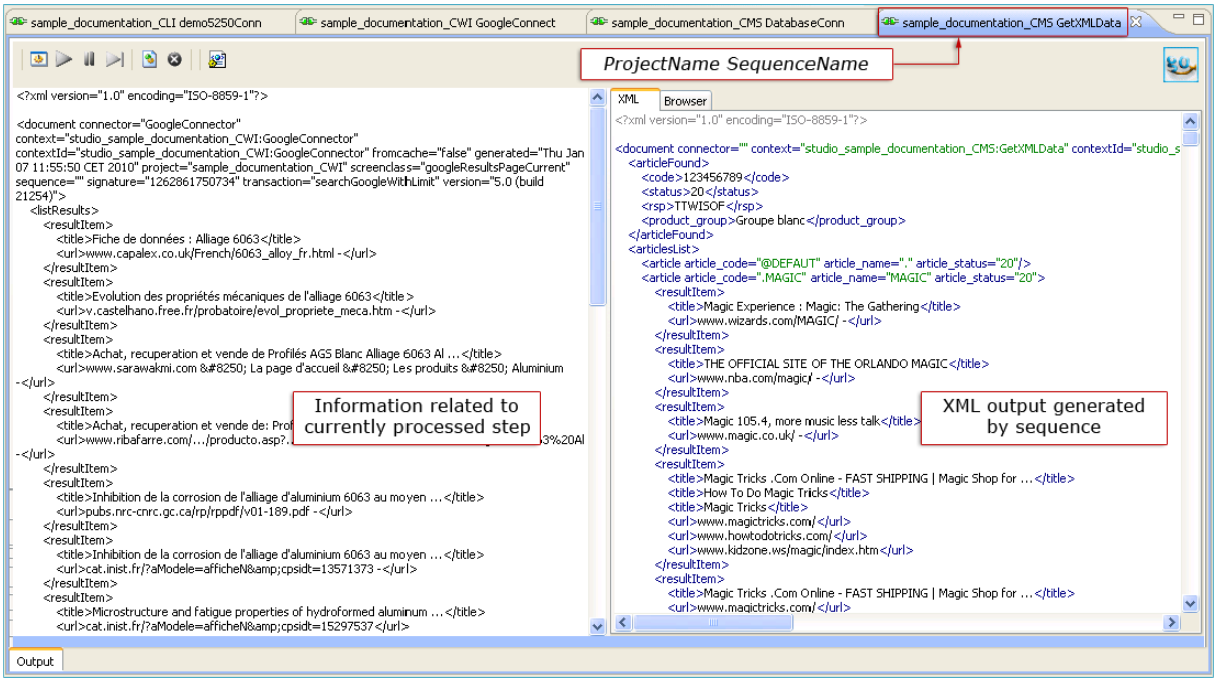


Figure 1 - 22: Sequence View

2 My First Convertigo Mashup Sequencer Project

This chapter describes the project prerequisites then gives instructions on how to set up a simple CMS project.

This first CMS project is divided into two sequences, each of which illustrates one of the two aspects of CMS sequences. Both sequences are developed in a dedicated section, with references from the second sequence to the first when setting procedures appeared redundant.

The procedure for executing a sequence is then described. It is similar whatever the executed sequence. Finally, sequence-specific XML outputs are analyzed, with possible optimization solutions.

2.1 Prerequisites

Before starting on this Convertigo Mashup Sequencer project, please read carefully the following sub-sections and notes:

2.1.1 Key Quick Guide Projects

This Mashup Sequencing project (called `sample_documentation_CMS`) is based on two other *Quick Guide* projects:

- the Convertigo Web Integrator (CWI) `sample_documentation_CWI` project, the

purpose of which is to launch a keyword-based Google search and generate an XML output containing Web page titles and URLs returned by Google.
In the context of the `sample_documentation_CMS`, a `searchGoogleWithLimit` CWI transaction has been created in `sample_documentation_CWI` project. This new CWI transaction is similar to the `searchGoogle` transaction developped in the CWI Quick Guide (refer to the *"Starting with Convertigo Web Integrator" Quick Guide*), with an extra variable (called `maxPages`) setting the number of pages returned by Google. This transaction is called in this CMS project.

- the Convertigo Legacy Integrator (CLI) `sample_documentation_CLI` project, the purpose of which is to connect to an article management legacy application, navigate through legacy screens and generate an XML output containing a number of article information (article code, name, product group, etc.) collected in the process. In the context of the `sample_documentation_CMS`, the `GetArticleData` CLI transaction is called.

It is highly recommended that you review CWI and CLI Quick Guide projects before starting a CMS project.
For more information, refer to the "Starting with Convertigo Web Integrator" and "Starting with Convertigo Legacy Integrator" Quick Guides.

2.1.2 Database Environment

Convertigo Mashup Sequencer includes a built-in HSQL database.

In the context of this *Quick Guide* project, the sample database contains two tables, `articles` and `web_sites`.

The table below describes these tables:

Table 2 - 1: "articles" table

Columns	Type	Description
`article_id`	Unsigned integer	Table **primary key**. Auto-incremented.
`article_code`	Varchar	Article code generated by `GetArticleData` transaction (*table* extraction rule).
`article_statut`	Varchar	Article status generated by `GetArticleData` transaction (*tag name* extraction rule).
`article_product_group`	Varchar	Article product group generated by `GetArticleData` transaction (*tag name* extraction rule).
`article_rsp`	Varchar	Article rsp generated by `GetArticleData` transaction (*tag name* extraction rule).
`article_nom`	Varchar	Article name generated by `GetArticleData` transaction (*table* extraction rule).

Table 2 - 2: "web_sites" table

Columns	Type	Description
`web_site_id`	Unsigned integer	Table **primary key**. Auto-incremented.
`article_code`	Varchar	**Relationship link** with `articles` table.
`web_site_url`	Varchar	Web site URL generated by `searchGoogleWithLimit` transaction.
`web_site_title`	Varchar	Web site title generated by `searchGoogleWithLimit` transaction.

2.2 Mashup Sequencing Project

This section describes the sample Mashup Sequencing project and the two project sequences. It also describes how to open the completed sample project from the Studio:

- Project Description
- First sequence description
- Second sequence description
- Opening the Sample Project from the Studio

2.2.1 Project Description

The purpose of this first Convertigo Mashup Sequencer project is to develop two sequences, each of them illustrating one of the two main purposes of CMS sequences:

1. First sequence description - the first sequence is meant at orchestrating transactions so as to collect data, arrange ("mix" as required by the user) collected data and produce an arranged XML output
2. Second sequence description - the second sequence is meant at orchestrating transactions so as to collect data, arrange ("mix" as required by the user) collected data, insert them in database and produce a *check* XML output - an XML informing the user about possible errors returned by SQL transactions.

2.2.2 First sequence description

The purpose of the first sequence, called `GetXMLData`, is to:

1. *Retrieve* the CLI `GetArticleData` transaction input variable value (`article_no`) from a sequence input variable (`article_code`) thanks to a step creating a node element (`article_num`).
2. *Call* the CLI `GetArticleData` transaction with this input value.

   The `GetArticleData` transaction generates an XML output with:

   - three XML tags containing article data (`article_status`, `article_rsp` and `product_group`) related to the article for which the article code has been sent as input variable;

- a three-column (`code`, `name`, `status`) XML table called `articles` and containing a list of article management information (respectively article code, name and status) independent from the input variable value.

3 *Check* if one of the three article data tag names exists, by looking for specific XML nodes in the `GetArticleData` transaction XML response.

If so, *generate* a parent complex XML element called `<articleFound>` including four child simple XML elements - `<code>`, `<status>`, `<rsp>` and `<product_group>` - defined as follows:

- the `<code>` element contains the value of the `article_code` variable,
- the `<status>`, `<rsp>` and `<product_group>` elements contain the contents of the `article_status`, `article_rsp` and `product_group` XML nodes generated by the previously called `GetArticleData` transaction.

If not, *generate* a parent complex XML element called `<articleNotFound>` including one child simple XML element (`<code>`) containing the value of the `article_code` variable.

4 *Check* if the `articles` XML table exists, by looking for the associated XML node in the `GetArticleData` transaction XML response.

If so, *generate* a parent complex XML element called `<articlesList>`.

5 *Iterate* on each row of the `articles` XML table.

6 For each row of the `articles` XML table, *call* the CWI `searchGoogleWithLimit` transaction by using as transaction input variables:

- the content of the `name` node included in each row of the `articles` XML table for the `keyword` variable
- a fixed value (2) for the `maxPages` variable.

7 For each row of the `articles` XML table, *generate* a complex `<article>` XML element with three attributes, the value of which is sourced from the `code`, `name` and `status` XML nodes of the row being iterated.

8 *Check* the presence of `resultItem` nodes in each XML output returned by the CWI `searchGoogleWithLimit` transaction (called for each row).

If present, *copy* the content of `resultItem` nodes together with their child nodes into the sequence XML output, as child nodes of the complex `<article>` element.

In terms of *steps* (Convertigo objects), the previous points can be represented as follows:

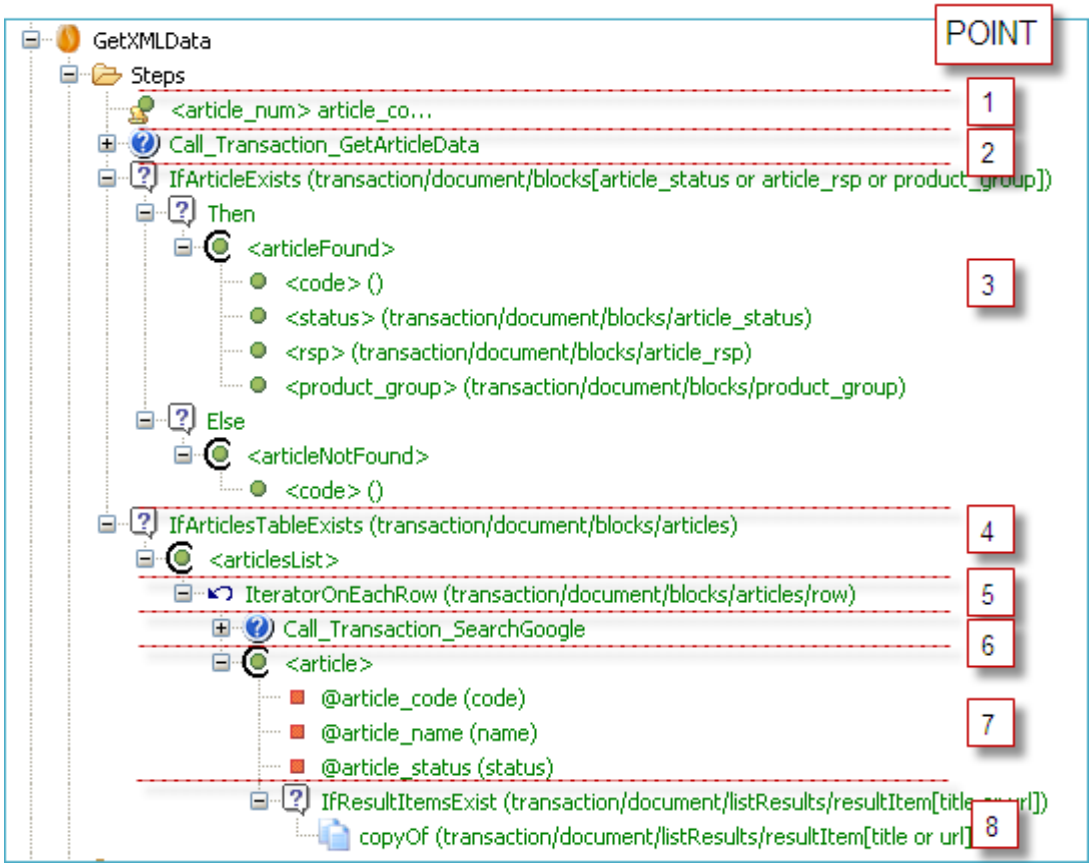


Figure 2 - 1: Steps tree structure of the first sequence

2.2.3 Second sequence description

The purpose of the second sequence, called `InsertDataInBase`, is to:

1 *Retrieve* the CLI `GetArticleData` transaction input variable value (`article_no`) from a sequence input variable (`article_code`) thanks to a step creating a node element (`article_num`).

2 *Call* the CLI `GetArticleData` transaction.

The `GetArticleData` transaction generates an XML output with:

- three XML tags containing article data (`article_status`, `article_rsp` and `product_group`) related to the article for which the article code has been sent as input variable;
- a three-column (`code`, `name`, `status`) XML table called `articles` and containing a list of article management information (respectively article code, name and status) independent from the input variable value.

3 *Check* if one of the three article data tag names exists, by looking for specific XML nodes in the `GetArticleData` transaction XML response.

If so, *populate* the `articles` database table with the content of the `article_status`, `article_rsp` and `product_group` nodes of the `GetArticleData` transaction XML output, by calling an SQL transaction called `InsertArticle`.

4 *Check* if the SQL transaction has been correctly executed, by looking for a specific

XML node (<sql_output>) in the InsertArticle transaction XML output.

5 If so, *generate* an <insertOk> tag name in the sequence XML output to keep a trace of the correct execution of the SQL transaction.
If not, *generate* an <insertError> tag name in the sequence XML output to keep a trace of SQL transaction errors.

The previous two steps are implemented as part of the sequence optimization, not as part of the normal sequence setting. For more information, see "Adding Steps to Check Database Transactions" on page 2-162.

6 *Check* if the articles XML table exists in the GetArticleData transaction response, by looking for the associated XML node in the XML output.

7 If so, *iterate* on each row and, for each row, populate the articles database table with the content of row nodes (including the code, name and status child nodes). The row node is child to the articles XML table belonging to the GetArticleData transaction XML output.
As described in step 3, the database table populating is performed by calling an SQL transaction called InsertArticle.

8 *Check* if the SQL transaction has been correctly executed, by looking for a specific XML node (<sql_output>) in the InsertArticle transaction XML output.

9 If so, *generate* an <insertOk> tag name in the sequence XML output to keep a trace of the correct execution of the SQL transaction.
If not, *generate* an <insertError> tag name in the sequence XML output to keep a trace of SQL transaction errors.

10 Then, for each row, *call* the CWI searchGoogleWithLimit transaction by using the name node value (of iterated row) as Google search input keyword, and a fixed value (2) for the maxPages variable.

11 *Check* if results are returned by Google, by looking for specific XML nodes in the searchGoogleWithLimit transaction XML response (resultItem node).

12 If so, *iterate* on each result item and, for each result item, populate the web_sites database table by calling an SQL transaction called InsertWebSite, with a mix of data from both transactions results: code node, child of row ; and url and title, children of resultItem - the row and resultItem nodes belong respectively to the GetArticleData and searchGoogleWithLimit transaction XML outputs.

13 *Check* if the SQL transaction has been correctly executed, by looking for a specific XML node (<sql_output>) in the InsertWebSite transaction XML output.

14 If so, *generate* an <insertOk> tag name in the sequence XML output to keep a trace of the correct execution of the SQL transaction.
If not, *generate* an <insertError> tag name in the sequence XML output to keep a trace of SQL transaction errors.

The previous two steps are implemented as part of the sequence optimization, not as part of the normal sequence setting. For more information, see "Adding Steps to Check Database Transactions" on page 2-162.

In terms of *steps* (Convertigo objects), the previous points can be represented as follows:

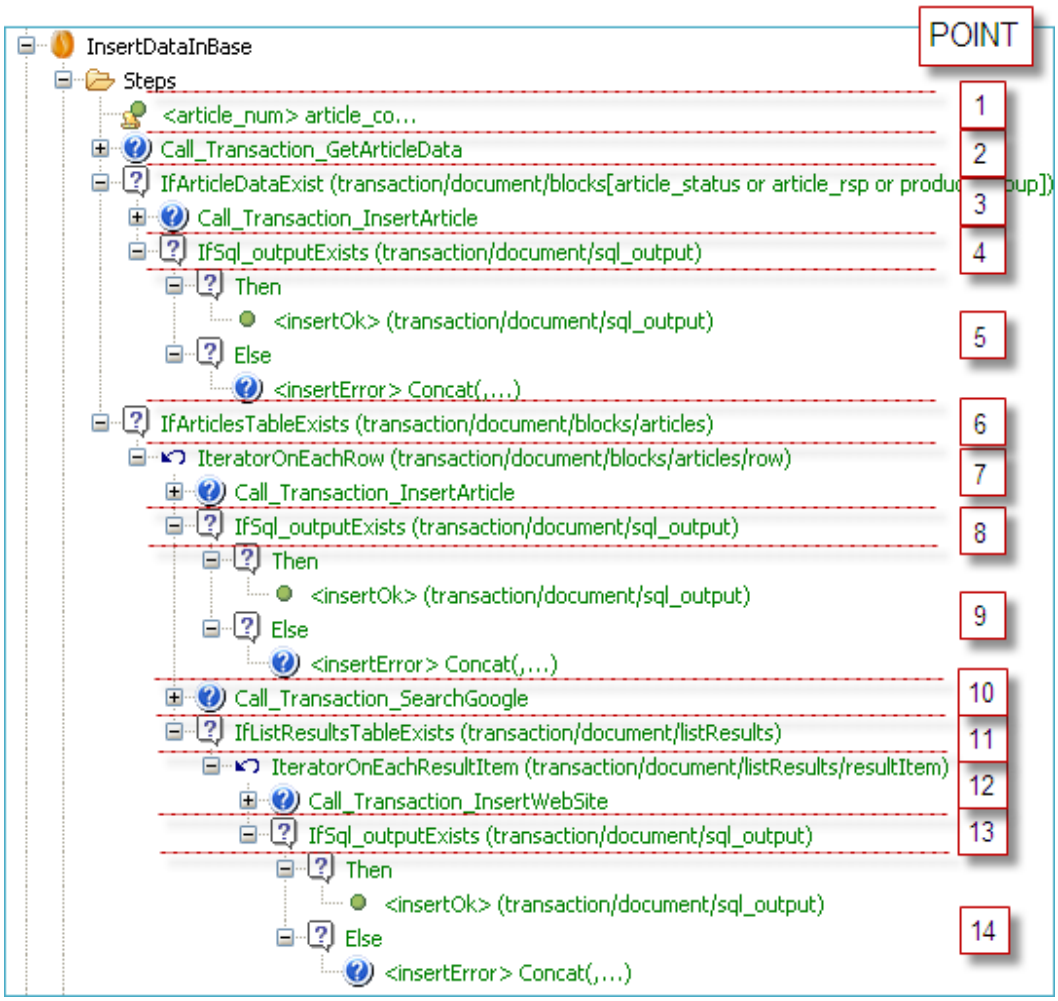


Figure 2 - 2: Steps tree structure of the second sequence

2.2.4 Opening the Sample Project from the Studio

The completed sample project is stored in the application installation folder.

The following procedure describes how to access it from the Studio using the **New Project** wizard.

To open the sample project from the Studio

The CMS sample project uses calls to transactions defined in two other sample projects: `sample_documentation_CWI` (Convertigo Web Integrator sample project) and `sample_documentation_CLI` (Convertigo Legacy Integrator sample project).

We will therefore open those two sample projects before opening the sample CMS project.

1 In the **Studio** toolbar, click on ***New > Project***:

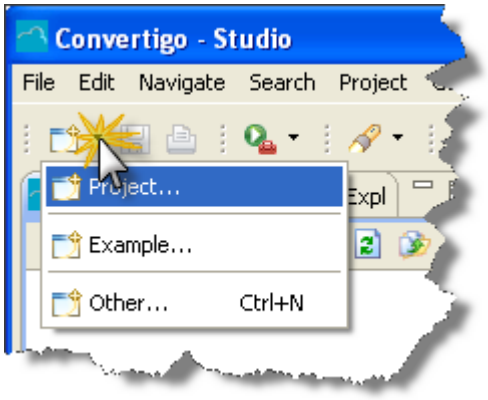


Figure 2 - 3: Launching the New Project wizard

A **New Project** wizard opens.

2 Expand **Convertigo Projects**, then **Documentation Samples**, and select **Legacy integrator**:

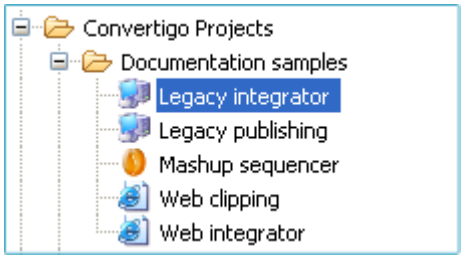


Figure 2 - 4: Selecting the CLI sample project

3 Click on **Next**, then **Finish**.

The sample project opens in the **Projects View**:

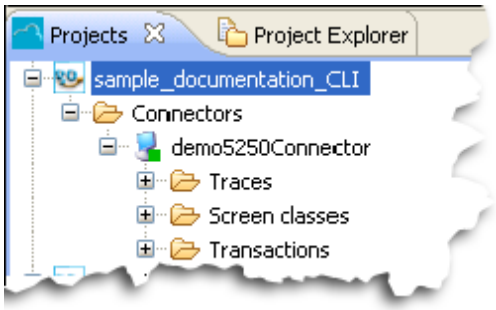


Figure 2 - 5: Sample project in Projects View

4 Repeat points 1 to 3 of this procedure to open the CWI required project (select **Web integrator** sample project in point 2).

After this additional sample project has been opened, both projects appear in the **Projects Folder**:

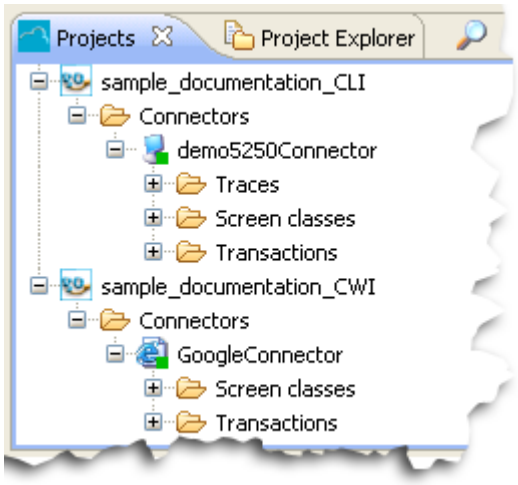


Figure 2 - 6: CLI and CWI sample projects in Projects Folder

5 Repeat points 1 to 3 of this procedure to open the sample CMS project (select **Mashup Sequencer** sample project in point 2).

After the sample project has been opened, the three projects appear in the **Projects Folder**:

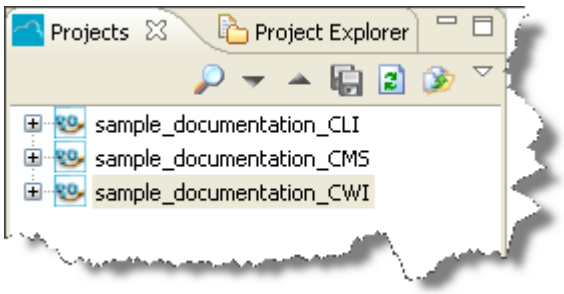


Figure 2 - 7: CLI, CWI and CMS sample projects in Projects Folder

You can now start setting up your own CMS project, `sample_doc_CMS`.

2.3 Setting Up the Project

The following sections explain step by step how to set up your first Mashup Sequencing project and how to start creating a sequence:

- Creating a Project and Setting Connector Parameters
- Creating a Sequence

2.3.1 Creating a Project and Setting Connector Parameters

This section shows how to create a project and set connector parameters.

In this example:

- the project is called `sample_doc_CMS`;
- the connector is an SQL database connector called `DatabaseConnector` accessing your Convertigo sample database environment.

To create a project and set a connector

1 Launch the **Convertigo Studio**;

2 Select ***File > New > Project*** click on  in the toolbar then select ***Project***.

A **New Project** wizard opens:

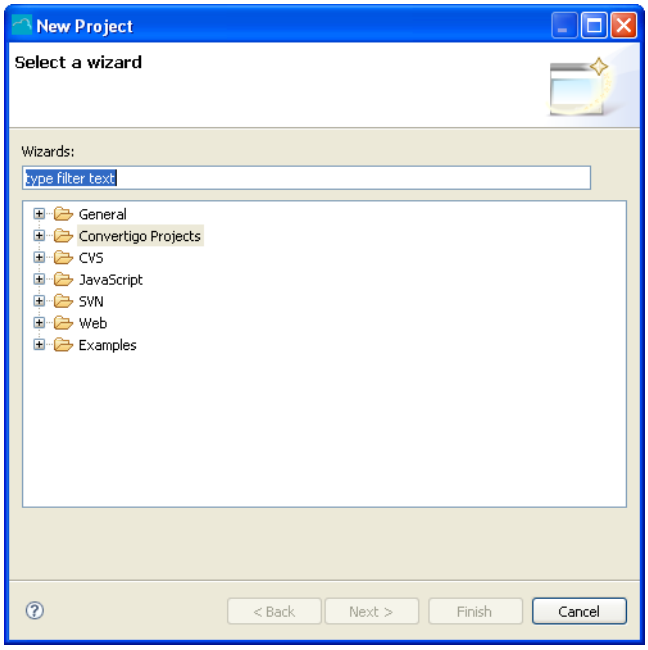


Figure 2 - 8: New Project Wizard

3 Expand **Convertigo Projects**, then **Sequencer** and select **Sequencer Project**:

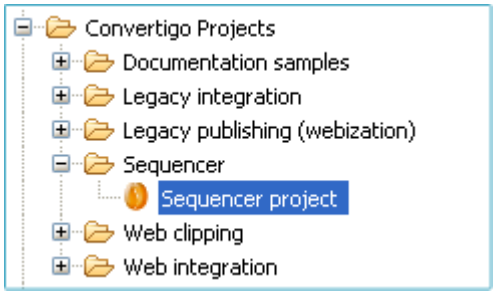


Figure 2 - 9: Selecting a project type

4 Click on **Next**:

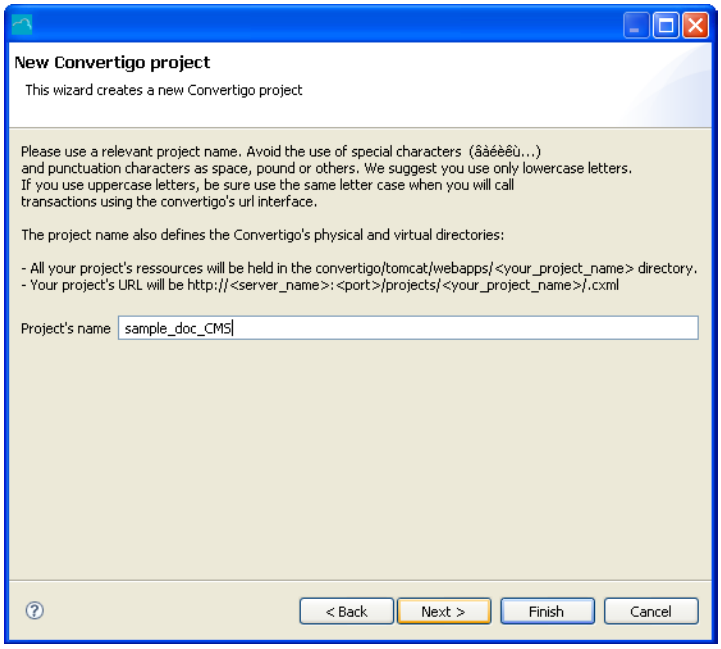


Figure 2 - 10: Entering a project name

5 In the **Project's name** field, type in the project name (for example `sample_doc_CMS`).

6 Click on **Next**:

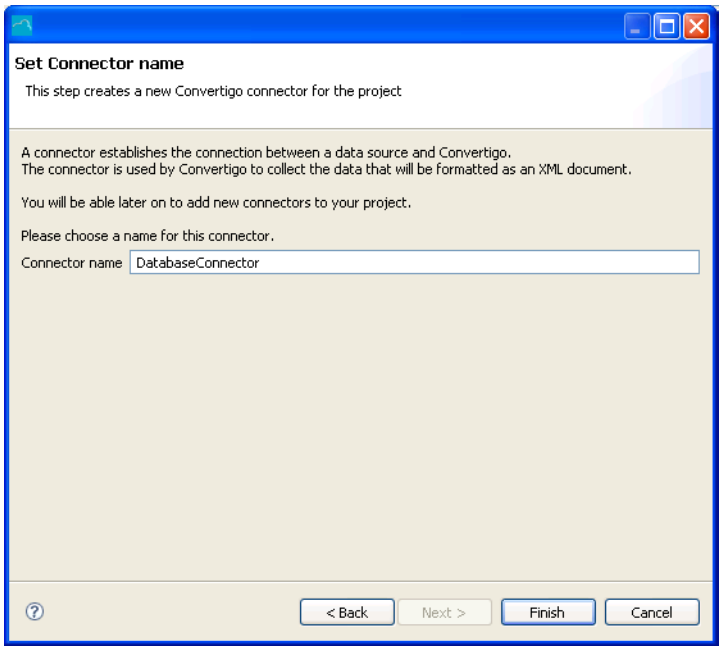


Figure 2 - 11: Entering a connector name

The **Connector name** field is filled by default with the connector name `<ProjectName>Connector`. You can choose another name (for example `DatabaseConnector`);

7 Click on **Finish**.

The project is automatically created together with an SQL connector and an empty sequence (`New_sequence`).

The SQL connector is automatically created together with a default transaction

(`defaultTransactionToBeConfigured`).

These objects appear in their respective folders (*Connectors, Transactions* and *Sequences* folders - see *"Transaction"* on page 1-5 and see *"Sequence"* on page 1-6), in the **Projects View**:

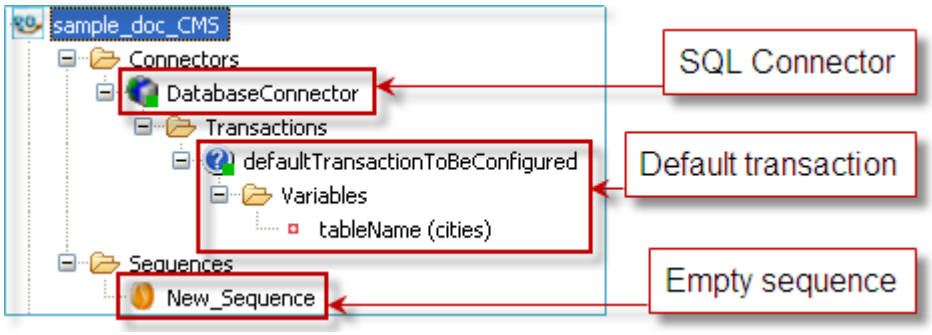


Figure 2 - 12: sample_doc_CMS project appearing in the Projects View

We now need to set the SQL connector parameters.

8 In the **Projects View**, select the connector with a left-click.

The **Properties View** is automatically updated:

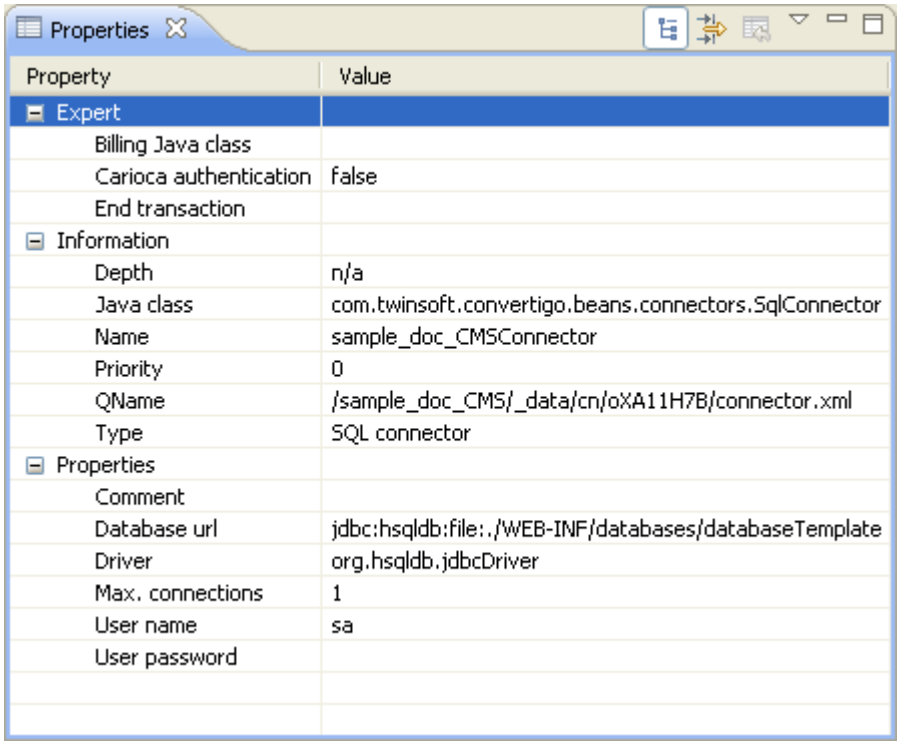


Figure 2 - 13: SQL connector properties

9 Click in the **Value** column of the **Database url** property.

The value is highlighted.

10 Change the value of the database schema name from `databaseTemplate` to `CMS_doc_database`.

The default database for new projects and the already configured database for this sample are in the same location.

The new database URL should look like:
`jdbc:hsqldb:file:./WEB-INF/databases/CMS_doc_database`

All SQL connector parameters can be tuned up depending on specific needs and database settings. For more information on SQL connector parameters, see Table "SQL connector parameters" on page 1-5.

11 Press `Enter`.

12 Save your project by clicking on  or by pressing `Ctrl + S`.

WHAT COMES NEXT?

The following steps consist in creating an empty sequence, then in setting up needed:

- *steps,*
- SQL *transactions* (second sequence only).

Theoretically, users can hardly anticipate which objects (steps, transactions, etc.) will be needed in a sequence. All SQL transactions can therefore be written in anticipation as per specifications. But in practice, another approach can be used, in which transactions are set progressively as they are needed in the sequence.

The two example sequences set up in this *Quick Guide* are described according to this second approach: objects are defined one after another as they are needed in the sequence.

2.3.2 Creating a Sequence

This section shows how to create an empty sequence and set input variables.

The example chosen in the following procedures illustrate the creation of the first sequence, `GetXMLData`. To create the second sequence (`InsertDataInBase`), follow the same procedures.

To create a sequence

1 Right-click on the project or on the **Sequences** folder.

A contextual menu appears:

Figure 2 - 14: Creating a new sequence

2 Select ***New > Sequence***.

A **New Sequence** wizard opens automatically.

3 Select **Generic Sequence**:

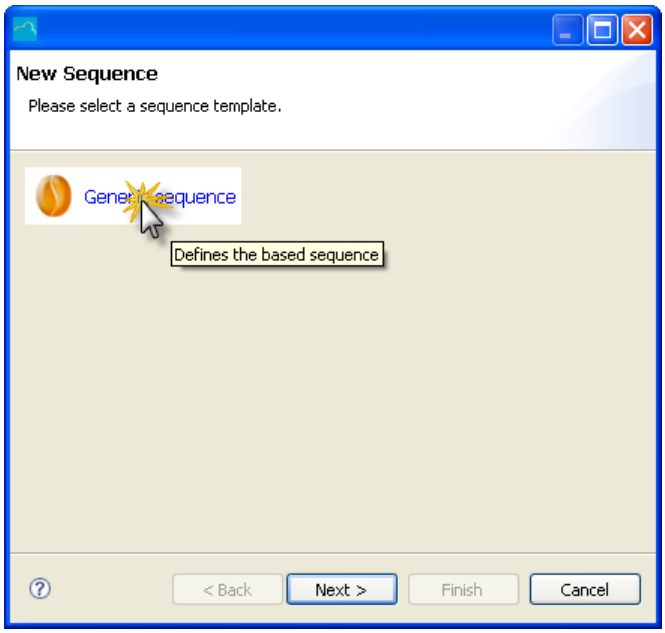


Figure 2 - 15: New Sequence wizard

4 Click on **Next**:

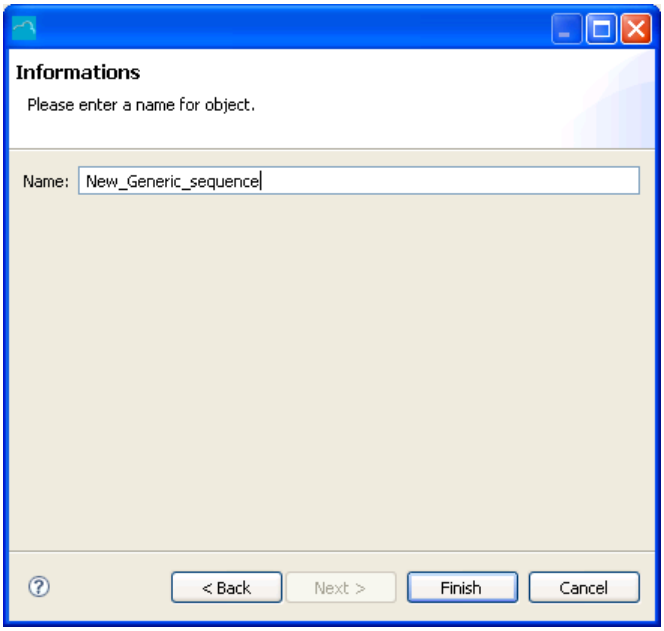


Figure 2 - 16: Giving a name to the new sequence

5 In the **Name** field, type in the name of the new sequence (for example `GetXMLData`).

6 Click on **Finish**.

7 Save your project by clicking on  or by pressing `Ctrl + S`.

Remember to save on a regular basis.

The new sequence appears in the project **Sequences** folder in the **Projects View**:

Figure 2 - 17: New sequence in project Sequences folder

The `GetArticleData` CLI transaction, which will be called in both sequences, takes as input variable an article code (called `article_no` in the CLI project):

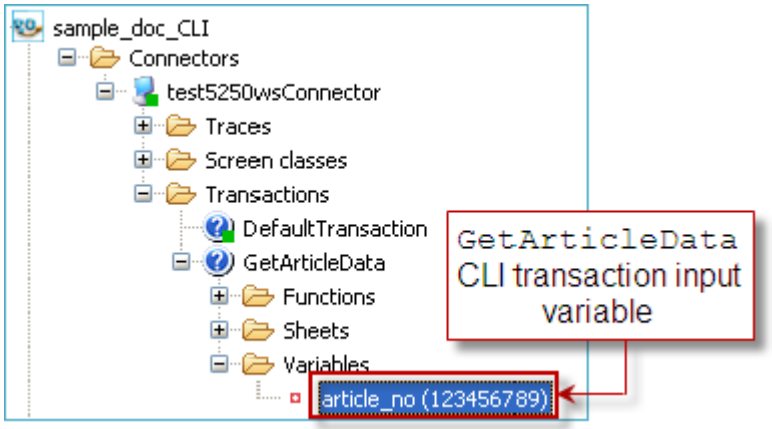


Figure 2 - 18: GetArticleData input variable

To this end, we will associate the same variable with the sequence, since the sequence is based on the `GetArticleData` transaction and calls the `googleSearch` transaction only as a way to enrich the sequence output.

Both sequences therefore need an input variable matching the `GetArticleData` transaction input variable.

To create a new sequence variable

1 In the **Projects View**, right-click on the sequence.

A contextual menu appears:

Figure 2 - 19: Creating a new variable

2 Select ***New > Variable***

A **New Variable** wizard is automatically launched.

3 Select **Variable**.

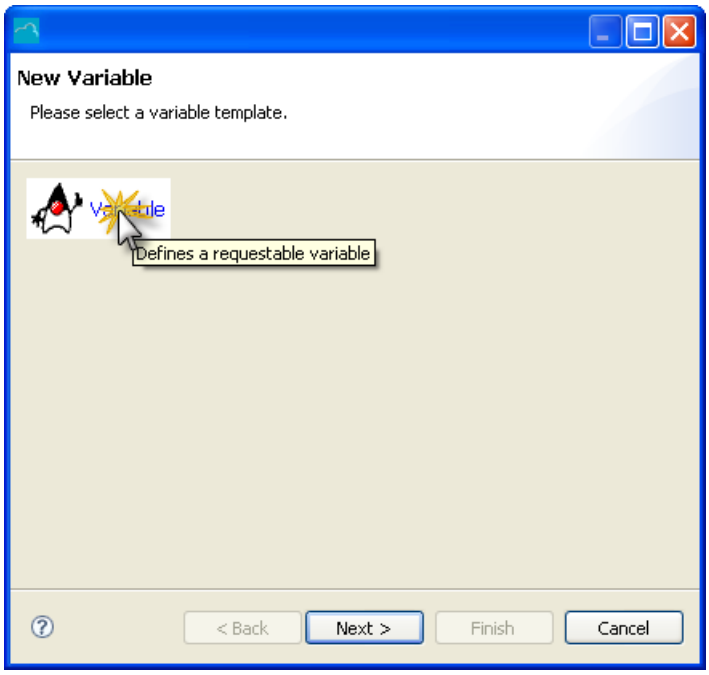


Figure 2 - 20: New Variable wizard

4 Click on **Next**;

5 In the **Name** field that appears, type in the name of the variable (for example `article_code`).

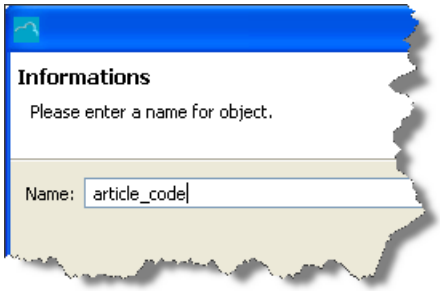


Figure 2 - 21: Giving a name to the new variable

6 Click on **Finish**.

The new variable appears in the sequence **Variables** folder in the **Projects View**:

Figure 2 - 22: New variable in sequence Variables folder

7 Save your project by clicking on  or by pressing `Ctrl + S`.

The sequence variable is now created and saved. We can now set its properties.

*Only the **Description** property is set here as an example. For more information on variable properties, see "Variable" on page 1-8).*

8 Select the `article_code` variable in the **Projects View**.

The **Properties View** is automatically updated:

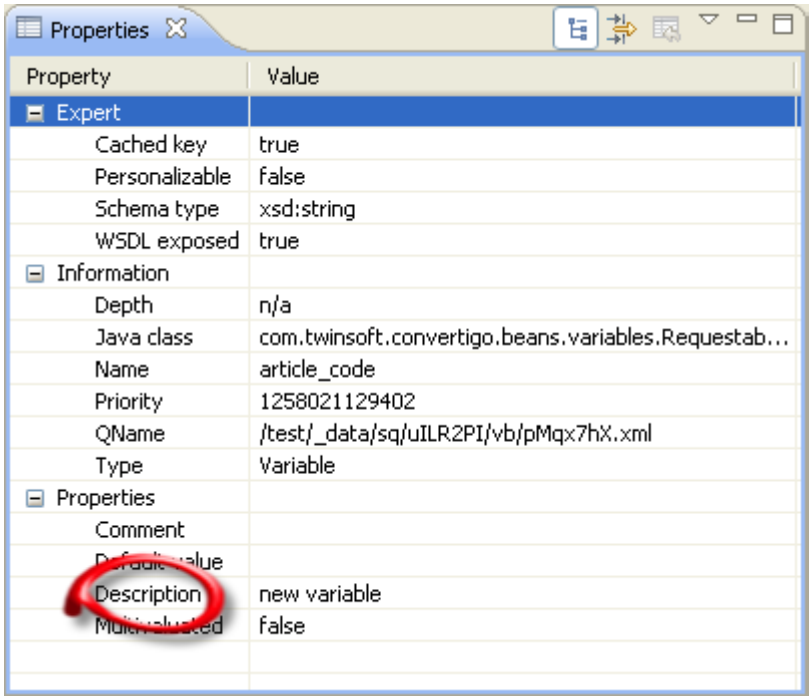


Figure 2 - 23: Variable properties

9 Click in the Value column of the **Description** property.

The property default value (`new variable`) is highlighted.

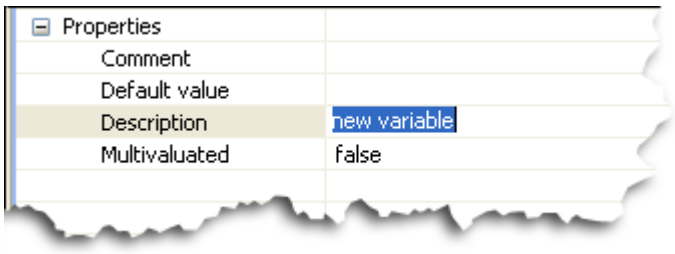


Figure 2 - 24: Default value variable property

10 Type in a description for the variable (for example `article code`).

In the context of this Quick Guide project, all other values are left as default values.

11 Press `Enter`.

The value property is updated.

12 Save your project by clicking on  or by pressing `Ctrl + S`.

The variable is now created and properly set.

WHAT COMES NEXT?

The `GetArticleData` transaction, which represents the starting point of the project, can now be called.

The following step consists in developing the sequence by creating and setting all necessary steps and SQL transactions:

- To see how to develop the first sequence, see *"Developing the First Sequence"* on page 2-18,
- To see how to develop the second sequence, see *"Developing the Second Sequence"* on page 2-80.

2.4 Developing the First Sequence

This section describes how to develop the first sequence, `GetXMLData` or, in other words, how to set all steps needed in the sequence.

According to the sequence description, one of the first steps of the sequence is to call the `GetArticleData` CLI transaction, in order to use its generated XML as source for further steps.

The following procedure shows how to create and set a *Call Transaction* step.

To create and set a Call transaction step

1 In the **Projects View**, right-click on the sequence.

A contextual menu appears.

2 Select ***New > Step***:

Figure 2 - 25: Creating a new step

A **New Step** wizard is automatically launched:

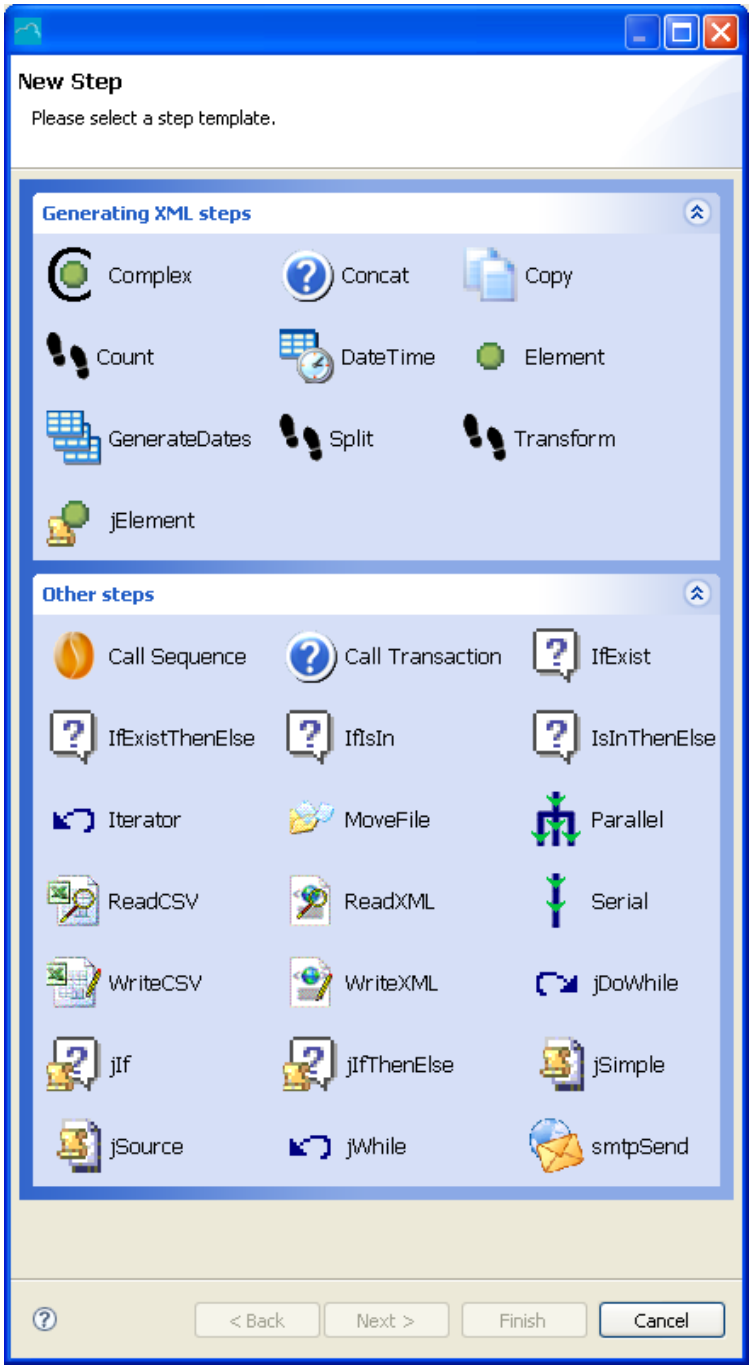


Figure 2 - 26: New Step Wizard

The **New Step** wizard is divided into:

- Steps generating output XML code, which *generate XML document parts* - these steps are used to produce a mashuped sequence XML output.
- Steps not generating output XML, which *process actions* - these steps are used to encode the sequence behaviour.

A *Call Transaction* step is not meant to generate XML ouput code.

3 In the **Other Steps** area of the window, select **Call Transaction** with a left-click.

4 Click on **Next**.

5 In the **Name** field that appears, type in the step name (for example `Call_Transaction_GetArticleData`):

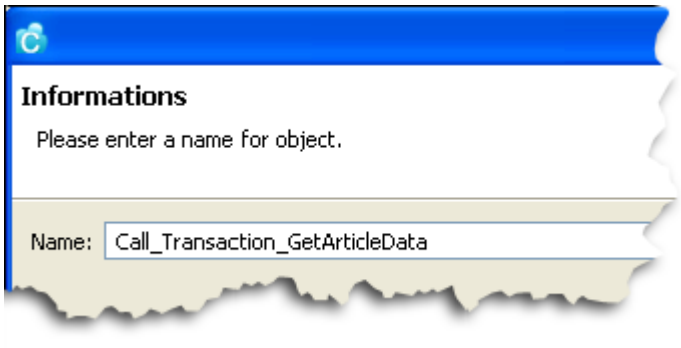


Figure 2 - 27: Giving a name to the step

6 Click on **Finish**.

The new step appears in the **Steps** folder of the project:

Figure 2 - 28: New Call Transaction step in Steps folder

Now that the *Call Transaction* step has been created, we will set its three main interrelated parameters: **Project**, **Connector** and **Transaction**.

The table below describes these parameters:

Table 2 - 3: Call Transaction step parameters

Parameter	Description
Project	Name of the project containing the target (called) transaction. The target project must be open. Set using a combo box listing all available projects in the Studio.
Connector	Name of the connector within the target project. Set using a combo box listing all available connectors defined for the project set as value of the **Project** property.
Transaction	Name of the target transaction. Set using a combo box listing all available transactions defined for the connector set as value of the **Connector** property.

7 In the **Projects View**, select the step.

The **Properties View** is automatically updated:

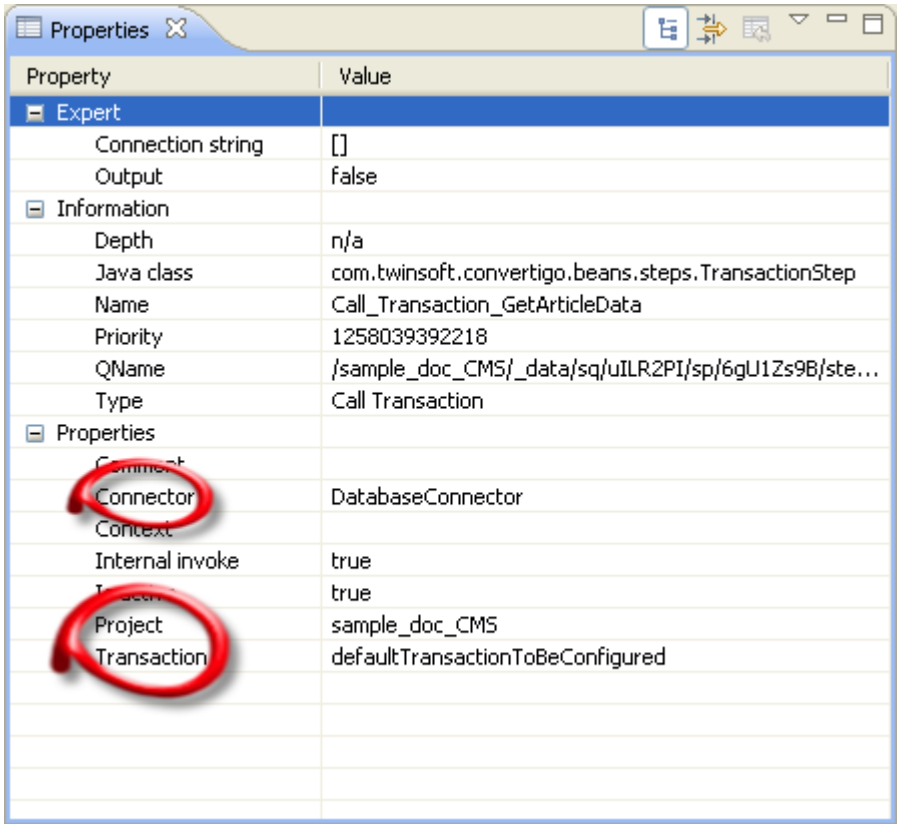


Figure 2 - 29: Call Transaction step properties

8 Click in the **Value** column of the **Project** property.

The current value (`sample_doc_CMS`) is highlighted and a drop-down symbol appears.

9 Click on .

10 Select the project containing the transaction to be called (for example `sample_documentation_CLI` for the `GetArticleData` transaction):

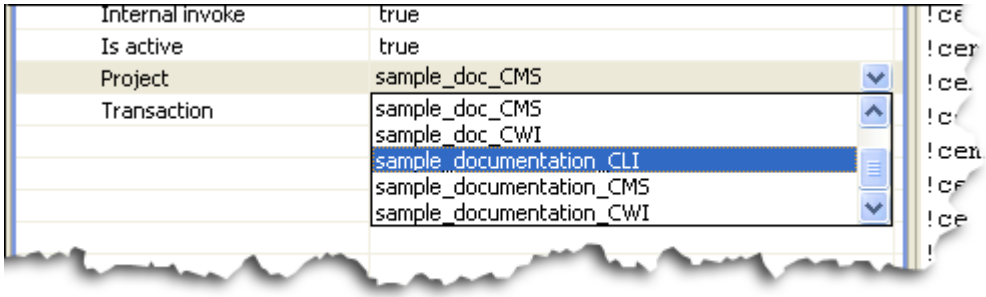


Figure 2 - 30: Setting the Project property of the Call Transaction step

11 Press `Enter`.

The values of the **Connector** and **Transaction** properties are updated.

12 Click in the **Value** column of the **Connector** property.

The current empty value is highlighted and a drop-down symbol appears.

13 Click on .

14 Select the connector containing the transaction to be called (for example, `demo5250Connector` for the `GetArticleData` transaction):

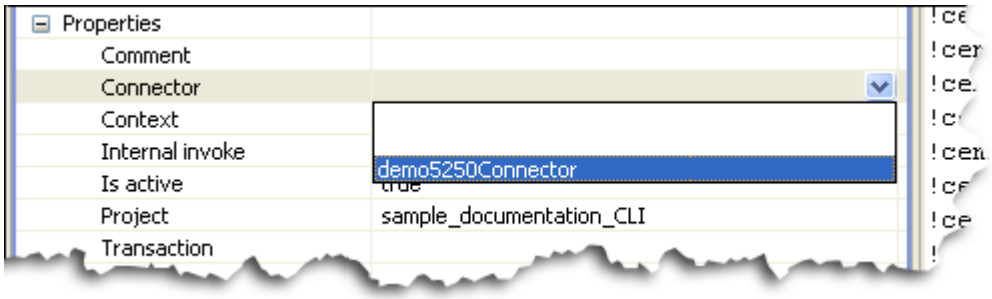


Figure 2 - 31: Setting the Connector property of the Call Transaction step

15 Press `Enter`.

The value of the **Transaction** property is updated.

16 Click in the **Value** column of the **Transaction** property.

The current empty value is highlighted and a drop-down symbol appears.

17 Click on .

18 Select the transaction (for example `GetArticleData`):

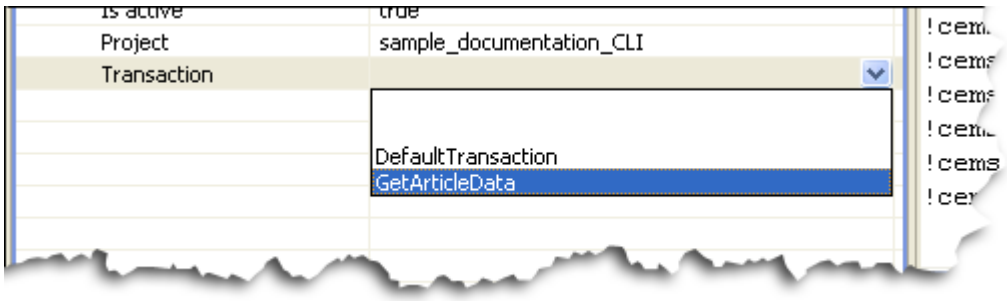


Figure 2 - 32: Setting the Transaction property of the Call Transaction step

19 Press `Enter`.

20 Save your project by clicking on or by pressing `Ctrl + S`.

The step is now set to call the `GetArticleData` target transaction. Since the transaction expects an input variable, we will now import target transaction variables at step level.

To import variables from a target transaction at step level

1 Right-click on the step.

A contextual menu appears:

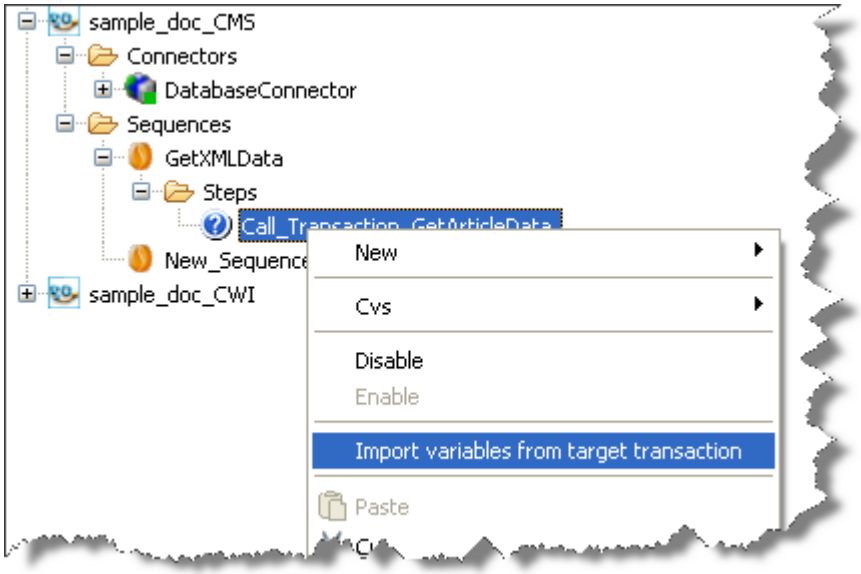


Figure 2 - 33: Import variables from target transaction

2 Select **Import variables from target transaction**.

Once variables have been imported from the target transaction:

- the step appears as "changed and unsaved" (green bolded characters, see Figure 1 - 9),
- variables appear in a **Variables** sub-folder (together with the default value imported from the target transaction between brackets, when applicable):

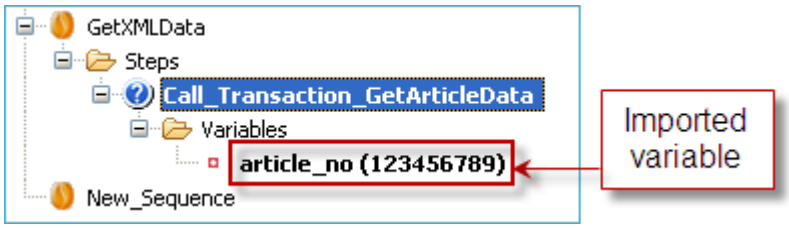


Figure 2 - 34: Variable imported from target transaction (and default value)

This variable must be given a value when calling the transaction.

The `GetXMLData` sequence was defined with an `article_code` variable (see *"To create a new sequence variable"* on page 2-15). When the sequence is executed, the variable receives its value from the call and is added to the sequence JavaScript scope with its value.

The imported variable used by the transaction represents the same data with a different name (`article_no`, see Figure 2 - 34), so we must now inform the sequence that the value to be sent to the transaction call is its `article_code` input variable value.

In other words, the `article_code` sequence variable must be modellized in a step serving as source (see *"Step"* on page 1 - 7) for collecting the `article_no` transaction variable value in the *Call Transaction* step.

To this end, we will:

1 *Create* a new step of *jElement* type - a *jElement* step defines an XML element based on a scope variable. In our case, the *jElement* step is defined as follows:

- this step generates XML and the generated XML node name is `<code>`;
- the Javascript expression of the step is `article_code` (expression of the variable sequence).

2 *Use* this step as a source for the `Call_Transaction_GetArticleData` input variable (`article_no`). This source can be viewed as a pointer pointing via the definition of a suitable XPath towards the XML generated by the previous step.

If both variables had the same name, the following step would be useless.

To create and set a jElement step serving as source for a step variable

1 Right-click on the sequence.

A contextual menu appears:

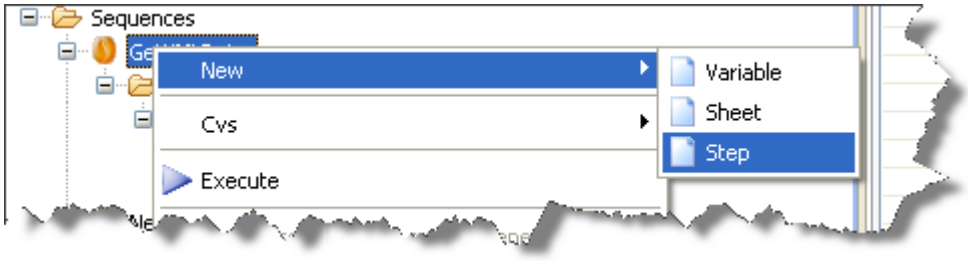


Figure 2 - 35: New step from sequence

2 Select ***New > Step***.

The **New Step** wizard is automatically launched.

3 In the **Generating XML steps** area, select **jElement**:

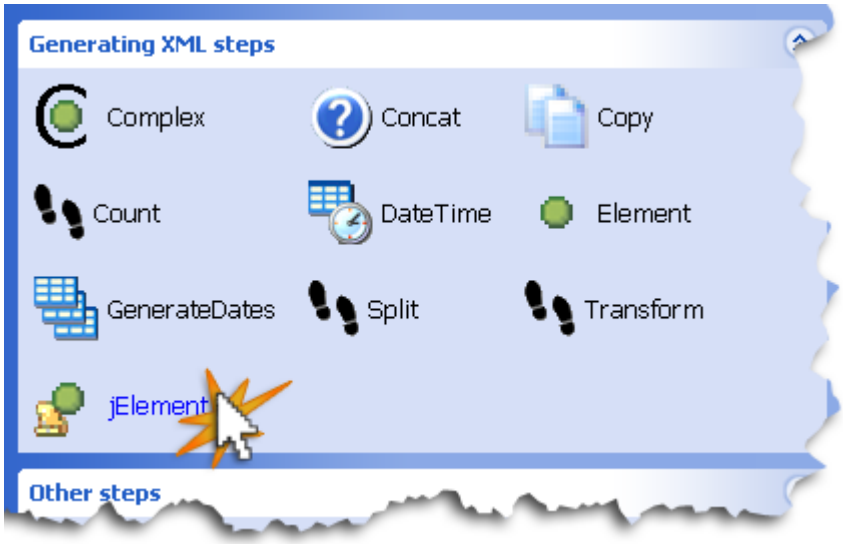


Figure 2 - 36: Selecting a jElement step

4 Click on **Next**.

5 In the **Name** field that appears, type in the *jElement* name (for example `article_num`):

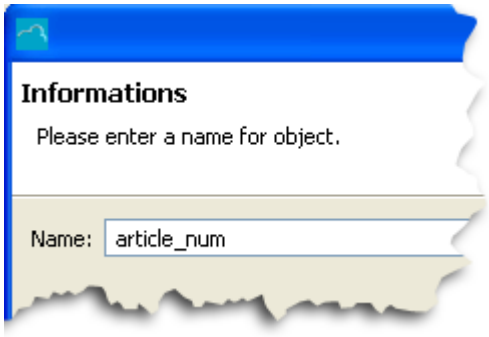


Figure 2 - 37: Entering the jElement name

6 Click on **Finish**.

The new step appears at the end of the Steps Tree Structure in the **Projects View**:

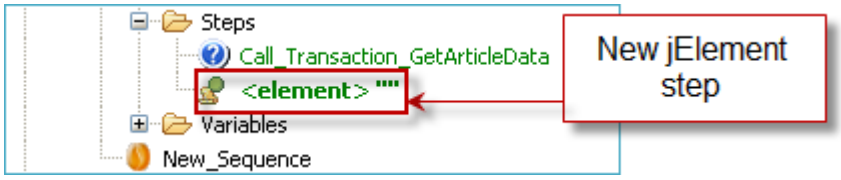


Figure 2 - 38: New jElement step in Steps folder

Now that the *jElement* step is created, we must set its two main properties: **Expression** and **Node name**. The table below describes these properties:

Table 2 - 4: jElement main properties

Property	Description
Expression	JavaScript expression (for example the variable name expression in this project), the value of which appears as content of the generated XML Element.
Node name	Name of the generated XML Element.

7 Select the *jElement* step with a left-click.

The **Properties View** is automatically updated:

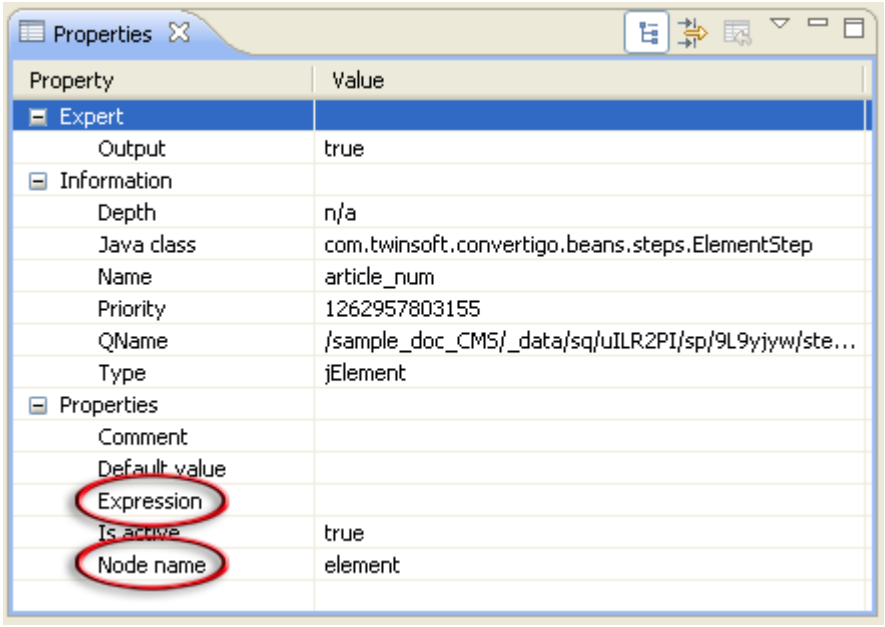


Figure 2 - 39: jElement step properties

8 Click in the **Value** column of the **Node name** property.

The default value (`element`) is highlighted.

9 Type in the value of the node name (for example `article_num`) and press `Enter`.

10 Click in the **Value** column of the **Expression** property.

A  button appears.

11 Click on the  button.

An empty **Javascript expression** window appears.

12 Type in the value of the JavaScript expression (for example `article_code`):

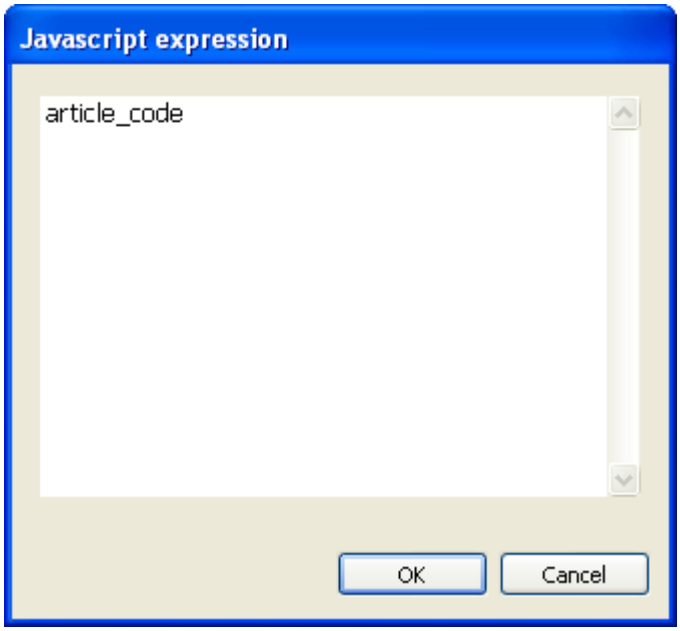


Figure 2 - 40: jElement Expression property value

13 Click on **OK**.

The *jElement* step is now properly set. It appears unsaved in the **Steps** folder of the project:

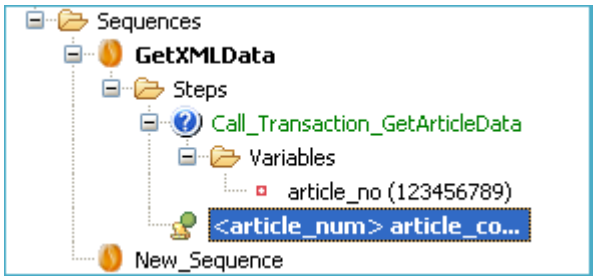


Figure 2 - 41: jElement in Steps folder.

14 Save your project by clicking on  or by pressing `Ctrl + S`.

This step makes available the value needed by the `Call_Transaction_GetArticleData` step, through its `article_no` variable.

To complete the setting, the source of this variable must now be set as pointing towards the XML node generated by the *jElement* step.

For the `Call_Transaction_GetArticleData` step variable source to be able to point towards the mentioned node, the jElement step must be placed before the `Call_Transaction_GetArticleData` step. In other words, its priority must be increased by one rank. This is done in the next procedure.

To increase a step priority

1 Select the step with a left-click:

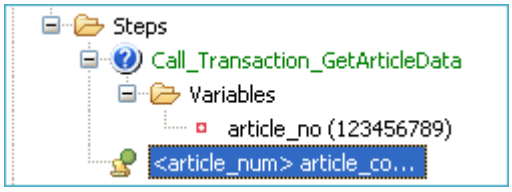


Figure 2 - 42: Selecting the step

2 Click on the **Increase selected object(s) priority** icon in the **Projects View** toolbar:

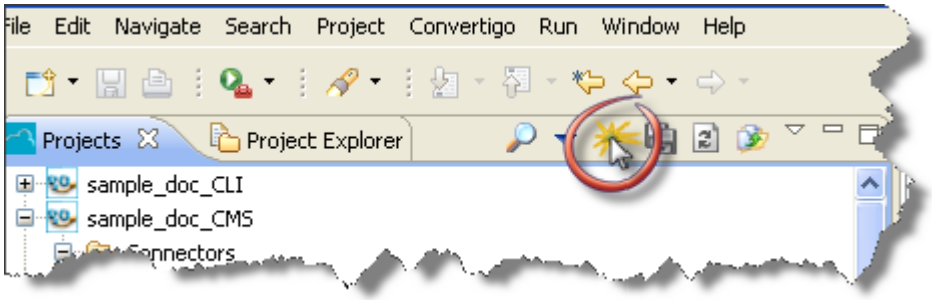


Figure 2 - 43: Increasing the step's priority

The step now appears before the previous step and the sequence is bolded, meaning

that it has been changed and unsaved:

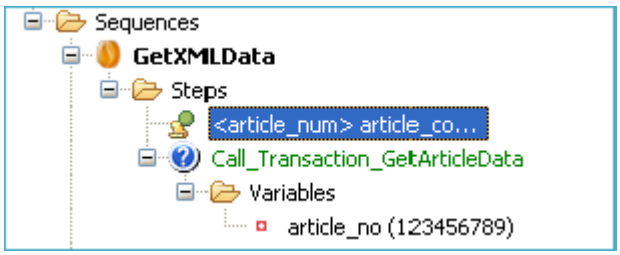


Figure 2 - 44: Step's priority increased by one rank

3 Save your project by clicking on or by pressing `Ctrl + S`.

The *jElement* step is now created and properly positioned in the sequence **Steps** folder.

In this project, when launching the sequence, the *jElement* step will therefore be executed before the *Call Transaction* step.

We can now set the *Call Transaction* step variable **Source** property, by defining the *jElement* step's generated XML node (`<article_num>`) as source for the `Call_Transaction_GetArticleData` step's variable, `article_no`.

To set a step's generated XML node as source for another step's variable

1 Select the *Call Transaction* step variable (`article_no`) with a left-click in the **Projects View**.

The **Properties View** is automatically updated:

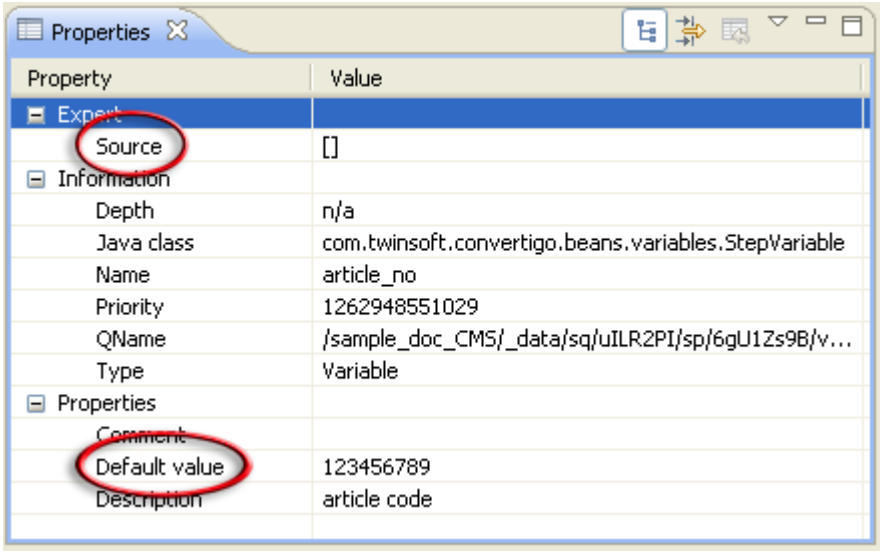


Figure 2 - 45: Properties of Call Transaction step article_code variable

2 In the **Properties View**, click in the **Value** column of the **Default Value** property.

The default value (imported from the transaction) is highlighted.

3 If applicable, delete the variable default value (because the variable value is sourced from the *jElement* step).

4 Press `Enter`.

5 Click in the **Value** column of the **Source** property.

The value is highlighted and a button appears to the right of the field.

6 Click on the button

The **Step Source** wizard is automatically launched.

For more information on the Step Source wizard, see Table "Step Source wizard description" on page 1-16.

7 Click on **New Source** in the upper left corner of the window.

The **Steps Tree Structure**, **XML Output Schema** and **XPath Evaluator** wizard panes become active:

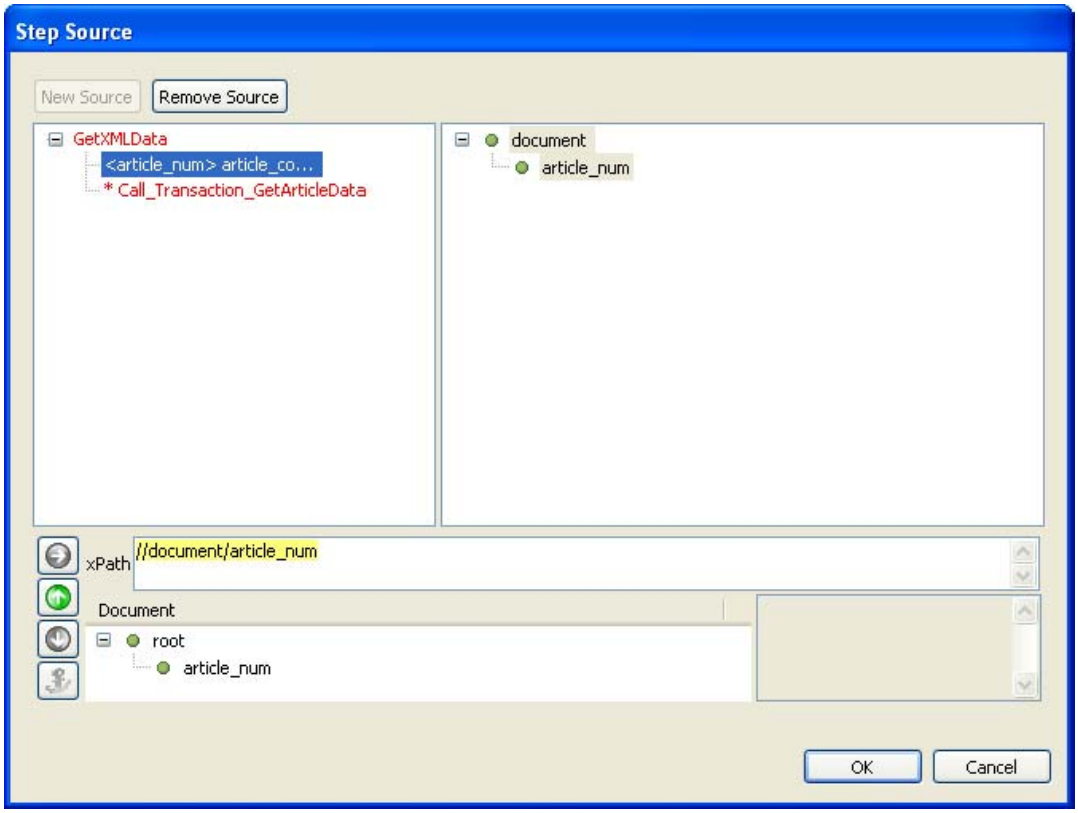


Figure 2 - 46: Step Source wizard

Both previously defined steps appear in the **Steps Tree Structure**. At this point in the project, only one step can be selected: the *jElement* step. It is selected by default when clicking on **New Source**.

The XML schema of the XML generated by the *jElement* step appears in the **XML Output Schema** pane to the right. It contains only one node called after the value of the **Node name** property set for the step (see Table *"jElement main properties"* on page 2-25): `article_num`.

8 In the **XML Output Schema**, select the `article_num` node:

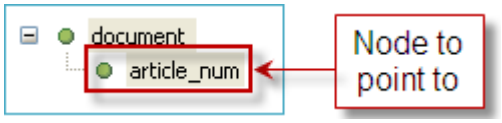


Figure 2 - 47: XML Output Schema of jElement step

The XPath to this node is automatically generated in the **xPath** field of the **XPath Evaluator** and the result of this XPath on the XML schema is displayed in the **Document** tree structure:

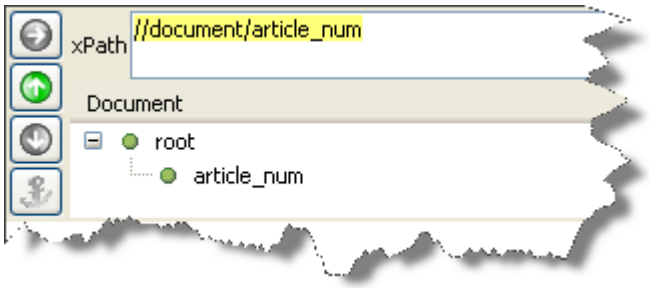


Figure 2 - 48: Generation of XPath to article_num node

9 Click on **OK**.

The `article_no` variable is automatically assigned the source value, which appears in the **Value** column of the **Source** cell:

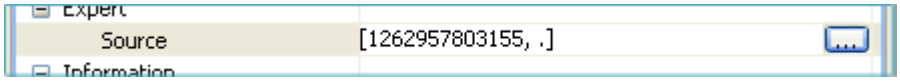


Figure 2 - 49: Variable source value automatically updated

The variable source of the `Call_Transaction_GetArticleData` step is therefore automatically and properly set to point towards the `article_num` node generated by the *jElement* step, the JavaScript expression of which is `article_code`.

As a conclusion, the `article_no` input variable imported from the target transaction and needed by the `Call_Transaction_GetArticleData` step has been assimilated with the `article_code` variable contained in the `GetXMLData` sequence JavaScript scope.

10 Save your project by clicking on  or by pressing `Ctrl + S`.

The *Call Transaction* step is properly set and its target transaction, `GetArticleData`, can now be called.

What comes next?

This step serves as a basis for a number of steps executed later in the sequence. Indeed, when called, the target transaction produces an XML output upon which a number of actions are based (for example checking the presence of nodes within the XML output, etc. - see *"Mashup Sequencing Project"* on page 2-3).

Following steps can therefore be created, and their sources set as pointing towards the required nodes of the `GetArticleData` transaction XML output schema (using XPaths).

The following step consists in checking if, after the `GetArticleData` transaction has been

called, the generated XML contains *at least* one of the following XML elements: `statut_article`, `rsp_article` or `product_group`.

To this end, an *IfExistThenElse* step will be created and set.

To create and set an IfExistThenElse step

1 Right-click on the sequence (for example `GetXMLData`).

A contextual menu appears.

2 Select ***New > Step***.

The **New Step** wizard is automatically launched.

3 In the **Other steps** area, select **IfExistThenElse**:

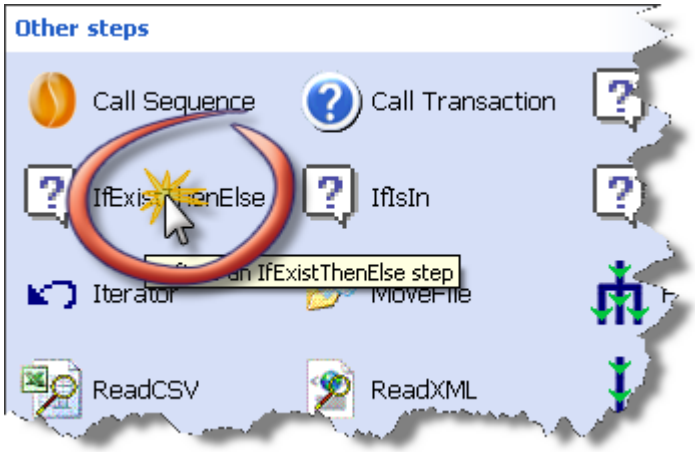


Figure 2 - 50: Selecting an IfExistThenElse step

4 Click on **Next**.

5 In the **Name** field that appears, type in the *IfExist* step name (for example `IfArticleExists`):

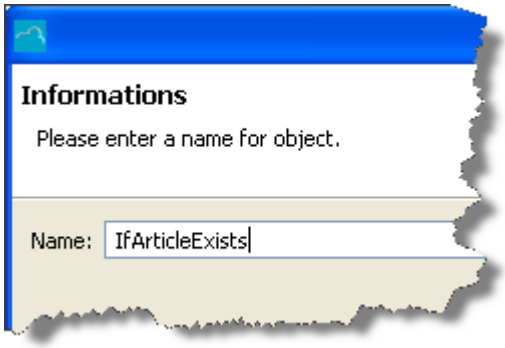


Figure 2 - 51: Entering the IfExistThenElse step name

6 Click on **Finish**.

The new step appears at the end of the Steps Tree Structure of the sequence in the **Projects View** with two sub-steps (*Then* and *Else*) corresponding to the two possible test result cases.

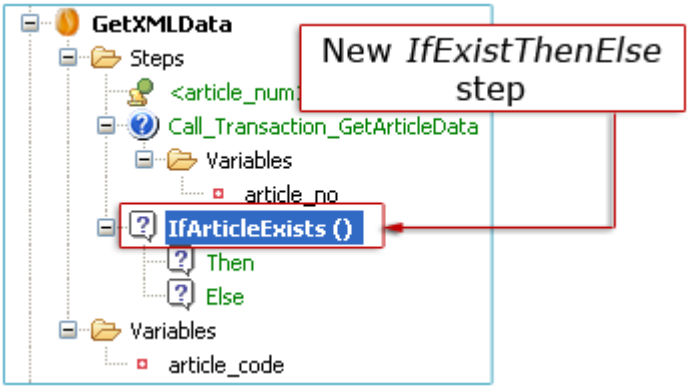


Figure 2 - 52: New IfExistThenElse step in Steps folder

Now that the *ifExistThenElse* step has been created, its source must be set so as to point towards XML nodes (`article_status`, `article_rsp` and `product_group`) to be checked within the XML output returned by the `GetArticleData` transaction (called by the preceding step).

7 Save your project by clicking on or by pressing `Ctrl + S`.

8 Select the *IfExistThenElse* step with a left-click.

The **Properties View** is automatically updated.

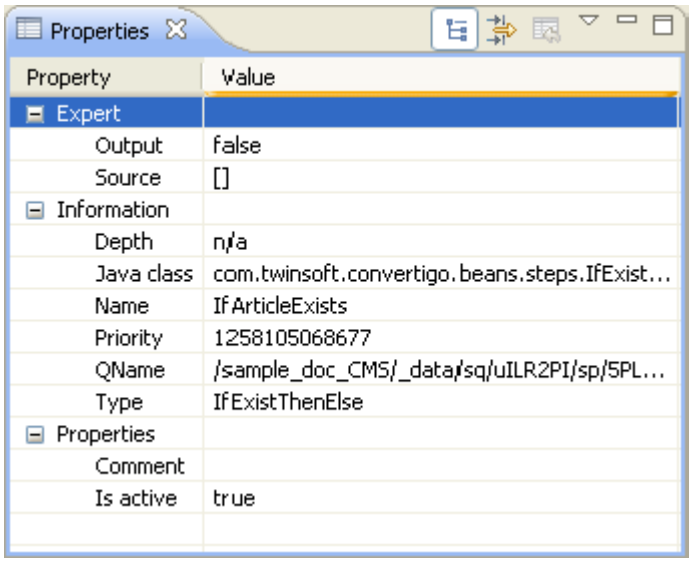


Figure 2 - 53: IfExistTheElse step properties

9 Click in the **Value** column of the **Source** property.

A button appears.

10 Click on the button to launch the **Step Source** wizard.

The **Step Source** wizard is automatically launched.

11 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

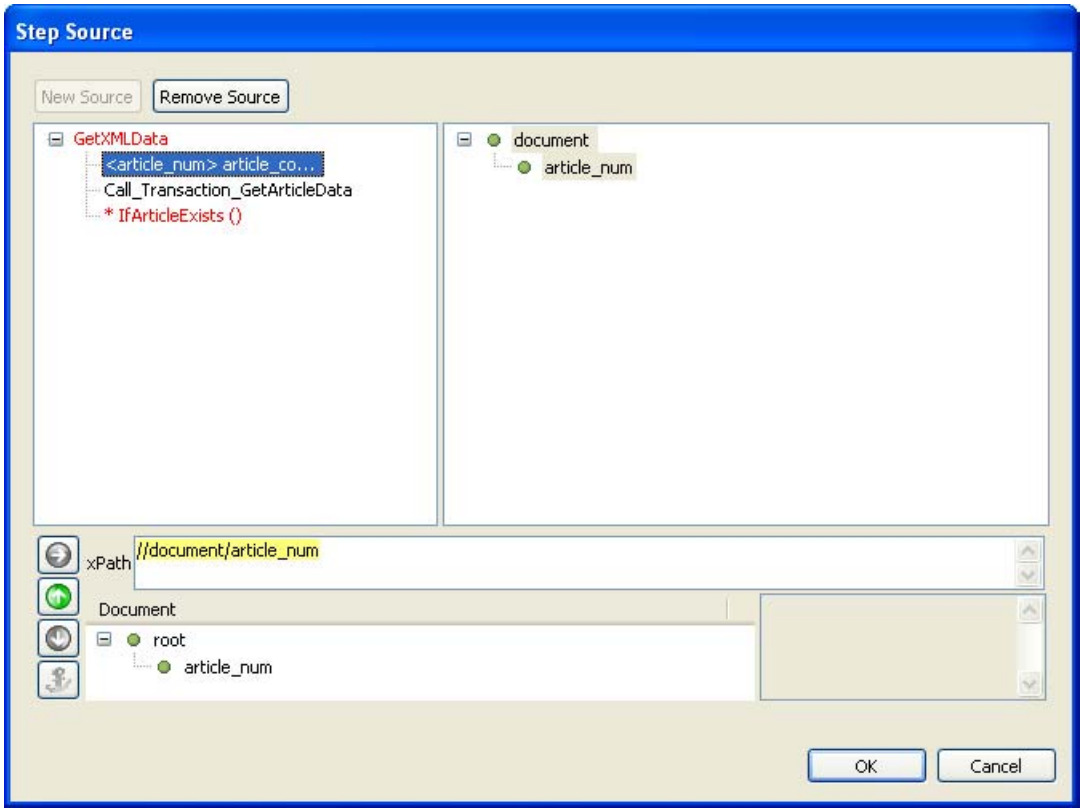


Figure 2 - 54: Setting of IfArticleDataExist step source

The previously defined steps appear in the **Steps Tree Structure**. At this point in the project, two steps can be selected: the *jElement* step and the *Call Transaction* step.

We need to check nodes in the XML generated by the `Call_Transaction_GetArticleData` step. This step must therefore be selected first to access its **XML Output Schema**.

12 In the **Steps Tree Structure**, select `Call_Transaction_GetArticleData` with a left-click.

The selected step's **XML Output Schema** is automatically displayed:

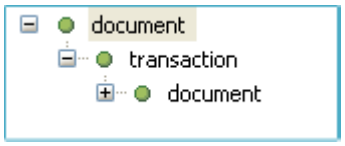


Figure 2 - 55: XML Schema of Call_transaction_GetArticleData step

13 Expand the `document` element, then `blocks` nodes by clicking on ⊞:

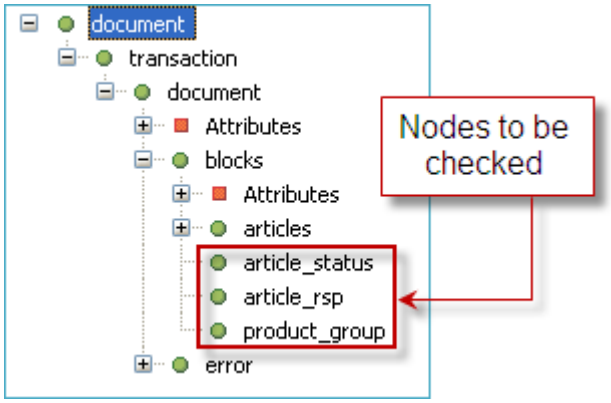


Figure 2 - 56: Nodes to check in the called transaction XML Output Schema

Selecting any node with a left-click automatically generates the XPath to the selected node in the **xPath** field of the **XPath Evaluator**.

The check to be performed must apply to the existence of *at least* one of these nodes.

To this end, we will:

- generate the XPath for one node only,
- edit the XPath expression using an `or` operator to include the two other nodes to be checked.

For more information on XPath syntax, see the XPath tutorial from W3Schools at http://www.w3schools.com/XPath

14 In the **XML Output Schema**, select the `article_status` node.

The XPath expression to this node is automatically generated in the **XPath Evaluator**:

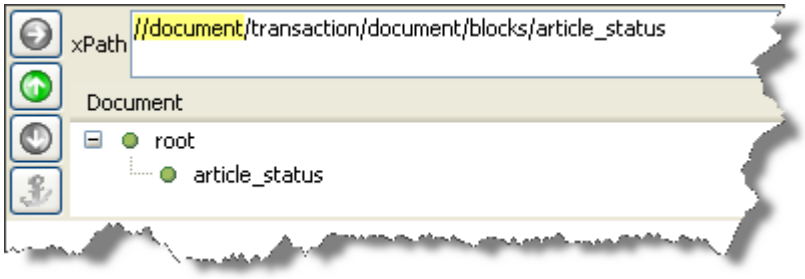


Figure 2 - 57: Generation of XPath to article_status node

15 Click in the **xPath** field of the **XPath Evaluator**.

16 Change the XPath expression to:

```
//document/transaction/document/blocks[article_status or
article_rsp or product_group]
```

This Xpath addresses a parent `blocks` element, containing at least a child node with tag name `article_status` or `article_rsp` or `product_group`. It matches if one of the above mentioned nodes exists in the *Call Transaction* step XML schema output.

17 Press `Enter` to check that the execution of the XPath expression on the XML Schema

returns expected nodes.

The **Document** tree structure displays the result of the execution:

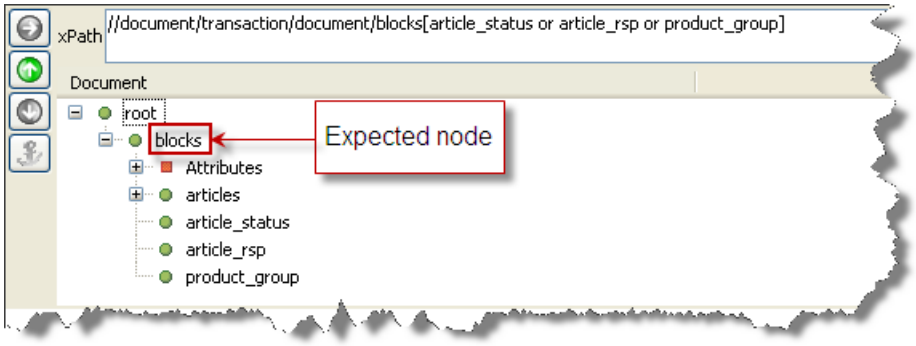


Figure 2 - 58: Display of XPath execution on XML Schema in Document tree structure

18 Click on **OK**.

The step source property is automatically updated.

19 Save your project by clicking on  or by pressing `Ctrl + S`.

WHAT COMES NEXT?

The step for checking the presence of required nodes in the `GetArticleData` transaction XML output is now set.

According to the project description (see *"First sequence description"* on page 2-3), the next milestone in the project is to set the two *IfExistThenElse* step sub-steps - *Then* and *Else* - so that the condition expressed in Table *"`IfArticleExists` sub-steps (Then and Else) definition"* on page 2-35 is fulfilled:

Table 2 - 5: `IfArticleExists` sub-steps (*Then* and *Else*) definition

If one of the nodes checked by the `IfArticleExists` step is present, THEN...	**...ELSE...**
...a parent *Complex* `<articleFound>` XML element as well as four child simple XML elements are generated. They are defined as follows: • `<code>` contains the value of the `article_code` variable (content sourced from the *jElement* step XML schema), • the content of the generated `<status>`, `<rsp>` and `<product_group>` elements is sourced from the `Call_Transaction_GetArticleData` step XML schema[a].	...a parent *Complex* `<articleNotFound>` XML element is generated together with a child `<code>` node, the content of which is sourced from the *jElement* step XML schema.

a. *The three tag names are created even if sourced XML nodes do not exist.*

The steps generating the above mentioned complex and simple elements will now be created and set as child steps of the `IfArticleExists` sub-steps (*Then* and *Else)*.

We will start by creating the *Then* sub-step generating the *Complex* XML element, `<articleFound>`.

To create and set a Complex step

1 Right-click on the *Then* sub-step.

A contextual menu appears.

2 Select ***New > Step***.

A **New Step** wizard is automatically launched.

3 In the **Generating XML steps** area, select **Complex**:

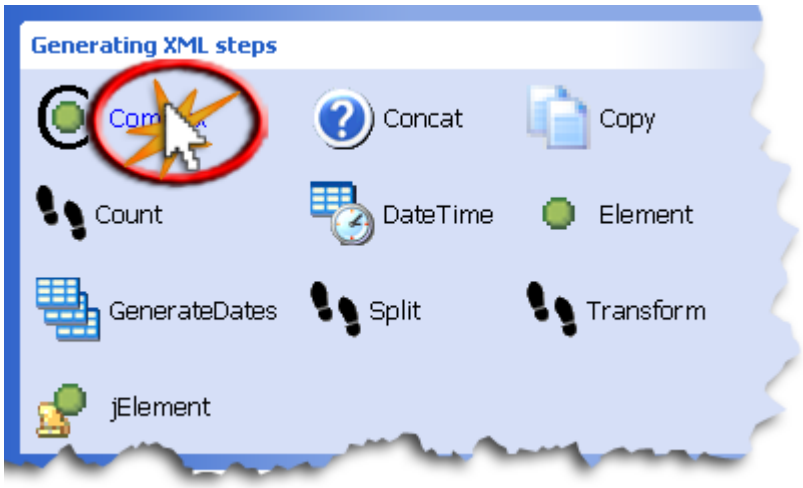


Figure 2 - 59: Selecting a Complex step

A *Complex* step generates a complex XML element.

4 Click **Next**.

5 In the **Name** field that appears, type in the *Element* name (for example `articleFound`):

Figure 2 - 60: Entering the Complex step name

6 Click on **Finish**.

The new step appears in the **Steps** folder of the project:

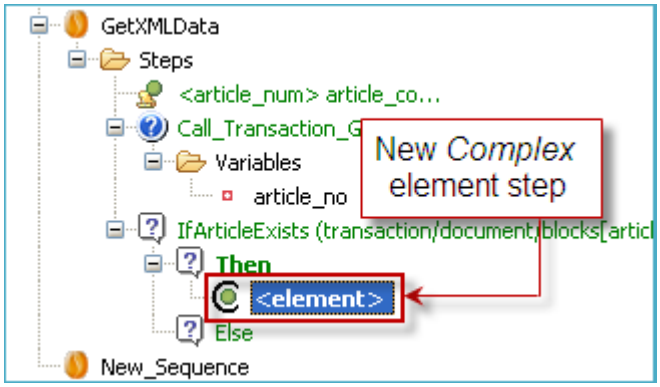


Figure 2 - 61: New Complex step in Steps folder

7 Save your project by clicking on  or by pressing `Ctrl + S`.

Now that the *Complex* step is created, we must set its main property: **Node Name**.

The value of the **Node Name** property sets the name of the XML tag generated by the *Complex* step when one of the three nodes previously checked by the `IfArticleExists` step is found in the generated `GetArticleData` transaction XML output.

In our case, this name is `articleFound`.

8 In the **Projects View**, select the *Complex* step with a left-click.

The **Properties View** is automatically updated:

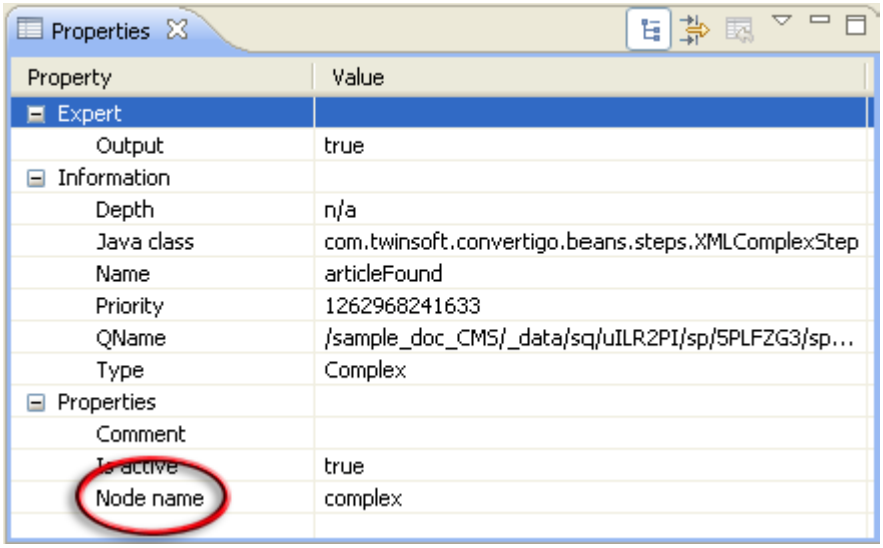


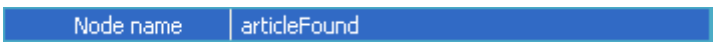
Figure 2 - 62: Complex step properties

9 Click in the **Value** column of the **Node Name** property.

The default value (`complex`) is highlighted.

10 Enter the required node name (for example `articleFound`).

11 Press `Enter`.

The value is updated: 

In the **Projects View**, the step appears changed and unsaved:

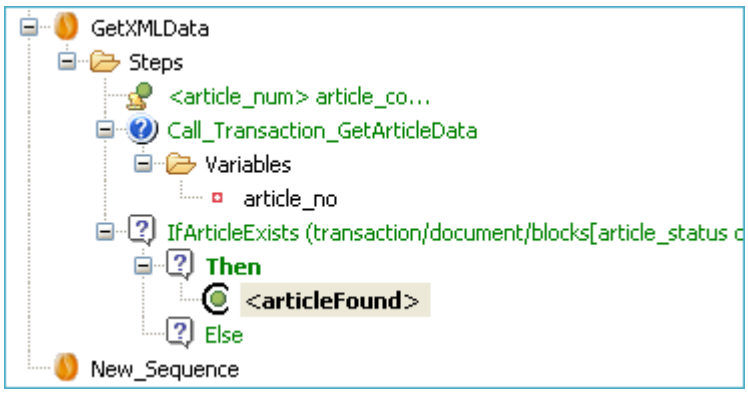


Figure 2 - 63: Complex step set

12 Save your project by clicking on or by pressing `Ctrl + S`.

The *Complex* step is now created and set to insert a complex `<articleFound>` XML node in the sequence XML output when required nodes are found in the `GetArticleData` transaction.

WHAT COMES NEXT?

We will now create and set the four *Element* steps, children of the previously created *Complex* step: `code`, `status`, `rsp` and `product_group`.

To create and set an Element step

1 Right-click on the parent step (for example `articleFound`).

A contextual menu appears.

2 Select ***New > Step***.

A **New Step** wizard is automatically launched.

3 In the **Generating XML step** area, select **Element**:

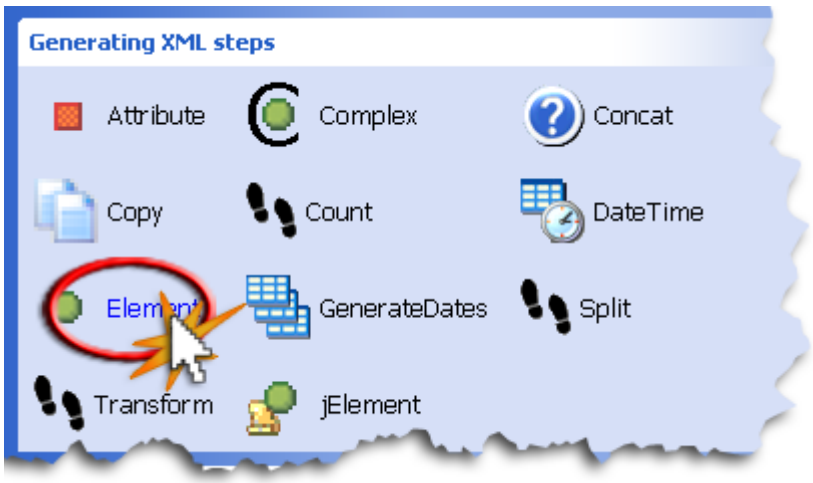


Figure 2 - 64: Selecting an Element step

An *Element* step generates a simple XML element.

4 Click **Next**.

5 In the **Name** field that appears, type in the *Element* name (for example `code`):

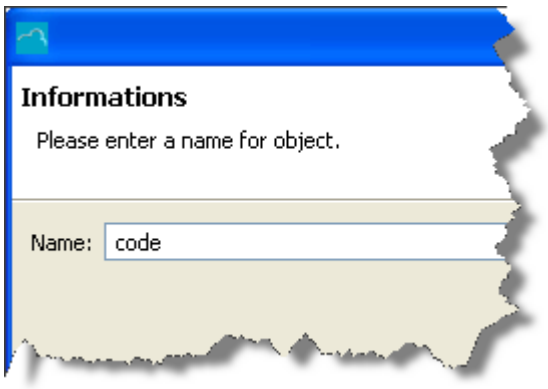


Figure 2 - 65: Entering the Element step name

6 Click on **Finish**.

The new step appears in the **Steps** folder of the project:

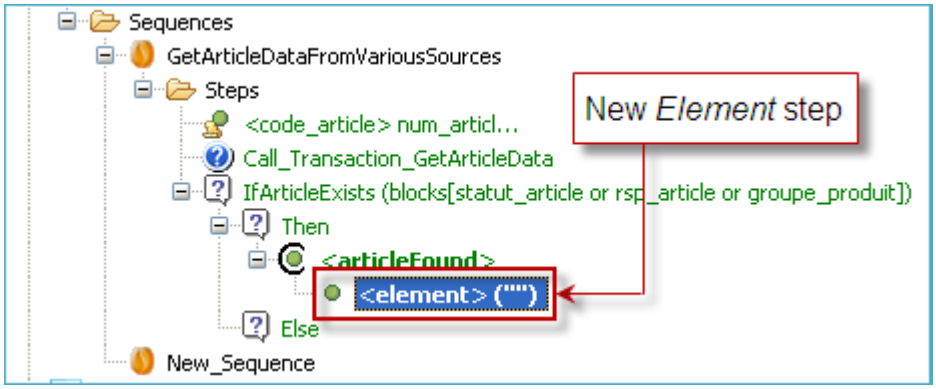


Figure 2 - 66: New Element step in Steps folder

7 Save your project by clicking on  or by pressing `Ctrl + S`.

Now that the *Element* step is created, we must set its two main properties: **Node Name** and **Source**.

The value of the **Node Name** property sets the name of the XML tag generated by the *Element* step (here, `code`).

As mentioned in Table *"`IfArticleExists` sub-steps (Then and Else) definition"* on page 2-35, we would like the content of the `code` element to be sourced from the `article_code` variable.

To this end, we will set the *Element* step source so that it points to the `article_num` node (containing the `article_code` variable value) generated by the first *jElement* step.

8 In the **Projects View**, select the *Element* step with a left-click.

The **Properties View** is automatically updated:

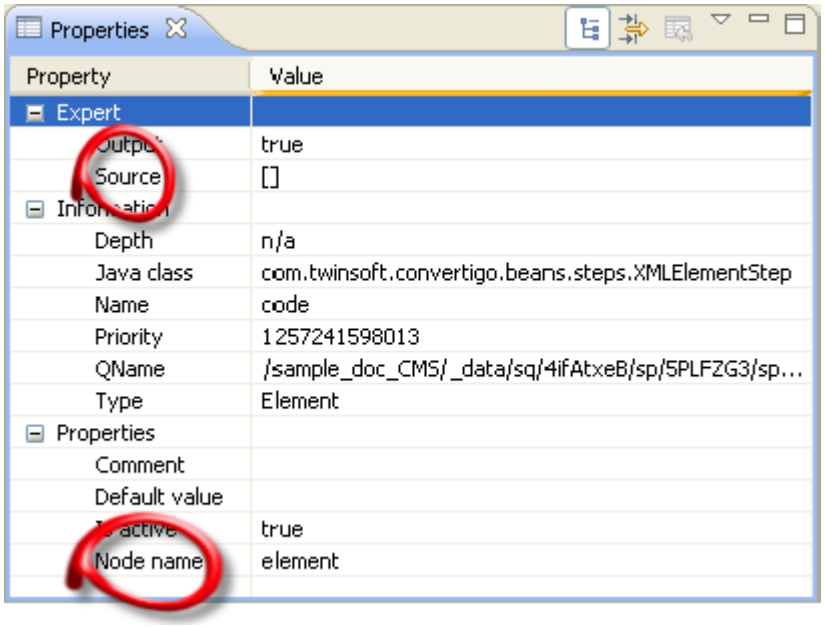


Figure 2 - 67: Element step properties

9 Click in the **Value** column of the **Node Name** property.

The value is highlighted.

10 Enter the required node name (for example `code`).

11 Press `Enter`.

The value is updated: Node name | code

12 Click in the **Value** column of the **Source** property.

A button appears.

13 Click on the button to launch the **Step Source** wizard.

The **Step Source** wizard is automatically launched.

14 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

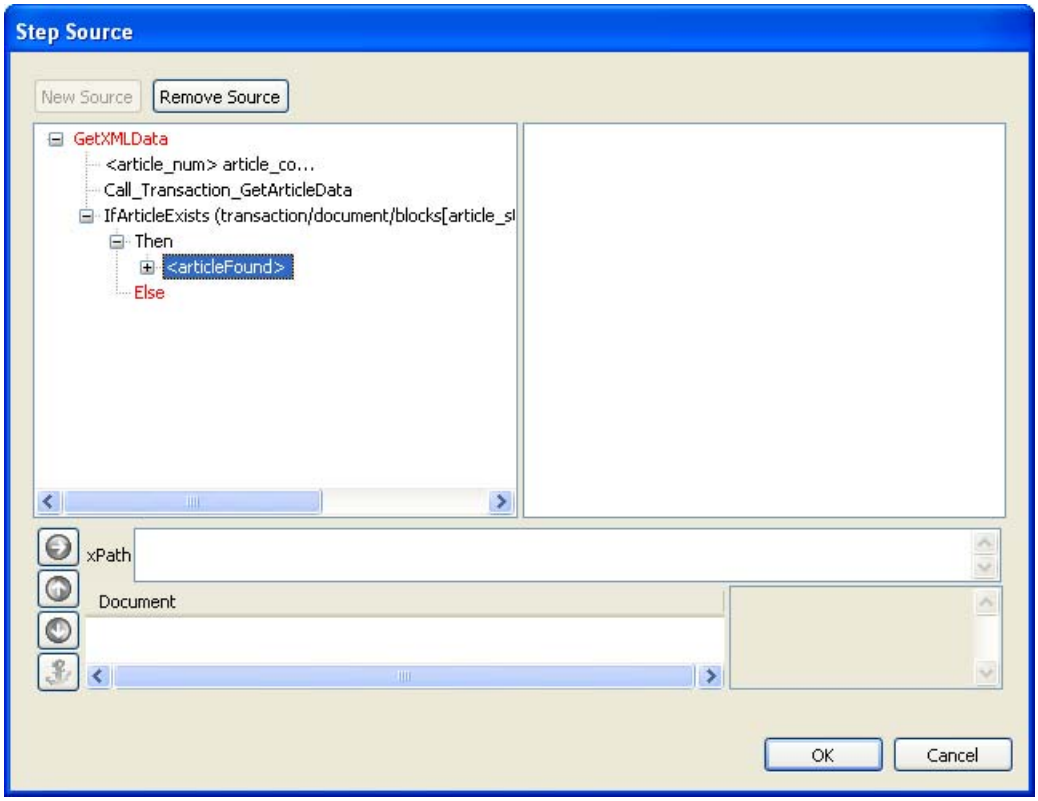


Figure 2 - 68: Setting of Element step source

The steps defined so far appear in the **Steps Tree Structure**.

15 In the **Steps Tree Structure**, select the `<article_num> article_code` step to display its XML schema.

The selected step's **XML Output Schema** is automatically displayed:

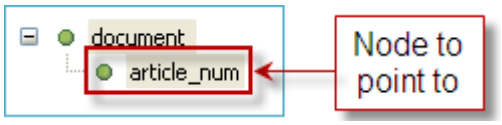


Figure 2 - 69: XML Output Schema of jElement step

This **XML Output Schema** contains only one node called after the value of the **Node name** property set for the step (see Table *"jElement main properties"* on page 2-25): `article_num`.

16 In the **XML Output Schema**, select the `article_num` node:

The XPath to this node is automatically generated in the **xPath** field of the **XPath Evaluator** and the result of this XPath on the XML schema is displayed in the **Document** tree structure:

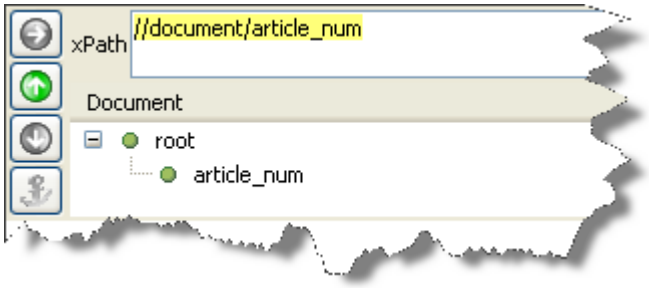


Figure 2 - 70: Generation of XPath to article_num node

17 Click on **OK**.

The *Element* step is now created and set to insert a simple <code> XML node and its content (sourced from the *jElement* step) in the sequence XML output.

18 Save your project by clicking on  or by pressing `Ctrl + S`.

WHAT COMES NEXT?

We will now create and set the remaining three *Element* steps.

The following procedure is similar to the previously described procedure for creating and setting the `code` element (see "To create and set an Element step" on page 2-38). Only the node name and source change. For documentation lightness purpose, the procedure is repeated here with a limited number of snapshots.

To create and set the `status`, `rsp` and `product_group` Element steps

1 Right-click on the parent step (for example `articleFound`).

A contextual menu appears.

2 Select ***New > Step***.

A **New Step** wizard is automatically launched.

3 In the **Generating XML step** area, select **Element**.

4 Click **Next**.

5 In the **Name** field that appears, type in the *Element* name (for example `status`):

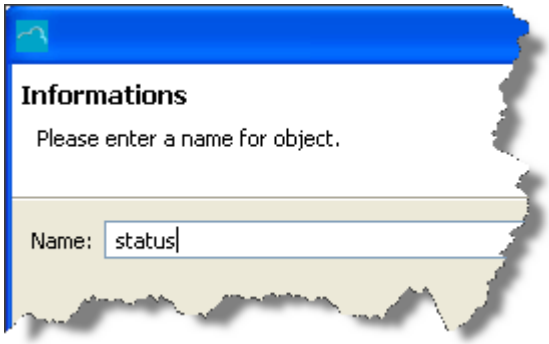


Figure 2 - 71: Entering the Element step name

6 Click on **Finish**.

The new step appears in the **Steps** folder of the project.

7 Save your project by clicking on  or by pressing `Ctrl + S`.

Now that the *Element* step is created, we must set its two main properties: **Node Name** and **Source**.

The value of the **Node Name** property sets the name of the XML tag generated by the *Element* step (here, `status`).

As mentioned in Table *"`IfArticleExists` sub-steps (Then and Else) definition"* on page 2-35, we would like the `status` element to be sourced from the `article_status` XML node generated by the `GetArticleData` transaction when called by the associated step.

To this end, we will set the *Element* step source so that it points to the `article_status` node generated by the first `Call_Transaction_GetArticleData` step.

8 In the **Projects View**, select the *Element* step with a left-click.

The **Properties View** is automatically updated.

9 Click in the **Value** column of the **Node Name** property.

The value is highlighted.

10 Enter the required node name (for example `status`).

11 Press `Enter`.

The value is updated.

12 Click in the **Value** column of the **Source** property.

A  button appears.

13 Click on the  button to launch the **Step Source** wizard.

The **Step Source** wizard is automatically launched.

14 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16).

The steps defined so far appear in the **Steps Tree Structure**.

15 In the **Steps Tree Structure**, select the `Call_Transaction_GetArticleData` step to display its XML schema.

The selected step's **XML Output Schema** is automatically displayed:

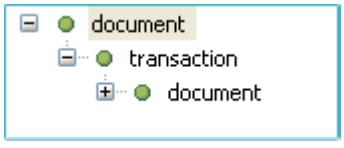


Figure 2 - 72: XML Schema of Call_transaction_GetArticleData step

This **XML Output Schema** contains all nodes generated by the `GetArticleData` transaction.

16 Expand the `blocks` node by clicking on ⊞:

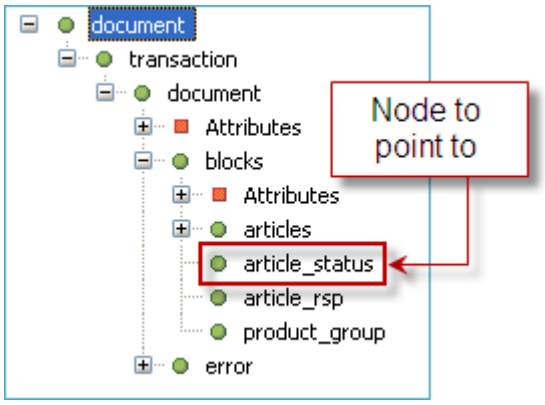


Figure 2 - 73: Source of status Element step

17 In the **XML Output Schema**, select the `article_status` node.

The XPath to this node is automatically generated in the **xPath** field of the **XPath Evaluator** and the result of this XPath on the XML schema is displayed in the **Document** tree structure:

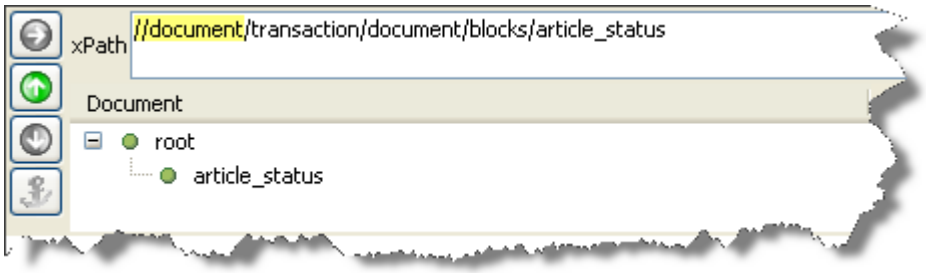


Figure 2 - 74: Generation of XPath to article_status node

18 Click on **OK**.

The *Element* step is now created and set to insert a simple `<status>` XML node and its content (sourced from the `Call_Transaction_GetArticleData` XML output) in the sequence XML output.

19 Save your project by clicking on  or by pressing `Ctrl + S`.

20 Repeat points 1 to 18 of this procedure to set the remaining two *Element* steps (`rsp_article` and `product_group`), the source of which must point to:

Table 2 - 6: Remaining *Element* step sources

The source of *Element* step...	...must point to node[a]...	...generated by step...
`<rsp>`	`article_rsp`	`Call_Transaction_GetArticleData`
`<product_group>`	`product_group`	

a. *see Figure 2 - 73.*

*You can also copy the previously created step (`code`) and paste it as child of the Complex element (`articleFound`), then set the **Node name** and **Source** properties of pasted steps as required.*

After all *Element* steps have been created and set, they appear in the sequence **Steps** folder as follows:

Figure 2 - 75: Element steps created and set as children of the Complex articleFound step

The *Then* sub-step functionally addressing the condition: *"If an article exists in the XML output generated by the `GetArticleData` transaction - through the presence of the `article_status`, `article_rsp` or `product_group` node - **then** create an `<articleFound>` element including specific article information"* is now created.

WHAT COMES NEXT?

We will now set the *Else* sub-step addressing the other option: "*If **no** article exists in the XML output generated by the `GetArticleData` transaction - through the **absence** of the `article_status`, `article_rsp` or `product_group` node - **then** create an `<articleNotFound>` element including specific article information*."

The setting of the *Else* sub-step is actually similar to the setting of the *Then* sub-step.

Indeed:

- like the *Then* sub-step, the *Else* sub-step generates a complex XML element (called `<articleNotFound>` in the latter case),
- like the *Then* sub-step, the *Else* sub-step generates a simple element called `<code>`, the content of which is sourced from the *jElement* step XML schema.

The table below describes the procedures to be followed to create and set the *Else* sub-step, based on procedures used to create and set the *Then* sub-step:

Table 2 - 7: *Else* sub-step "setting plan"

N.	Setting	Procedure	Comment
1	Creation and setting of the `articleNotFound` *Complex* step	To create and set a Complex step	**Node Name** property value to be set as `articleNotFound`

Table 2 - 7: *Else* sub-step "setting plan" (...)

N.	Setting	Procedure	Comment
2	Creation and setting of the `code` *Element* step	To create and set an Element step	-

Please follow carefully above mentioned procedures and comments, then pass on to the next procedure.

You can also copy required steps (`articleFound` and `code`), paste them as children of the Else sub-step, then set the `articleNotFound` step ***Node Name*** *property as required.*

Before passing on to the next procedure, make sure that the sequence **Steps** folder contains the following steps:

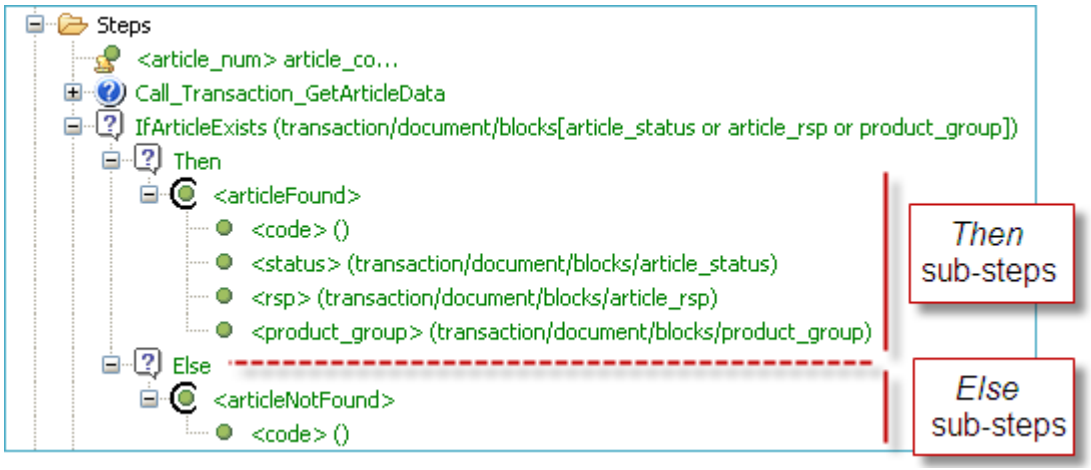


Figure 2 - 76: IfArticlesExist step Then and Else sub-step tree structure

What comes next?

Points 1 to 3 of the first sequence description (see *"First sequence description"* on page 2-3) are now implemented in terms of Convertigo *steps*.

According to the sequence description, we now need to:

1 *Check* the presence of the `articles` XML table in the XML generated by the `GetArticleData` transaction. This check is performed by an *IfExists* step (*see "To create and set the `IfArticleDataExist` step"* on page 2-81).

2 If so, *generate* an `<articlesList>` complex element then *iterate* on each row of the `articles` table using an *Iterator* step and, for each row:

- *call* the `searchGoogleWithLimit` CWI transaction,
- *generate* a complex `<article>` XML element with three attributes, the value of which is sourced from the `articles` XML table `code`, `name` and `status` XML nodes generated by the previously called `GetArticleData` transaction.
- *check* the presence of `resultItem` nodes in each XML output returned by the CWI `searchGoogleWithLimit` transaction (called for each row) and, if present, *copy* the content of `resultItem` nodes together with their child nodes into the sequence

XML output.

The table below sums up the different CMS objects (steps and SQL transactions) to be created and set in this second part of the first sequence:

Table 2 - 8: CMS objects needed in the second part of first sequence project

CMS object type	Object name	Description
IfExist step	`IfArticlesTableExists`	Checks in the `GetArticleData` CLI transaction XML output the presence of `articles` nodes.
	`IfResultItemsExist`	Checks in the `searchGoogleWithLimit` transaction XML output the presence of `resultItems` nodes.
Complex step	`ArticlesList`	Complex Element named `ArticlesList`.
	`Article`	Complex Element named `Article`.
Iterator step	`IteratorOnEachRow`	Iterates on each `row` node of the `articles` XML table generated by the `GetArticleData` transaction.
Call transaction step	`Call_Transaction_searchGoogle`	Calls CWI project-specific `searchGoogleWithLimit` transaction
Attribute step	`article_code`	Attribute named `article_code`.
	`article_name`	Attribute named `article_name`.
	`article_status`	Attribute named `article_status`.
Copy step	`IteratorOnEachRow`	Iterates on each `row` node of the `articles` XML table generated by the `GetArticleData` transaction.

The following procedures show how to create and set these ojects as they are needed in the sequence.

The next step needed in the sequence is the `IfArticlesTableExists` step. The purpose of this step is to check the presence of `articles` node in the XML output generated by the `GetArticleData` transaction.

To do so, we will create an *IfExist* step called `IfArticlesTableExists`, the source of which will point towards the required `articles` node in the XML output generated by the `Call_Transaction_GetArticleData` step.

To create and set an IfExist step

1 In the **Projects View**, right-click on the sequence (for example `GetXMLData`).

A contextual menu appears.

2 Select ***New > Step***.

The **New Step** wizard is automatically launched.

3 In the **Other steps** area, select **IfExist**:

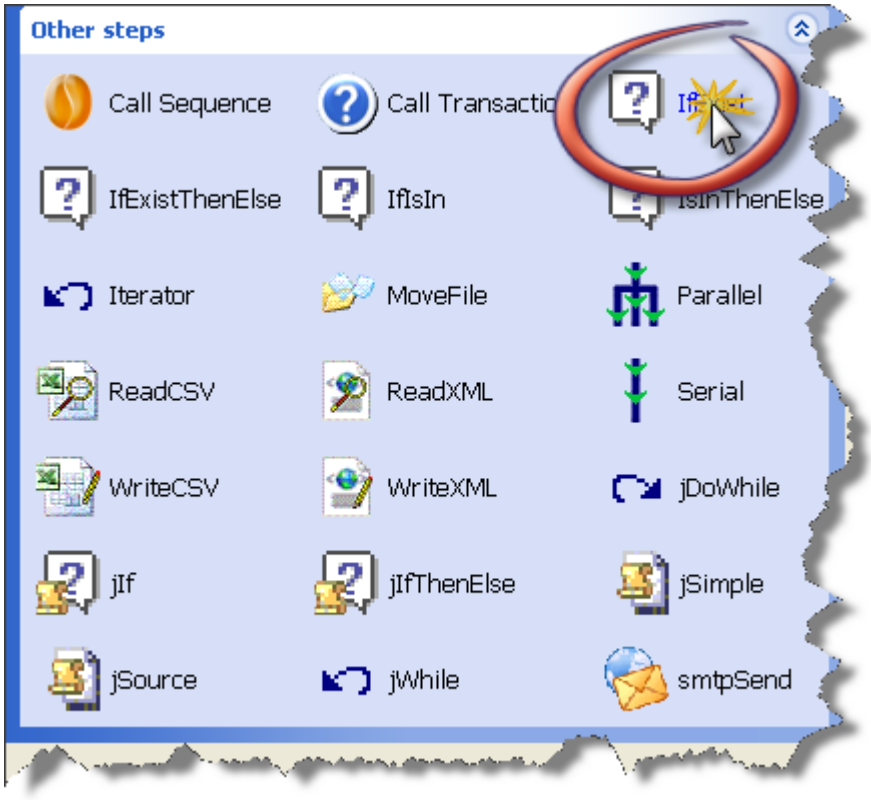


Figure 2 - 77: Selecting an IfExist step

4 Click on **Next**.

5 In the **Name** field that appears, type in the *IfExist* step name (for example `IfArticlesTableExists`):

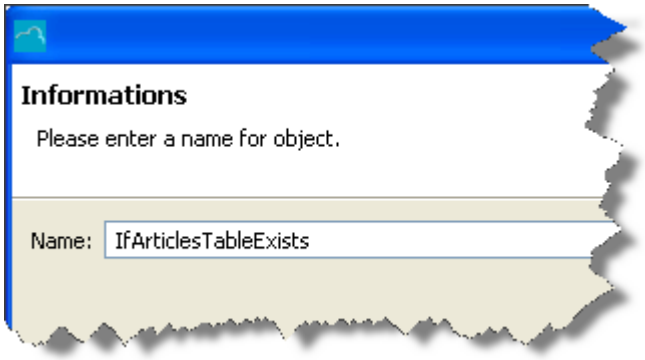


Figure 2 - 78: Entering the IfExists step name

6 Click on **Finish**.

The new step appears at the end of the sequence tree structure in the **Projects View**:

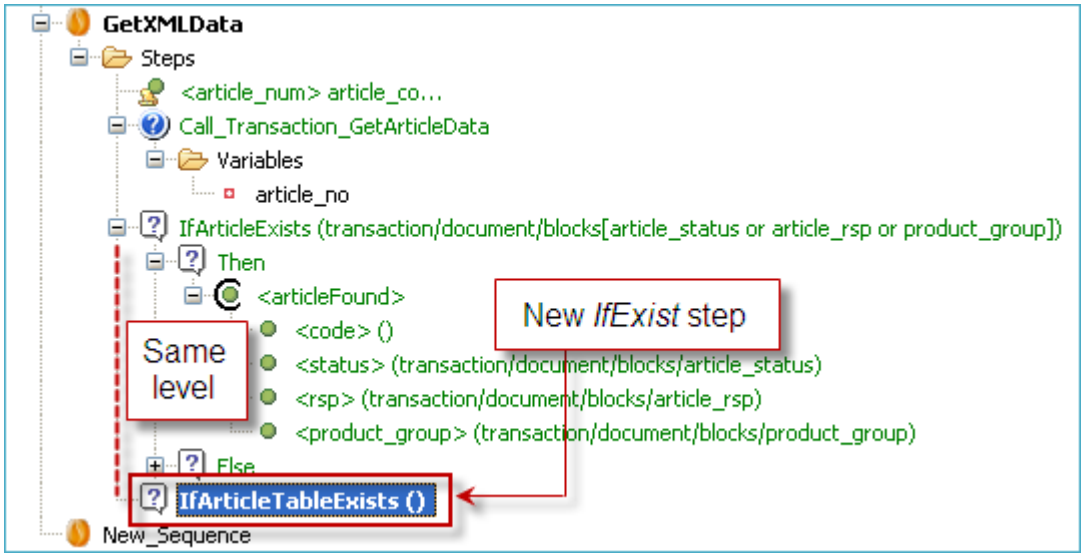


Figure 2 - 79: New IfExist step in Steps folder

The *IfExistThenElse* and *IfExist* steps stand alone. They are not correlated and can therefore appear at the same level in the sequence.

7 Save your project by clicking on or by pressing `Ctrl + S`.

Now that the *ifExist* step has been created, its source must be set.

The step source needs to point towards the XML node (`articles`) to be checked within the XML output returned by the `GetArticleData` transaction (called by the `Call_Transaction_GetArticleData` step).

8 Select the *IfExist* step with a left-click.

The **Properties View** is automatically updated.

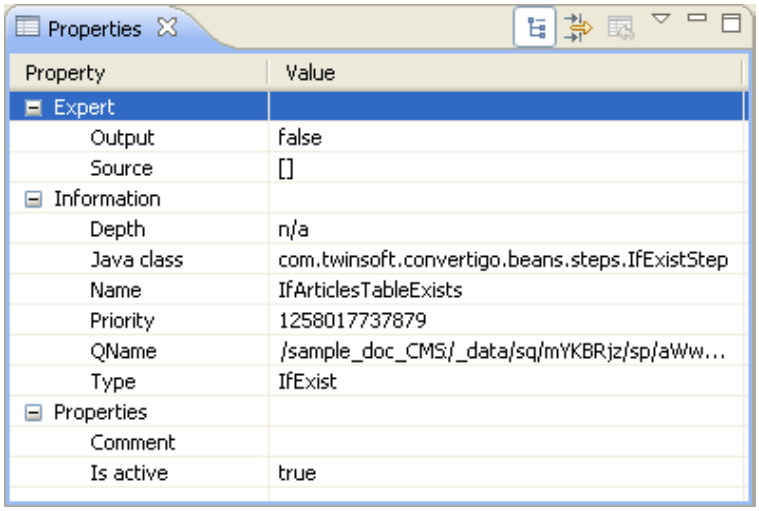


Figure 2 - 80: IfExist step properties

9 Click in the **Value** column of the **Source** property.

A button appears.

10 Click on the button to launch the **Step Source** wizard.

The **Step Source** wizard is automatically launched.

11 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

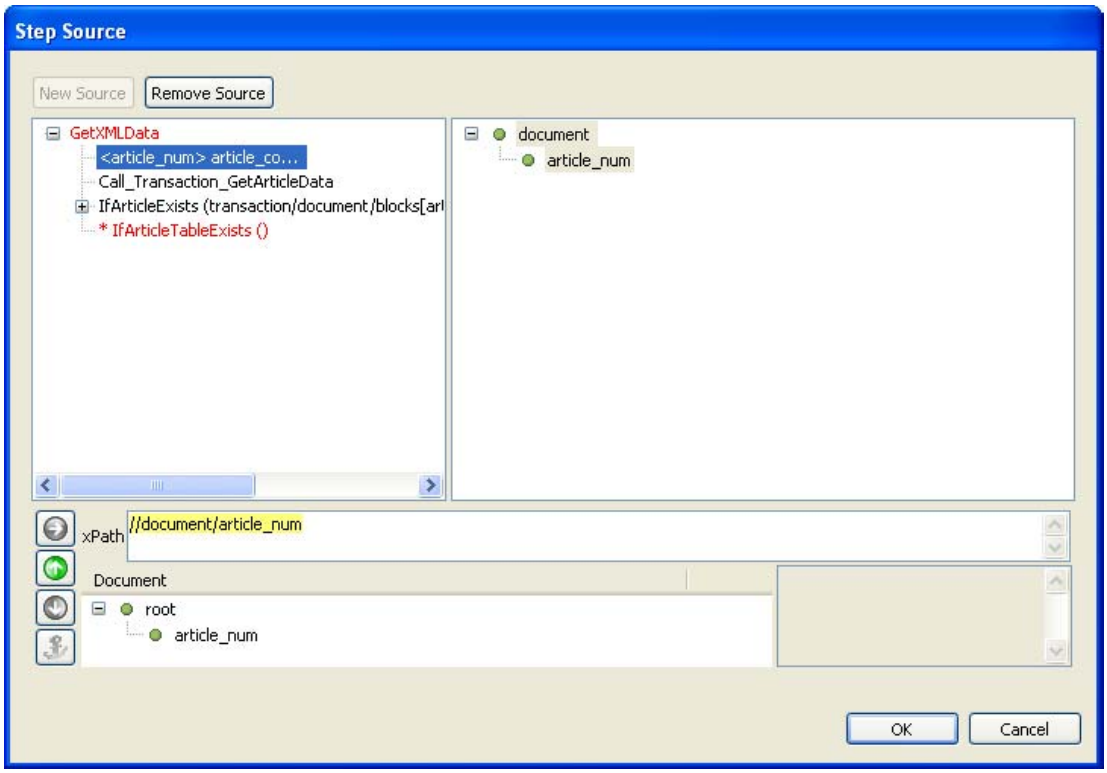


Figure 2 - 81: Setting of IfArticlesTableExists step source

The *IfExist* check must be performed on the XML schema of the output generated when calling the `GetArticleData` transaction. The step to be selected is therefore `Call_Transaction_GetArticleData`.

12 In the **Steps Tree Structure**, select the required step (for example `Call_Transaction_GetArticleData`) with a left-click.

The selected step's **XML Output Schema** is displayed:

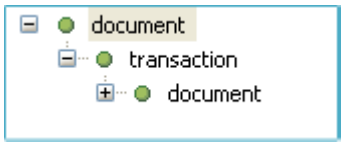


Figure 2 - 82: XML Schema of Call_transaction_GetArticleData step

13 Expand the `document`, then `blocks` nodes by clicking on ⊞:

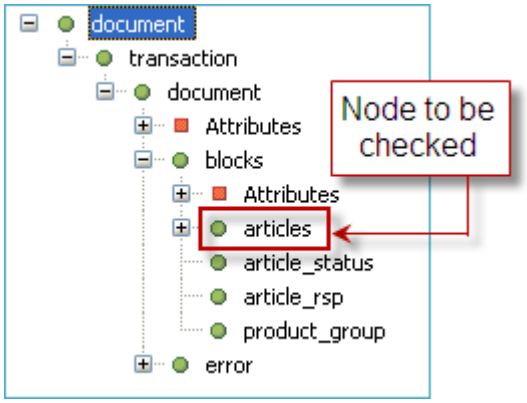


Figure 2 - 83: Node to check in the called transaction XML output

14 In the **XML Output Schema**, select the node to be checked (`articles`).

15 The XPath expression to this node is automatically generated in the **XPath Evaluator**:

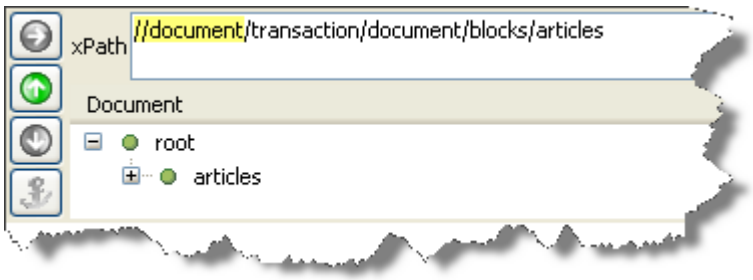


Figure 2 - 84: Generation of XPath to articles node

16 Click on **OK**.

The **Source** step property is automatically updated:

Figure 2 - 85: Source property automatically updated in Properties View

17 Save your project by clicking on  or by pressing `Ctrl + S`.

The *IfExist* step is now created and properly set.

WHAT COMES NEXT?

Now that the the presence of `articles` node in the XML output generated by the `GetArticleData` transaction has been tested thanks to the *IfExist* step, we will start by creating the *Complex* XML element, `<articlesList>`.

The procedure for creating the `articlesList` *Complex* step is close to the procedure for creating the `articleFound` *Complex* step (see *"To create and set a Complex step"* on page 2-36).

The main difference lies in the name of the complex element.

The following procedures describe how to create and set the `articlesList` *Complex* step.

To create the articlesList Complex step

1 To create the `articlesList` *Complex* step, follow the procedure for creating and

setting a *Complex* step (see *"To create and set a Complex step"* on page 2-36) based on the `IfArticlesTableExists` parent step.

*Before proceeding to the next step, make sure that the value of the **Node Name** property is correct (`articlesList`).*

After the step has been created, it appears in the sequence **Steps** folder:

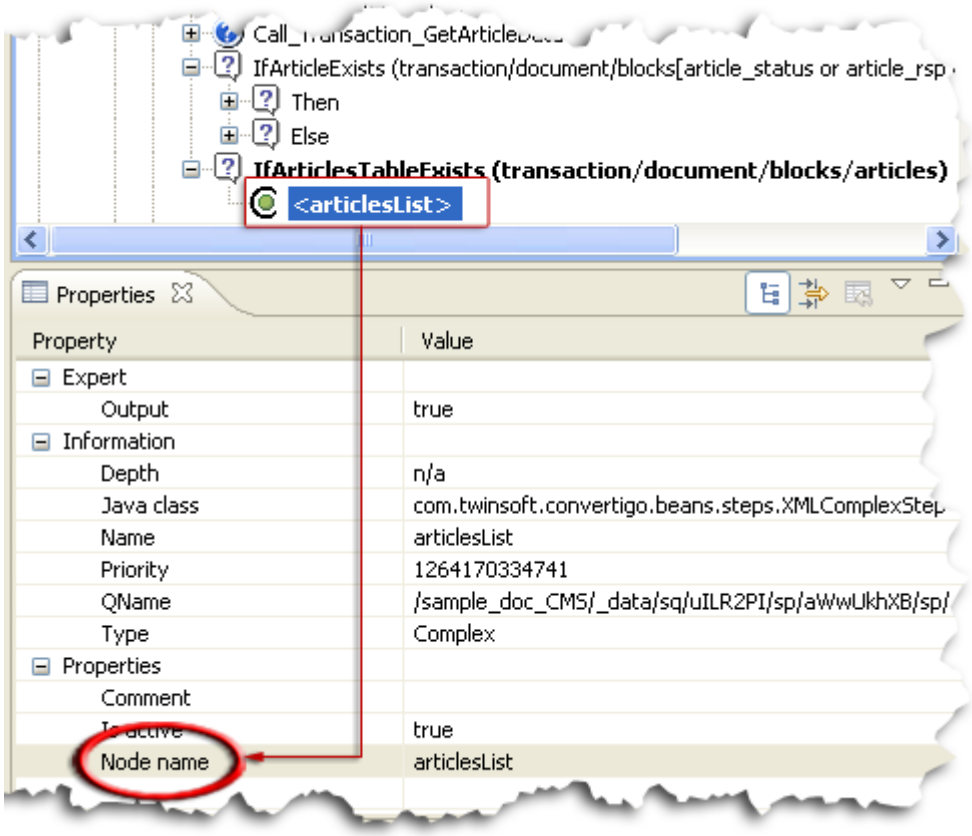


Figure 2 - 86: New Complex step and Node Name setting

2 Save your project by clicking on  or by pressing `Ctrl + S`.

The *Complex* step is now created and set to insert a complex `<articlesList>` XML node in the sequence XML output when required node is found in the `GetArticleData` transaction.

What comes next?

The following milestone in the project consists in iterating on each row of the `articles` XML table (for an example of `articles` XML table, refer to the *"Starting with Convertigo Legacy Integrator" Quick Guide*) to perform operations on these rows (insert row data into database table, call CWI transaction using row data, etc.).

To this end, we will now create and set an *Iterator* step.

The iteration must be performed ***only if the articles table exists***. This means that the *Iterator* step must appear as a child of the `IfArticlesTableExists` step.

To create and set an Iterator step

1 In the **Projects View**, right-click on the parent step (for example the `articlesList` step).

A contextual menu appears.

2 Select ***New > Step***.

A **New Step** wizard is automatically launched:

3 In the **Other Steps** area of the window, select **Iterator** with a left-click:

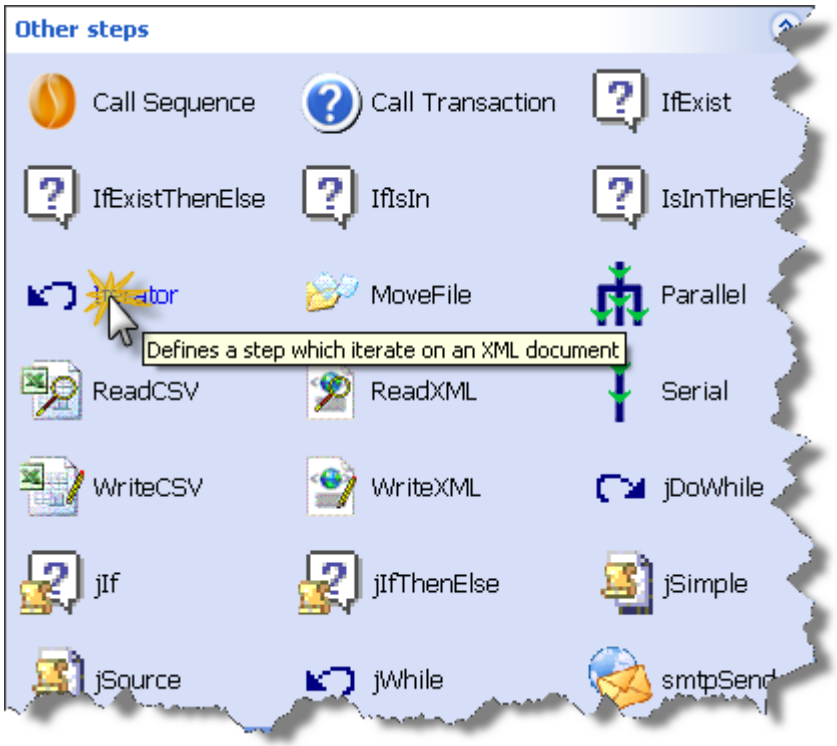


Figure 2 - 87: New Iterator step

4 Click on **Next**.

5 In the **Name** field that appears, type in the step name (for example `IteratorOnEachRow`):

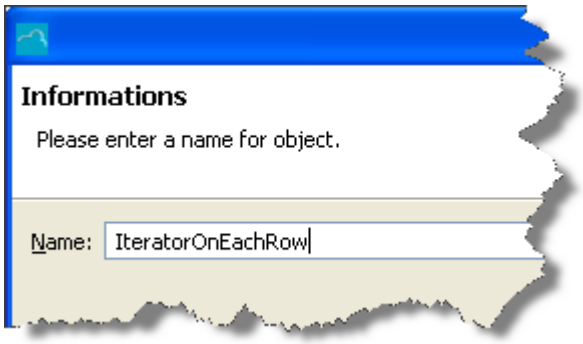


Figure 2 - 88: Giving a name to the step

6 Click on **Finish**.

The new step appears in the **Steps** folder of the project:

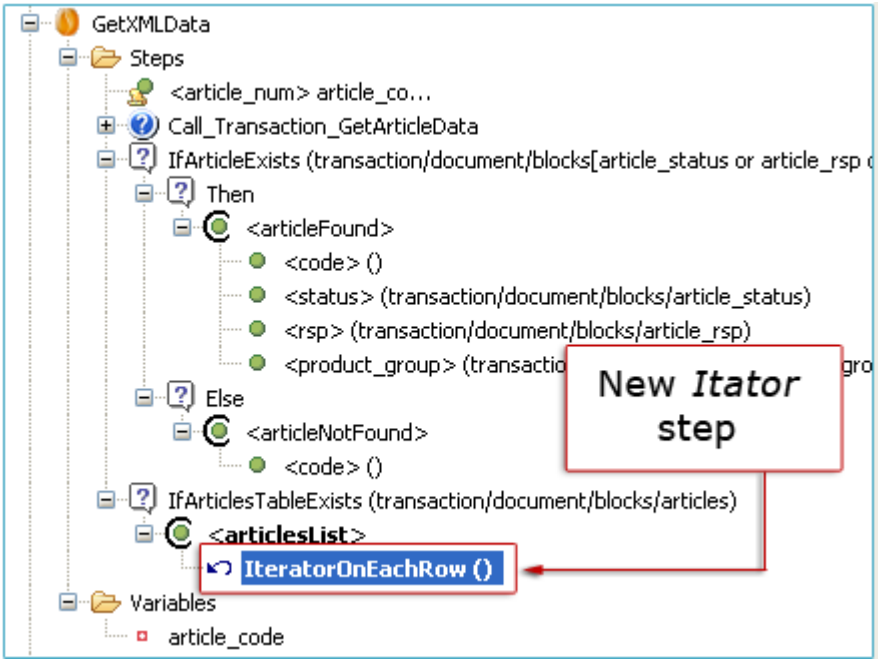


Figure 2 - 89: New Iterator step in Steps folder

7 Save your project by clicking on  or by pressing `Ctrl + S`.

Now that the *Iterator* step has been created, we will set its **Source** property so that it points towards the required node (`row`) of the `Call_Transaction_GetArticleData` step XML schema.

8 Select the *Iterator* step (for example `IteratorOnEachRow`) with a left-click.

The **Properties View** is automatically updated:

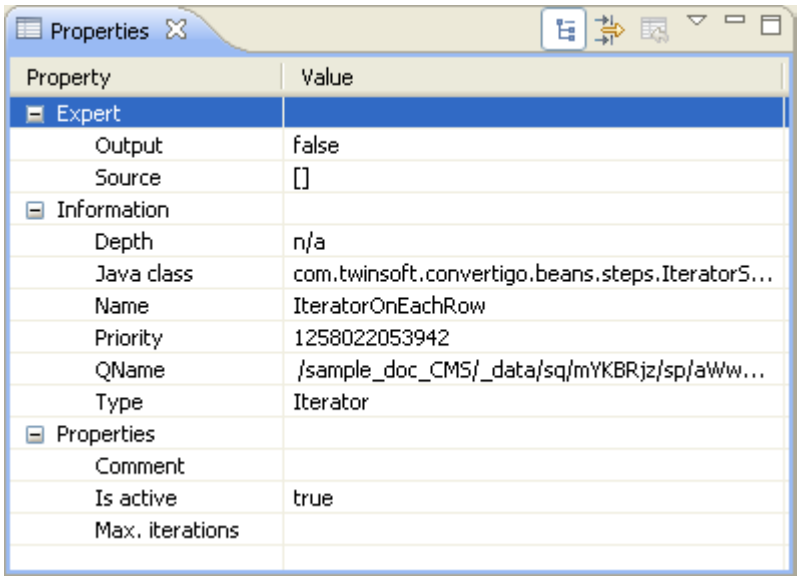


Figure 2 - 90: Iterator step properties

9 In the **Properties View**, click in the **Value** column of the **Source** property.

A  button appears.

10 Click on the  button.

The **Step Source** wizard is automatically launched.

11 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

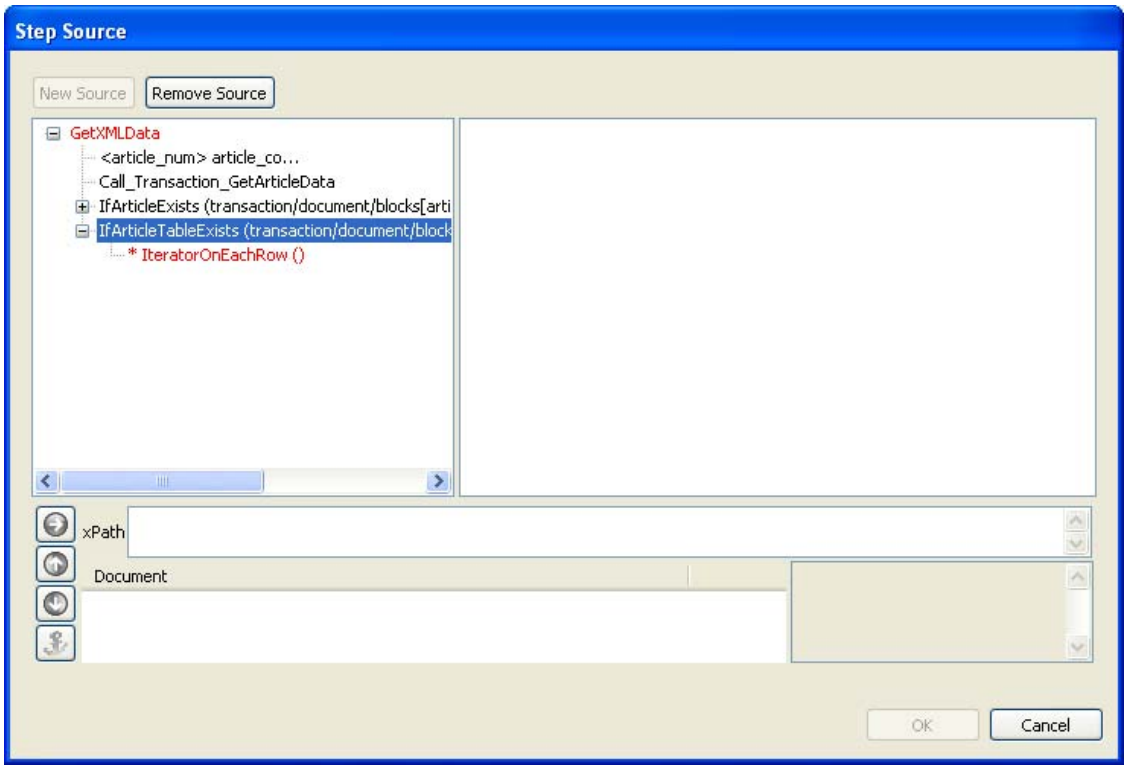


Figure 2 - 91: Generating iterator step source

The iteration must be performed on each `row` node of the `articles` XML table included in the output generated when calling the `GetArticleData` transaction. The step to be selected is therefore `Call_Transaction_GetArticleData`.

12 In the **Steps Tree Structure**, select the required step (for example `Call_Transaction_GetArticleData`) with a left-click.

The selected step's **XML Output Schema** is displayed:

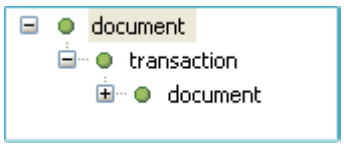


Figure 2 - 92: XML Schema of Call_transaction_GetArticleData step

13 Expand the `document`, `blocks` then `articles` nodes by clicking on ⊞:

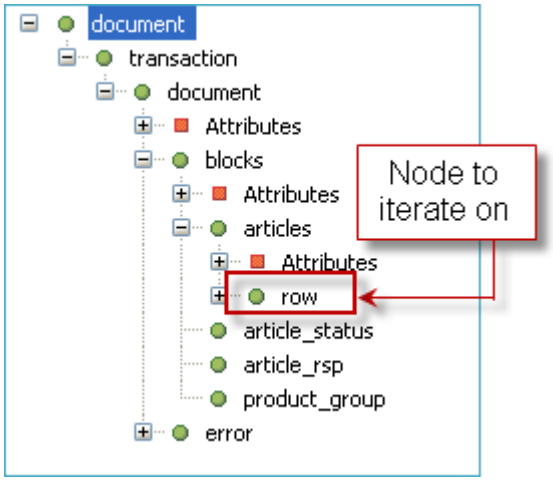


Figure 2 - 93: Node to iterate on in the called transaction XML output

14 In the **XML Output Schema**, select the `row` node.

One `row` node only appears in the **XML Output Schema** *because this is schema. When executing the `GetArticleData` transaction, several `row` nodes will be generated.*

15 The XPath expression to this node is automatically generated in the **XPath Evaluator**:

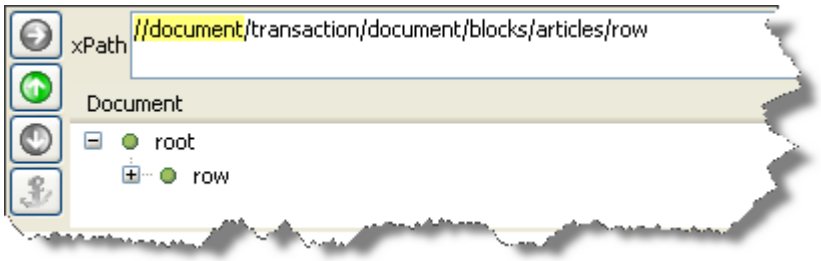


Figure 2 - 94: Generation of XPath to row node

16 Click on **OK**.

The **Source** step property is automatically updated:

Figure 2 - 95: Source property automatically updated in Properties View

17 Save your project by clicking on  or by pressing `Ctrl + S`.

The *Iterator* step is now created and properly set.

WHAT COMES NEXT?

Each `row` node is parent to three nodes: `code`, `name` and `status`.

The `IteratorOnEachRow` step therefore serves as source for child steps such as:

- `Call_Transaction_searchGoogle`, since this step variable must point through a source to the `name` node of ***iterated rows***, for the `searchGoogleWithLimit` transaction to use the content of this node as input variable (search keyword).

- the generated `article` element sub-steps, since this complex element includes three attributes (`article_code`, `article_name` and `article_status`), the values of which are sourced from the `code`, `name` and `status` nodes of ***iterated rows***.

The following step consists in creating the `Call_Transaction_searchGoogle` step.

This step is defined as follows:

- it is called for each article of the `articles` XML table generated by the `GetArticleData` transaction,
- it calls the `searchGoogleWithLimit` CWI transaction, in order to use its generated XML as source for further steps and to get more data about each article,
- it uses as input variables the transaction input variables, `keyword` and `maxPages` (these variables must be imported from the target transaction). The source of the `keyword` step variable must point to the `name` node of the row currently processed by the `IteratorOnEachRow` step (meaning that the `Call_Transaction_SearchGoogle` step is child to the `IteratorOnEachRow` step).

We will now create and set this step.

To create and set the `Call_Transaction_SearchGoogle` step

1. In the **Projects View**, right-click on the parent step (for example `IteratorOnEachRow`).

   A contextual menu appears.

2. Select ***New > Step***.

   A **New Step** wizard is automatically launched.

3. In the **Other Steps** area of the window, select **Call Transaction** with a left-click.

4. Click on **Next**.

5. In the **Name** field that appears, type in the step name (for example `Call_Transaction_SearchGoogle`):

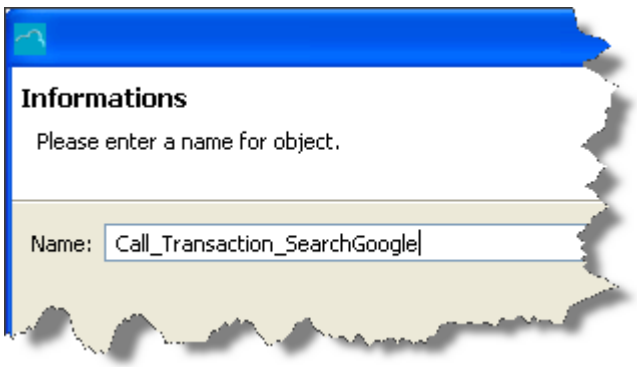


Figure 2 - 96: Giving a name to the step

6. Click on **Finish**.

   The new step appears in the **Steps** folder of the project:

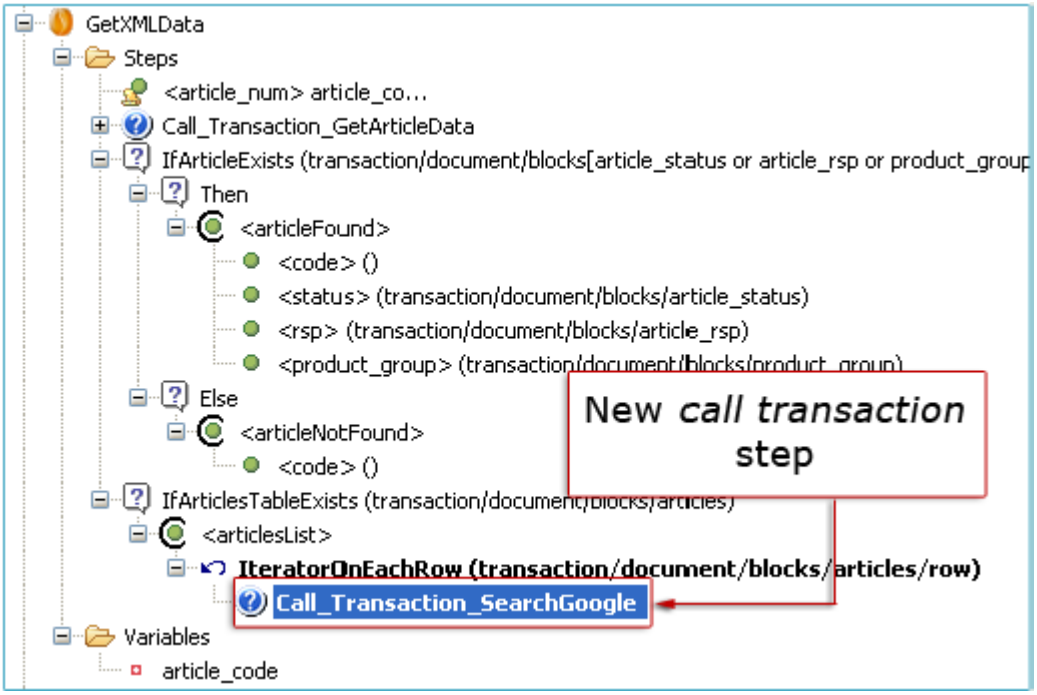


Figure 2 - 97: New Call Transaction step in Steps folder

Now that the *Call Transaction* step has been created, we will set its three main interrelated parameters:

- **Project** - in this case, the CWI project (`sample_documentation_CWI`),
- **Connector -** in this case, the `sample_documentation_CWI` project's sole connector (`GoogleConnector`),
- **Transaction -** in this case, the `searchGoogleWithLimit` transaction.

7 In the **Projects View**, select the step.

The **Properties View** is automatically updated.

8 Click in the **Value** column of the **Project** property.

The current value (`sample_doc_CMS`) is highlighted and a drop-down symbol appears.

9 Click on .

10 Select the project containing the transaction to be called (for example `sample_documentation_CWI` for the `searchGoogleWithLimit` transaction):

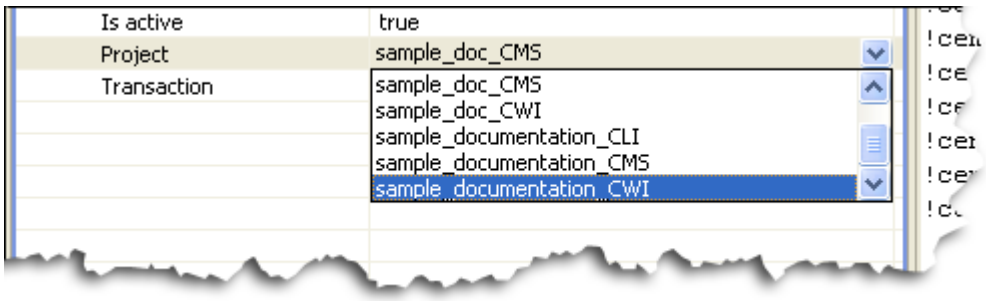


Figure 2 - 98: Setting the Project property of the Call Transaction step

11 Press `Enter`.

The values of the **Connector** and **Transaction** properties are updated.

12 Click in the **Value** column of the **Connector** property.

The current value is highlighted and a drop-down symbol appears.

13 Click on .

14 Select the connector containing the transaction to be called (for example `GoogleConnector` for the `searchGoogleWithLimit` transaction):

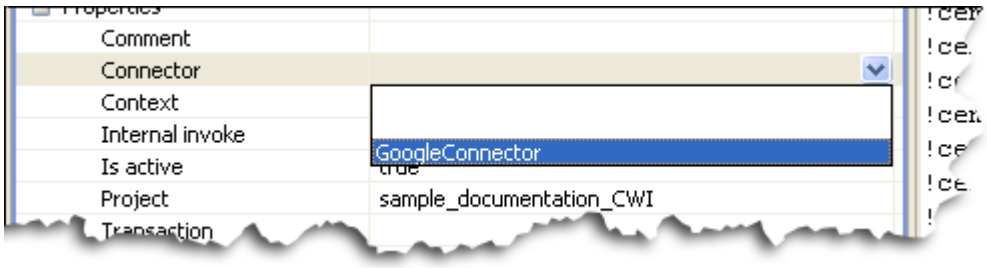


Figure 2 - 99: Setting the Connector property of the Call Transaction step

15 Press `Enter`.

The value of the **Transaction** property is updated.

16 Click in the **Value** column of the **Transaction** property.

The current value is highlighted and a drop-down symbol appears.

17 Click on .

18 Select the transaction (for example `searchGoogleWithLimit`):

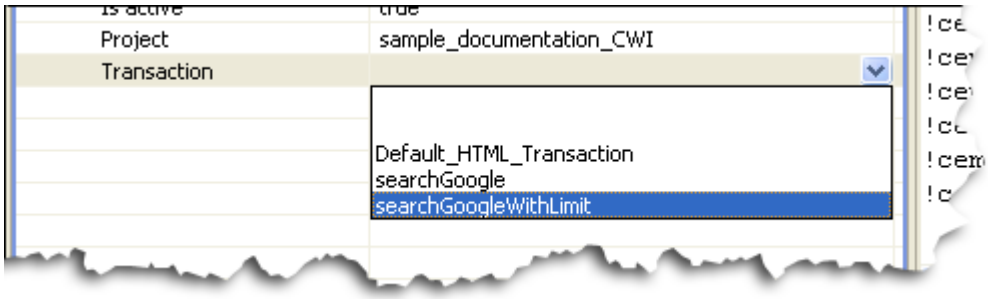


Figure 2 - 100: Setting the Transaction property of the Call Transaction step

19 Press `Enter`.

20 Save your project by clicking on or by pressing `Ctrl + S`.

The step is now set to call the `searchGoogleWithLimit` target transaction. Since the transaction expects a number of input variables, we will now import these variables from the `searchGoogleWithLimit` target transaction.

The procedure for importing variables has already been described. For more information, see "To import variables from a target transaction at step level" on page 2-22.

21 Right-click on the step.

A contextual menu appears:

22 Select **Import variables from target transaction**.

Once variables have been imported from the target transaction:

- the step appears as "changed and unsaved" (green bolded characters, see Figure 1 - 9),
- variables appear in a **Variables** sub-folder (together with the default value imported from the target transaction between brackets, when applicable):

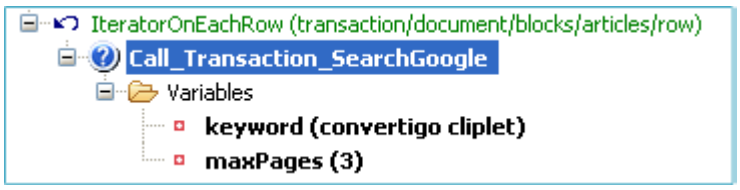


Figure 2 - 101: Variables imported from target transaction (and default values)

The *Call Transaction* step is now created and its variables are imported. These variables now need to be set.

The keyword variable source needs to point towards the `name` node of the `articles` XML table row currently processed by the `IteratorOnEachRow` step.

This source will now be set using the **Step Source** wizard.

To set the variable source of the `Call_Transaction_SearchGoogle` step using the Step Source wizard

1 Select the *Call Transaction* step `keyword` variable with a left-click.

The **Properties View** is automatically updated:

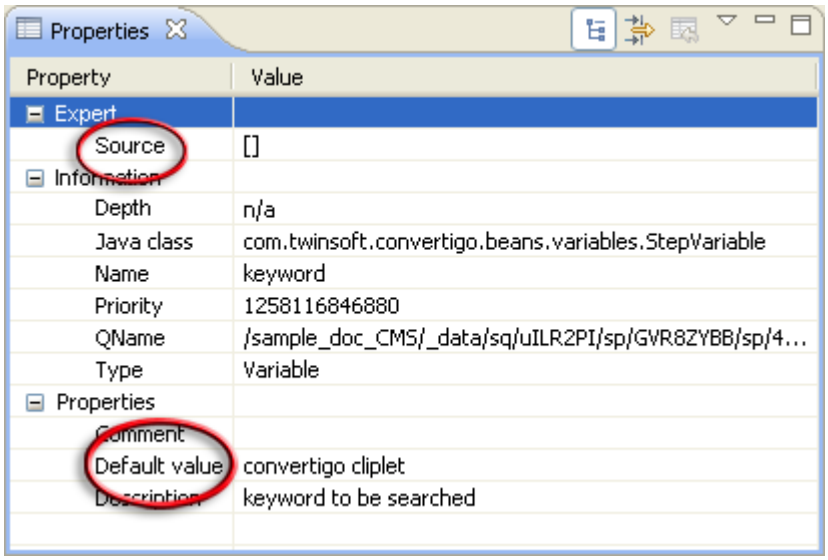


Figure 2 - 102: Properties of Call Transaction step keyword variable

2 In the **Properties View**, click in the **Value** column of the **Default Value** property.

The default value (imported from the transaction) is highlighted.

3 If applicable, delete the variable default value (because the variable value is sourced from the *Iterator* step).

4 Press `Enter`.

5 Click in the **Value** column of the **Source** property.

The value is highlighted and a button appears to the right of the field.

6 Click on the button.

The **Step Source** wizard is automatically launched.

For more information on the Step Source wizard, see Table "Step Source wizard description" on page 1-16.

7 Click on **New Source** in the upper left corner of the window.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

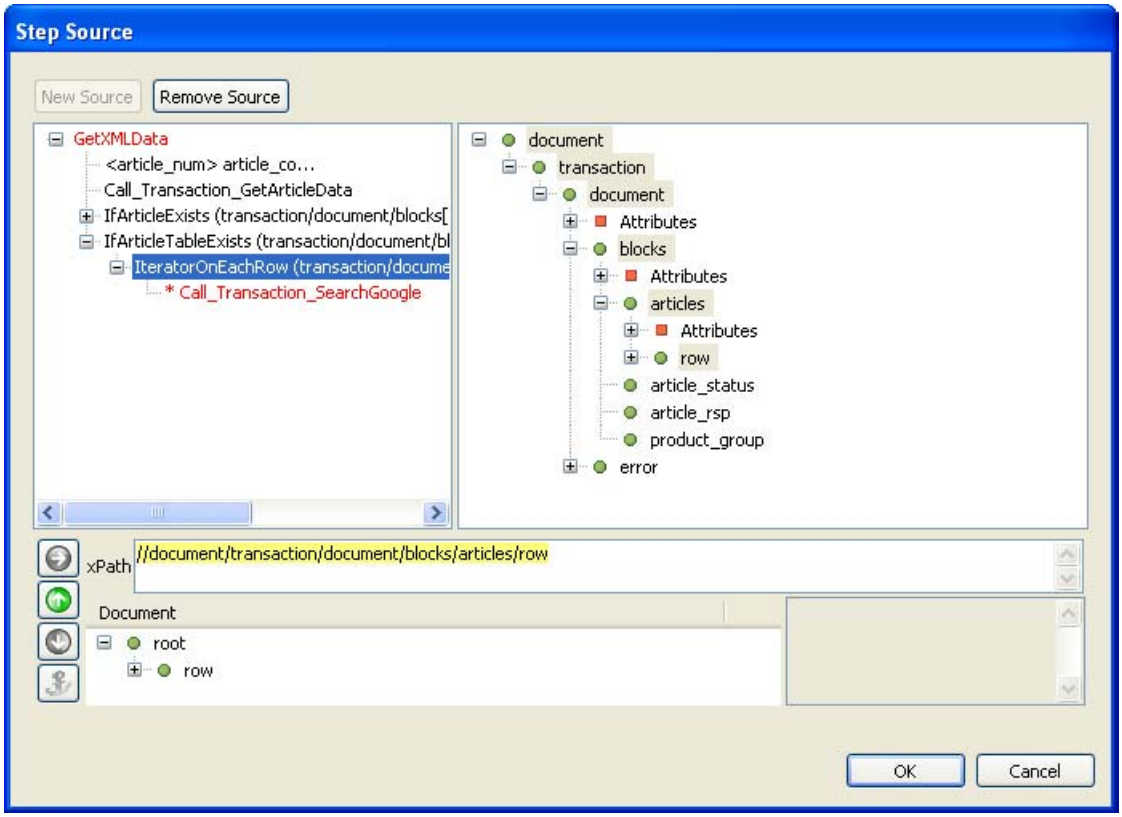


Figure 2 - 103: Setting of keyword variable source

The **Steps Tree Structure** displays all steps defined so far. The *Iterator* step is selected by default.

The `keyword` variable source must point towards the `name` node of the ***currently processed*** `row` node (ongoing iteration) in the `articles` XML table. The step selected by default (`IteratorOnEachRow`) is therefore correct, since its **XML Output Schema** corresponds to the currently processed row (source of the `IteratorOnEachRow` step).

In the **XML Output Schema**, the `row` node as well as its parent nodes are selected. This corresponds to the pregenerated *Iterator* step source XPath (highlighted in yellow in the **xPath** field of the **XPath Evaluator**).

8 In the **XML Output Schema**, expand the `row` node by clicking on ⊞:

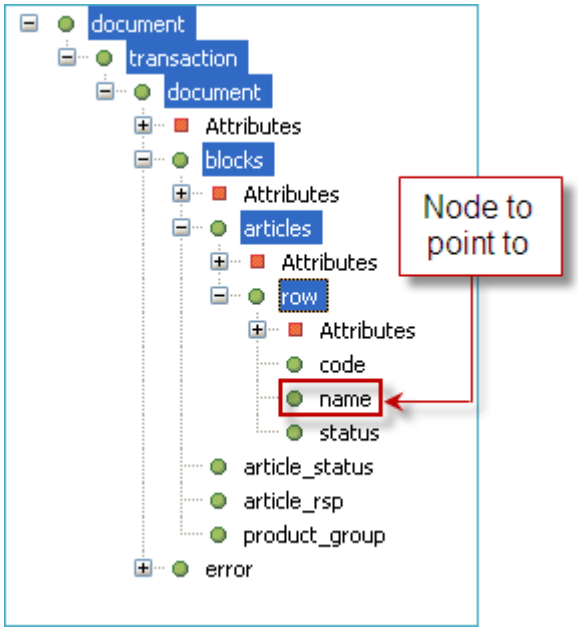


Figure 2 - 104: Source node for the keyword variable

9 Select the `name` node with a left-click.

The XPath to this node is automatically generated in the **XPath Evaluator**:

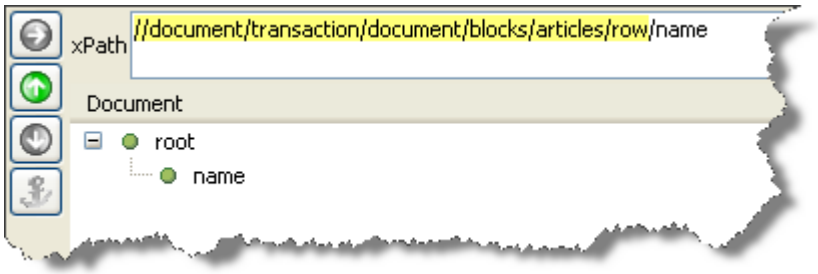


Figure 2 - 105: XPath to name node

The `keyword` variable source is now properly set.

10 Click on **OK**.

The variable source value is automatically updated in the **Properties View**.

We will now set the `maxPages` variable. Unlike other variables set so far, the `maxPages` variable can be given a fixed integer value. No source will be set for this variable, only a default value.

1 Select the *Call Transaction* step `maxPages` variable with a left-click.

The **Properties View** is automatically updated:

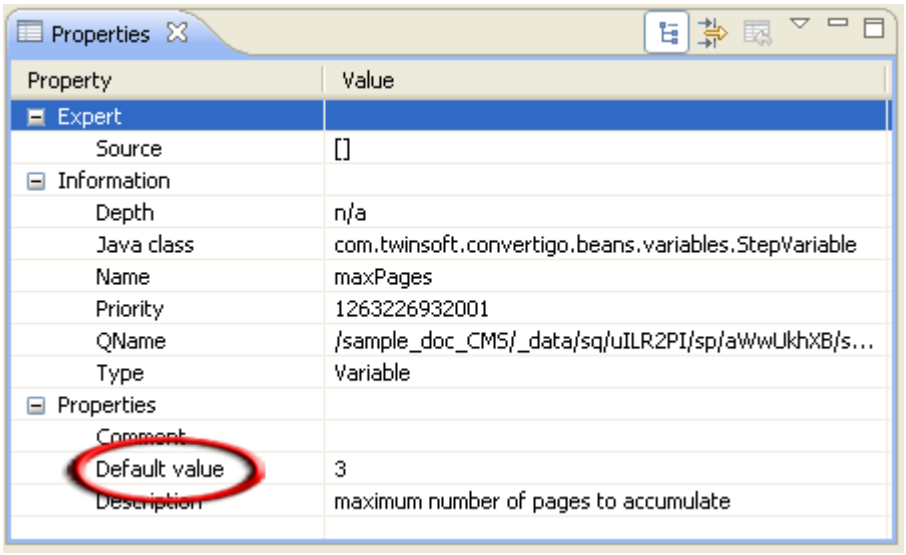


Figure 2 - 106: Properties of Call Transaction step maxPages variable

2 In the **Properties View**, click in the **Value** column of the **Default Value** property.

The default value (imported from the transaction) is highlighted.

3 Type in the required value (2 is a correct value in this project) or leave it unchanged.

4 Press `Enter`.

The variable is updated.

5 Save your project by clicking on  or by pressing `Ctrl + S`.

The `Call_Transaction_SearchGoogle` is now properly set to:

- call the `searchGoogleWithLimit` CWI transaction,
- enter as keyword the value of each `name` node returned by the `GetArticleData` transaction,
- limit the number of pages returned by Google to 2.

WHAT COMES NEXT?

When called, the `searchGoogleWithLimit` transaction generates an XML table defined as follows:

- the generated XML table is called `listResults`;
- it contains as many `resultItem` rows as result items returned by the Google search engine;
- each `resultItem` row has two columns: `title` and `url`.

For more information on the `searchGoogleWithLimit` transaction (which is similar to the `searchGoogle` transaction), refer to the "Starting with Convertigo Web Integrator" Quick Guide.

After the `searchGoogleWithLimit` transaction XML output is generated, we would like article mashuped information to appear in the sequence XML output, in the form of an

`article` complex XML element including:

- three attributes (`article_code`, `article_name` and `article_status`) which values are sourced from the `GetArticleData` transaction,
- and a list of complex elements (`resultItem`) which values are sourced from the `searchGoogleWithLimit` transaction.

To this end, we will now create the `article` *Complex* step then three *Attribute* steps set as children of the complex step.

The procedure for creating the `article` *Complex* step is close to the procedure for creating the `articleFound` *Complex* step and its four simple XML elements (see *"To create and set a Complex step"* on page 2-36 and see *"To create and set an Element step"* on page 2-38).

The main difference lies in the type of generated child XML elements:

- *attributes* in the former case.
- *elements* in the latter case.

The following procedures describe how to create and set the `article` *Complex* step and its three child *Attribute* steps.

To create the `article` Complex step and its attributes

1 To create the `article` *Complex* step, follow the procedure for creating and setting a *Complex* step (see *"To create and set a Complex step"* on page 2-36) based on the `IteratorOnEachRow` parent step.

*Before proceeding to the next step, make sure that the value of the **Node Name** property is correct (`article`).*

After the step has been created, it appears in the sequence **Steps** folder:

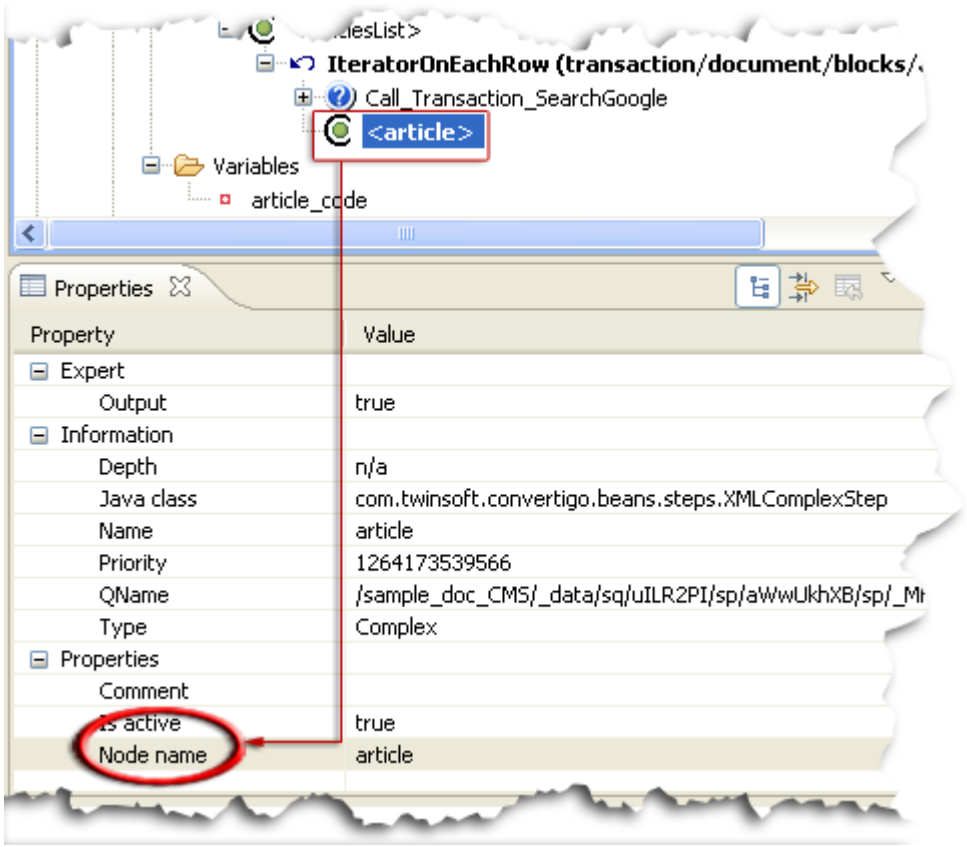


Figure 2 - 107: New Complex step and Node Name setting

2 Save your project by clicking on or by pressing `Ctrl + S`.

We will now create and set the attributes of the `article` complex element. Each attribute corresponds to one of the three columns of the `articles` XML table:

Table 2 - 9: Attributes of the `article` complex element

Attribute...	...corresponds to column...
`article_code`	`code`
`article_name`	`name`
`article_status`	`status`

As such, attribute sources must point to the nodes of the `GetArticleData` transaction XML ouput corresponding to these columns.

In the sequence XML output, we want one `article` complex element to be generated for *each row* of the `articles` XML table. This means that the content of attributes must be sourced from the `code`, `name` and `status` nodes of the `IteratorOnEachRow` step XML schema.

To create and set an Attribute step

1 Right-click on the parent step (for example `article`).

A contextual menu appears.

2 Select ***New > Step***.

A **New Step** wizard is automatically launched.

3 In the **Generating XML step** area, select **Attribute**:

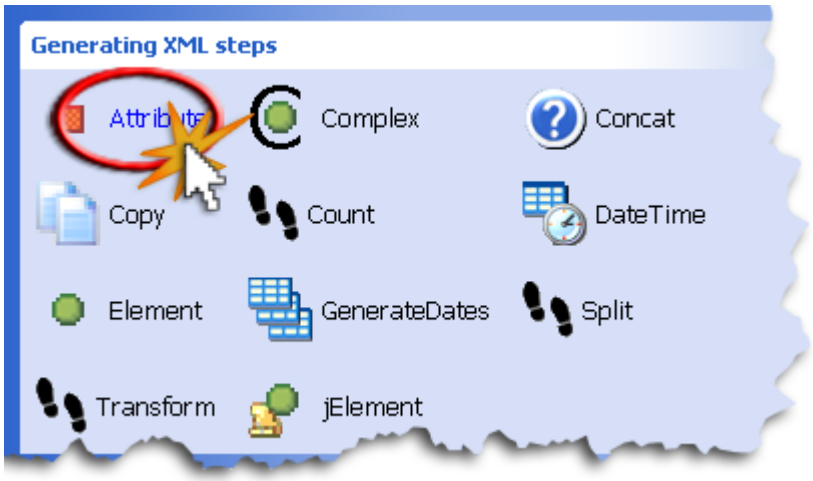


Figure 2 - 108: Selecting an Attribute step

An *Attribute* step generates an XML attribute.

4 Click **Next**.

5 In the **Name** field that appears, type in the *Attribute* name (for example `article_code`):

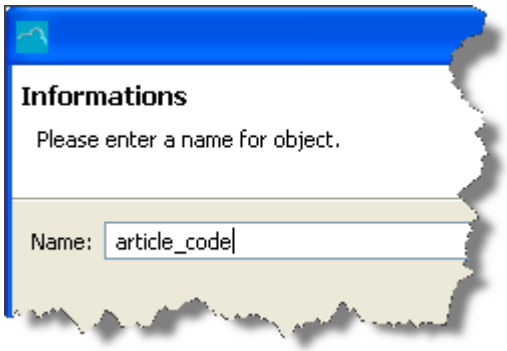


Figure 2 - 109: Entering the Element step name

6 Click on **Finish**.

The new step appears in the **Steps** folder of the project:

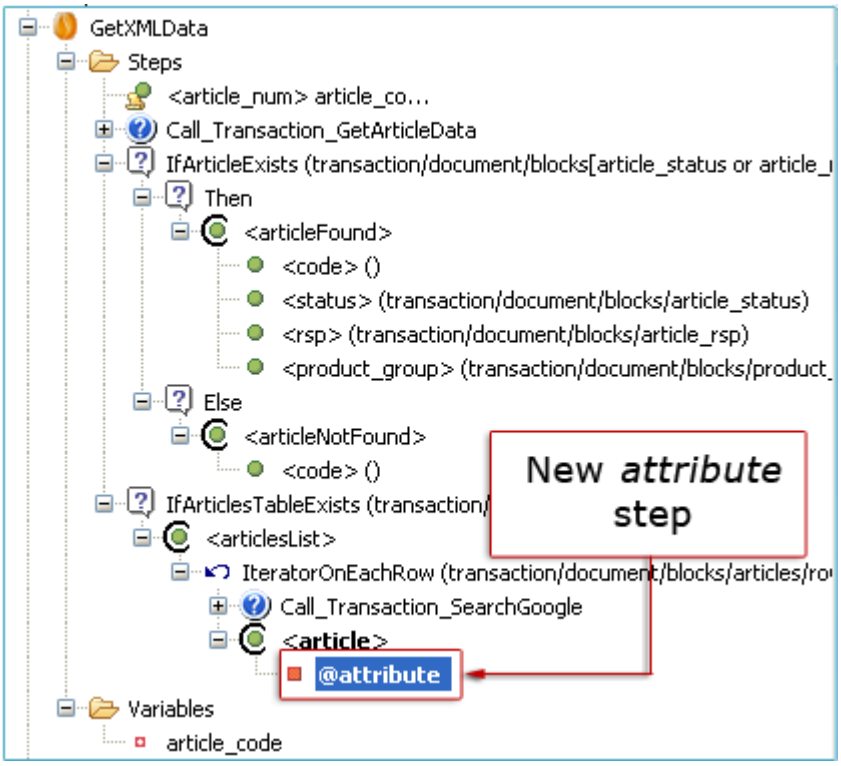


Figure 2 - 110: New Attribute step in Steps folder

7 Save your project by clicking on  or by pressing `Ctrl + S`.

Now that the *Attribute* step is created, we must set its two main properties: **Node Name** and **Source**.

The value of the **Node Name** property sets the name of the XML tag generated by the *Attribute* step (here, `article_code`).

As mentioned in Table *"Attributes of the `article` complex element"* on page 2-65, we would like the content of each `article_code` attribute to be sourced from each `code` node of the `articles` XML table.

To this end, we will set the *Attribute* step source so that it points to the `code` node of the `IteratorOnEachRow` step XML schema.

8 In the **Projects View**, select the *Attribute* step with a left-click.

The **Properties View** is automatically updated:

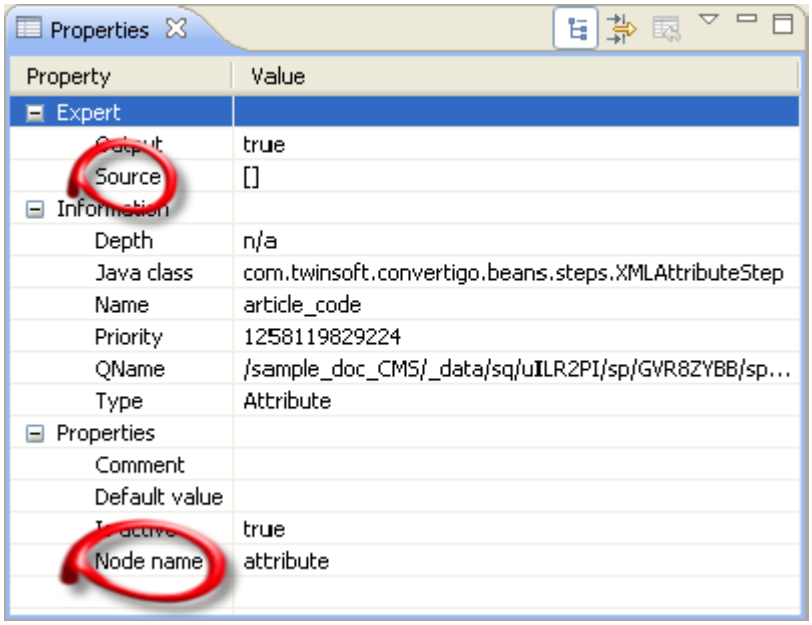


Figure 2 - 111: Attribute step properties

9 Click in the **Value** column of the **Node Name** property.

The value is highlighted.

10 Enter the required node name (for example `article_code`).

11 Press `Enter`.

The value is updated: 

In the **Projects View**, the step display name has changed from `@attribute` (default **Node name** value) to `@article_code`:

Figure 2 - 112: Attribute node name updated in Projects View

12 In the **Properties View**, click in the **Value** column of the **Source** property.

A  button appears.

13 Click on the  button to launch the **Step Source** wizard.

The **Step Source** wizard is automatically launched.

14 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

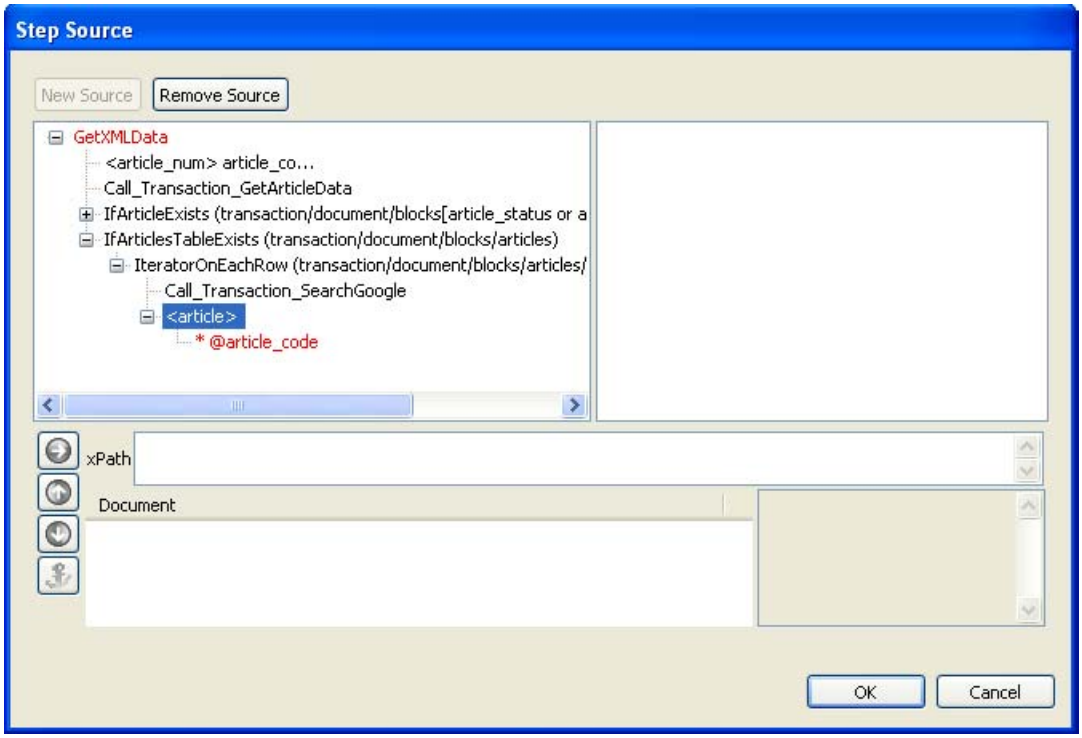


Figure 2 - 113: Setting of Attribute step source

The steps defined so far appear in the **Steps Tree Structure**.

15 In the **Steps Tree Structure**, select the `IteratorOnEachRow` step to display its XML schema.

The selected step's **XML Output Schema** is automatically displayed.

16 Expand the row node by clicking on ⊞.

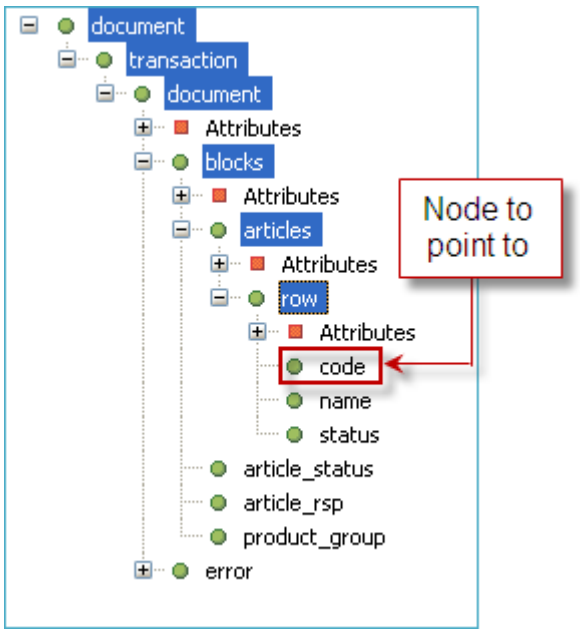


Figure 2 - 114: XML Output Schema of IteratorOnEachRow step

17 Select the `code` node with a left-click.

The XPath to this node is automatically generated in the **xPath** field of the **XPath**

Evaluator and the result of this XPath on the XML schema is displayed in the **Document** tree structure:

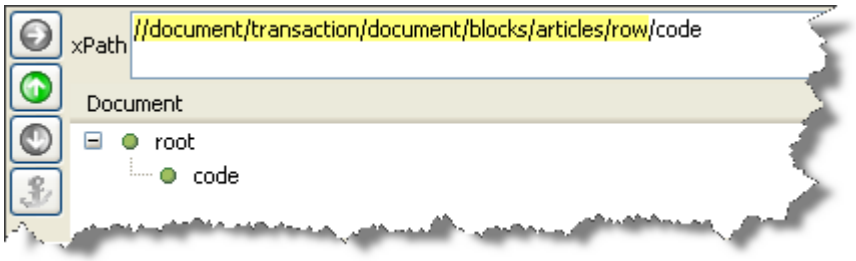


Figure 2 - 115: Generation of XPath to code node of iterated row in articles XML table

18 Click on **OK**.

The *Attribute* step is now created and set to generate an `article_code` XML attribute within each generated `article` complex XML element in the sequence XML output.

We will now set the remaining two attributes of `article` complex elements: `article_name` and `article_status`.

19 To set the two remaining attributes, repeat this procedure and set attribute sources so that they point towards required nodes of the `IteratorOnEachRow` step XML schema : name and `status` (see Figure 2 - 114).

*You can also copy the previously created step (`article_code`) and paste it as child of the Complex element (`article`), then set the **Node name** and **Source** properties of pasted steps as required.*

20 Save your project by clicking on  or by pressing `Ctrl + S`.

After all three attributes have been created and set, they appear in the sequence **Steps** folder as follows:

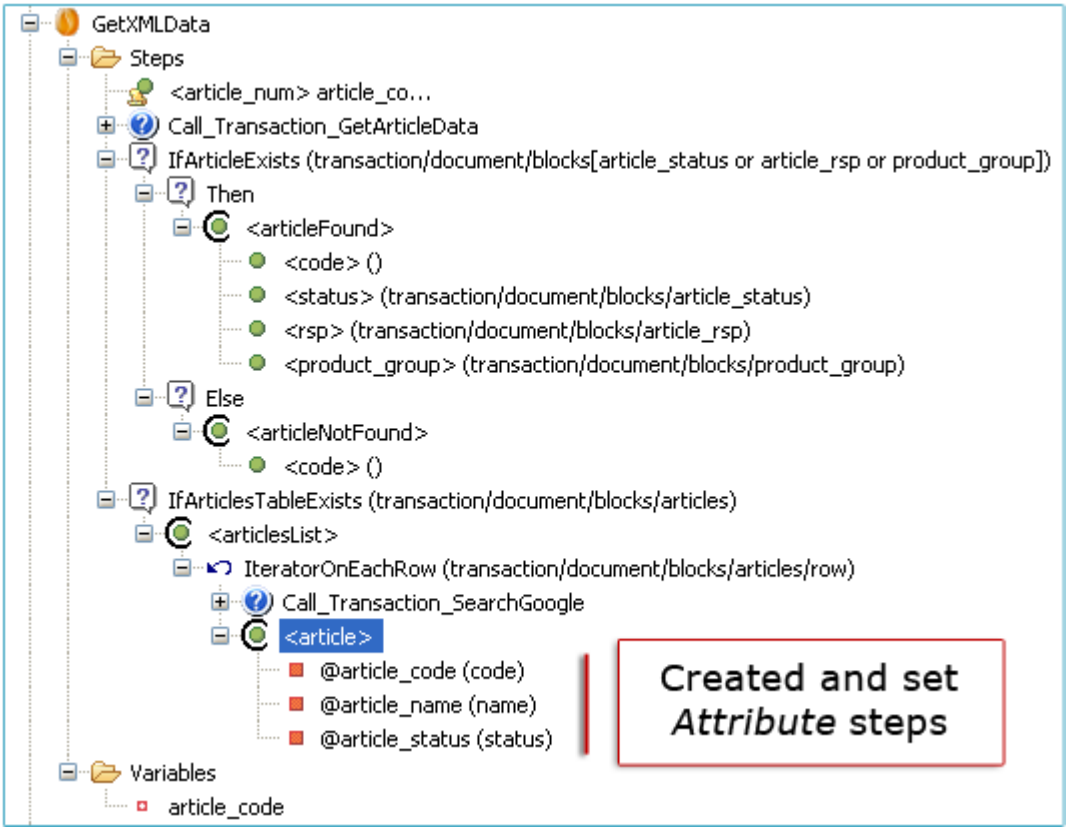


Figure 2 - 116: Attribute steps created and set as children of the Complex article step

The `article` *Complex* step and its three attributes are now created and set.

WHAT COMES NEXT?

The final milestones of the sequence consist in:

- *Checking* whether the `searchGoogleWithLimit` transaction (called by the `Call_Transaction_SearchGoogle`) generated a `listResults` XML table including `resultItem` nodes and,
- if so, *copying* the content of `resultItem` nodes into the sequence XML as part of `article` complex element.

This final setting enables us to include in the sequence XML output articles together with their code, name and status, followed by result items of a Google search based on the article name.

To check the presence of `resultItems` nodes in the `searchGoogleWithLimit` transaction XML output, we will now create and set an *IfExist* step called `IfResultItemsExist`.

The following procedure is similar to the previously described procedure for creating and setting the `IfArticlesTableExists` step (see "To create and set an IfExist step" on page 2-47). Only the step name and source change. For documentation lightness purpose, the procedure is repeated here with a limited number of snapshots.

To create and set the `IfResultItemsExist` step

1 Right-click on the parent step (here, the `article` step).

A contextual menu appears.

2 Select ***New > Step***.

The **New Step** wizard is automatically launched.

3 In the **Other steps** area, select **IfExist**.

4 Click on **Next**.

5 In the **Name** field that appears, type in the *IfExist* step name (for example `IfResultItemsExist`):

Figure 2 - 117: Entering the name of the fExist step

6 Click on **Finish**.

The new step appears at the end of the **Steps** folder in the **Projects View**:

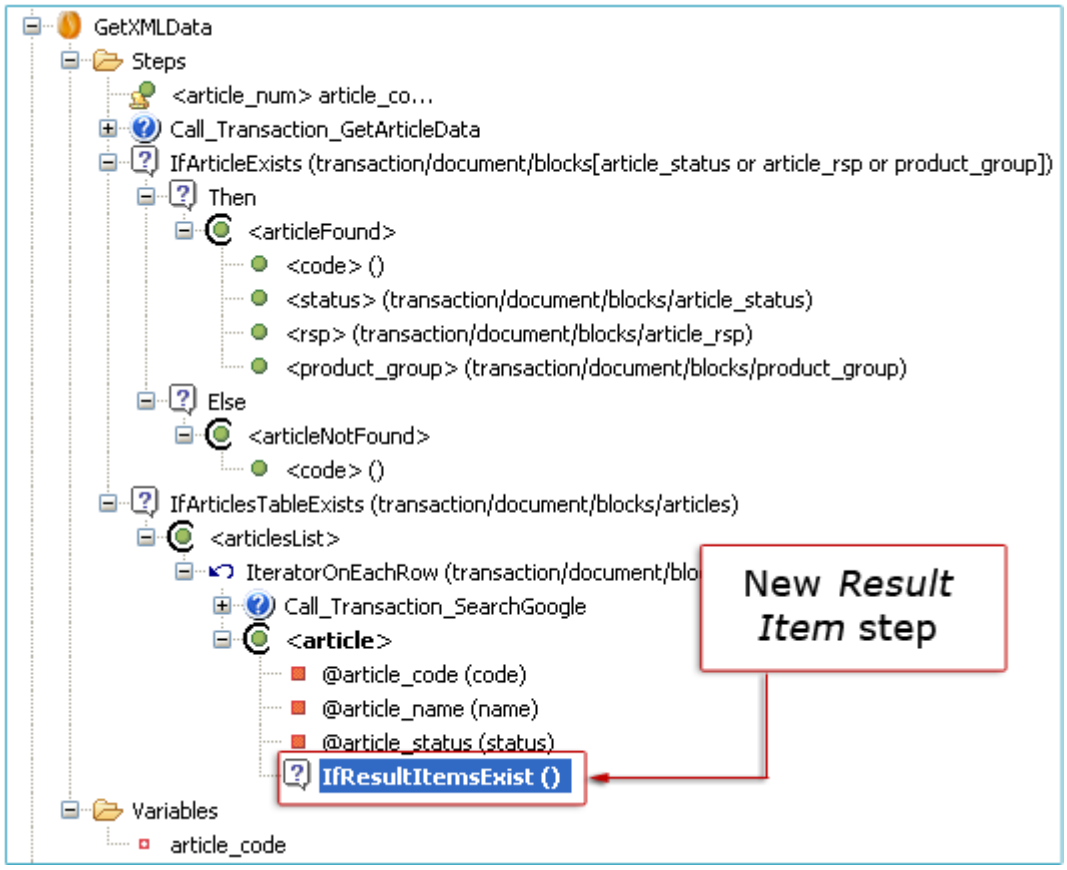


Figure 2 - 118: New IfExist step in Steps folder

7 Save your project by clicking on or by pressing `Ctrl + S`.

Now that the *ifExist* step has been created, its source must be set.

The step source needs to point towards the XML node (`resultItems`) to be checked within the XML output returned by the `searchGoogleWithLimit` transaction (called by the `Call_Transaction_searchGoogle` step).

We will now set the step source.

8 Select the *IfExist* step with a left-click.

The **Properties View** is automatically updated.

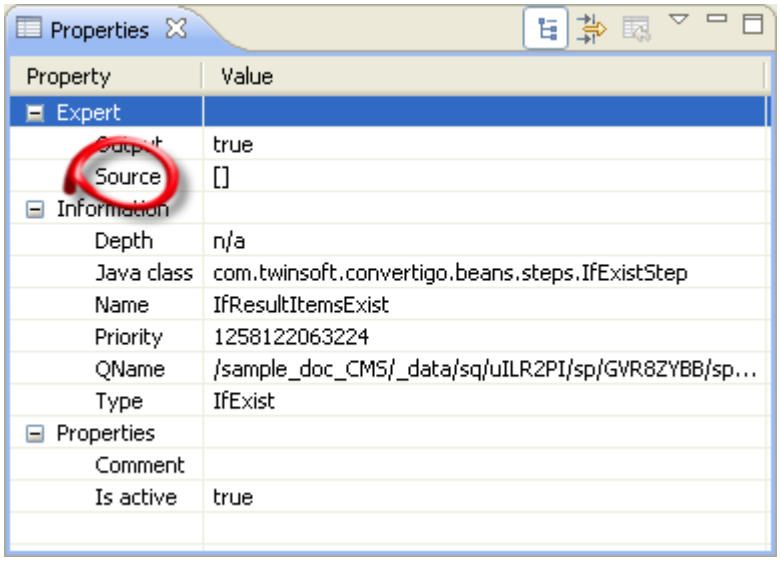


Figure 2 - 119: IfExist step properties

9 Click in the **Value** column of the **Source** property.

A button appears.

10 Click on the button to launch the **Step Source** wizard.

The **Step Source** wizard is automatically launched.

11 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

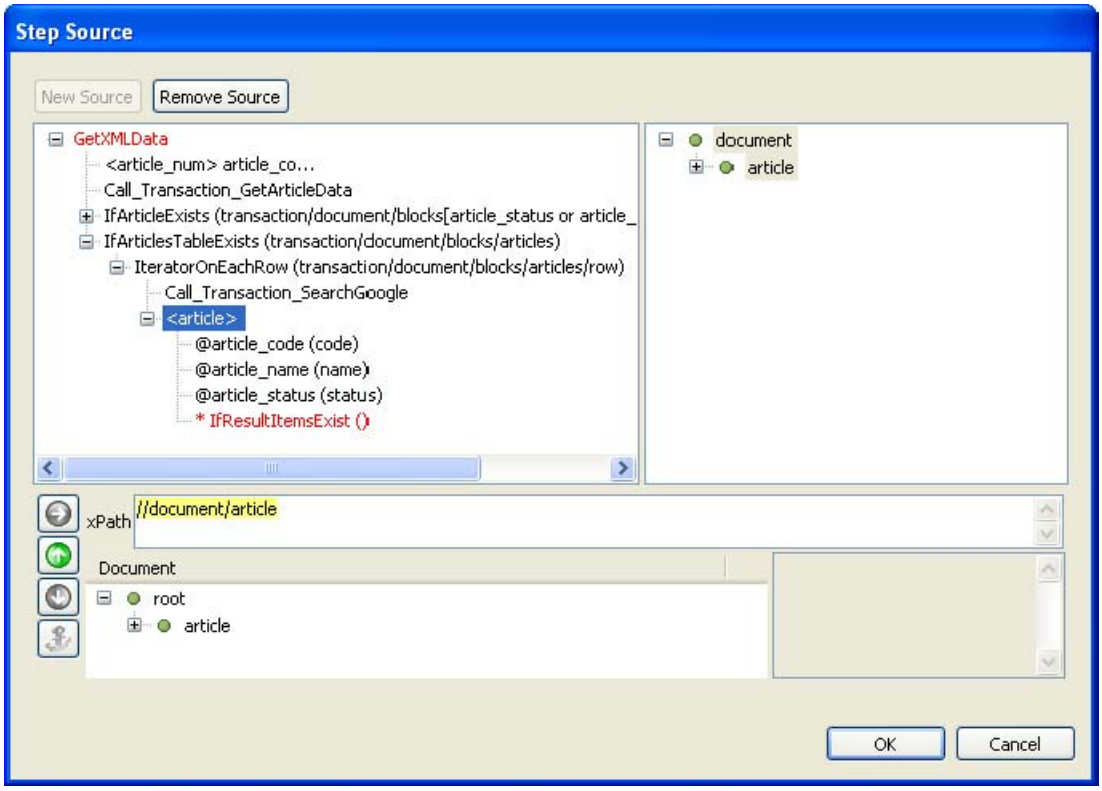


Figure 2 - 120: Setting of IfResultItemsExist step source

The *IfExist* check must be performed on the XML schema of the output generated when calling the `searchGoogleWithLimit` transaction. The step to be selected is therefore `Call_Transaction_searchGoogle`.

12 In the **Steps Tree Structure**, select the required step (for example `Call_Transaction_searchGoogle`) with a left-click.

The selected step's **XML Output Schema** is displayed:

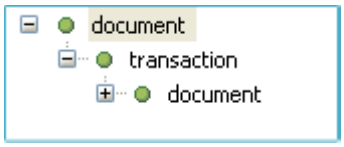


Figure 2 - 121: XML Schema of Call_transaction_SearchGoogle output

13 Expand the `document`, then `listResults` node including the `resultItem` node to be checked by clicking on ⊞:

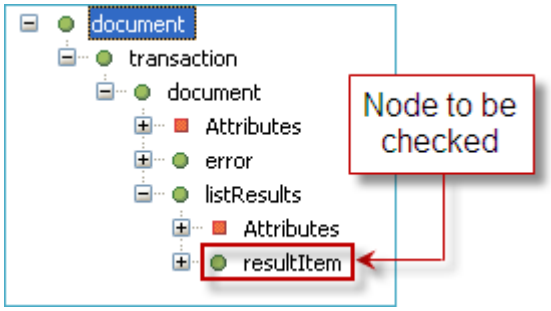


Figure 2 - 122: Node to check in the called transaction XML output

14 In the **XML Output Schema**, select the `resultItem` node with a left-click.

The XPath expression to this node is automatically generated in the **XPath Evaluator**:

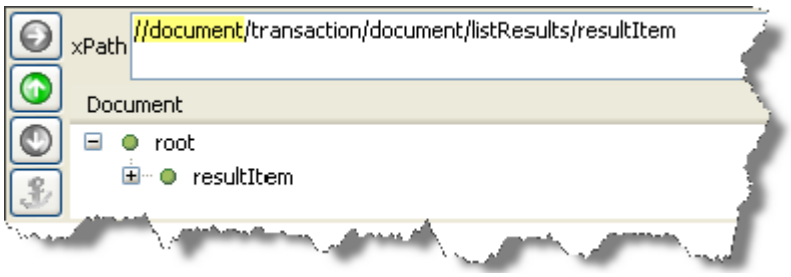


Figure 2 - 123: Generation of XPath to resulItems node

15 Click on **OK**.

The **Source** step property is automatically updated:

Figure 2 - 124: Source property automatically updated in Properties View

16 Save your project by clicking on  or by pressing `Ctrl + S`.

The *IfExist* step is now created and properly set.

WHAT COMES NEXT?

The following milestone in the project consists in copying the google search result items included in `resultItems` nodes into the sequence XML output.

To this end, we will now create and set a *Copy* step called `CopyOf`.

Copy steps copy the content of XML nodes and the content of their children nodes into the XML output of the sequence of which they are part.

To create and set a Copy step

1 Right-click on the parent step (here, the `IfResultItemsExist` step).

A contextual menu appears.

2 Select ***New > Step***.

The **New Step** wizard is automatically launched.

1 In the **Generating XML step** area, select **Copy**:

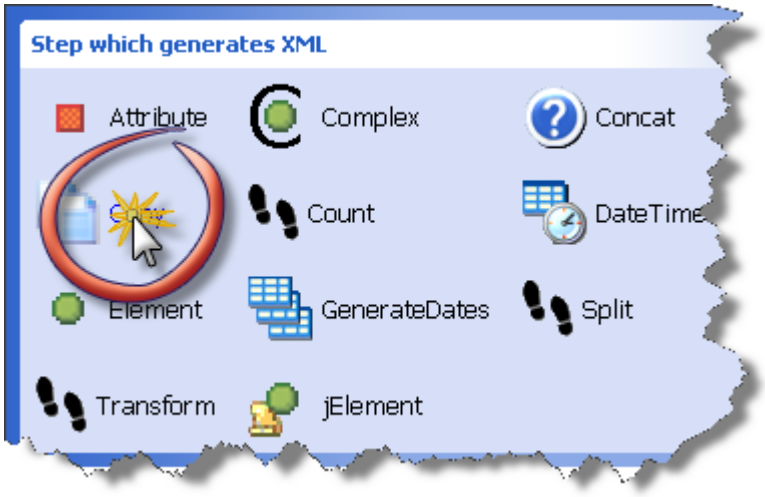


Figure 2 - 125: Selecting a Copy step

2 Click on **Next**.

3 In the **Name** field that appears, type in the *Copy* step name (for example `CopyResultItems`):

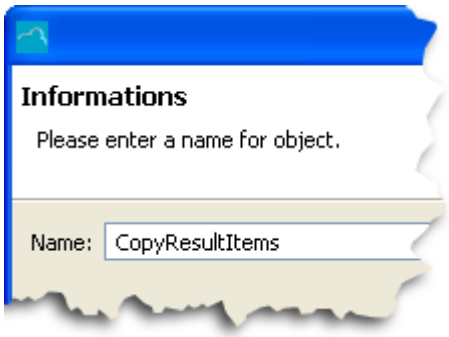


Figure 2 - 126: Entering the name of the Copy step

4 Click on **Finish**.

The new step appears at the end of the **Steps** folder in the **Projects View**:

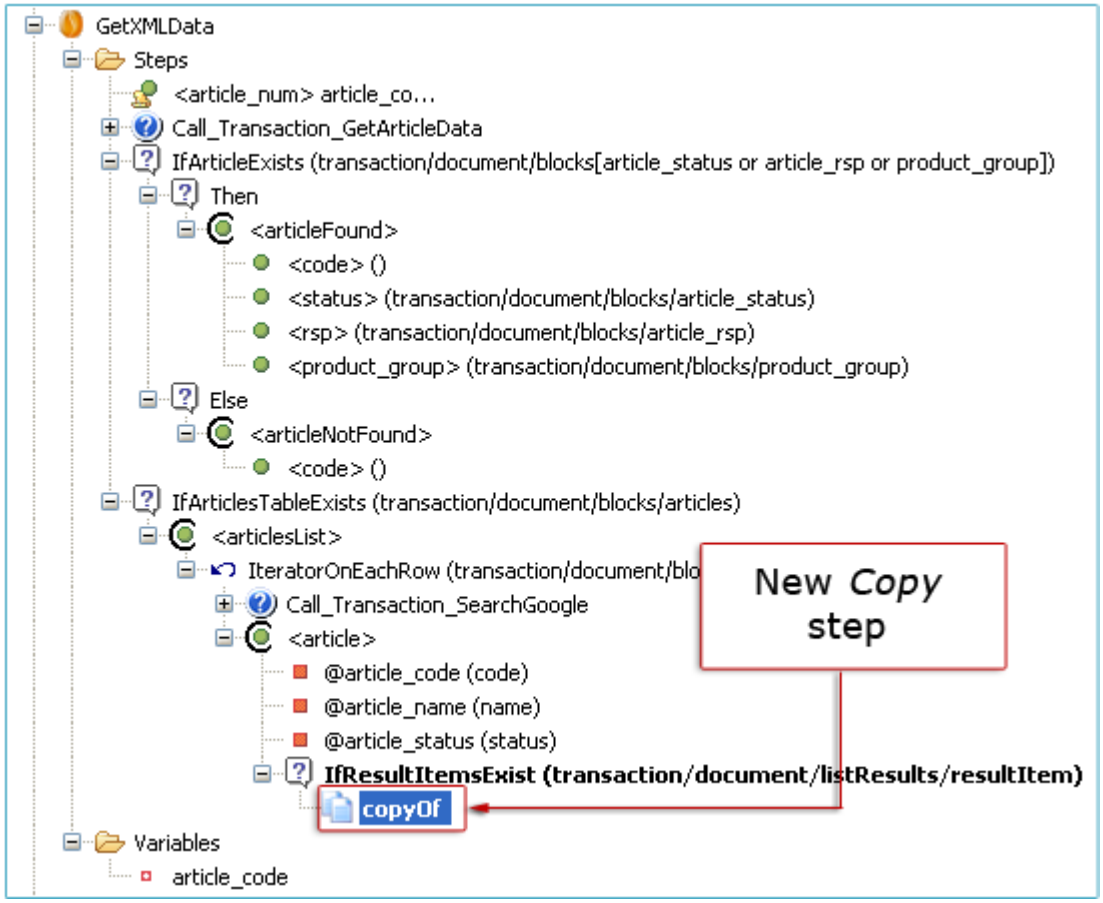


Figure 2 - 127: New Copy step in Steps folder

5 Save your project by clicking on  or by pressing `Ctrl + S`.

Now that the *Copy* step has been created, its source must be set.

This will be done through the **Source** property.

6 In the **Projects View**, select the *Copy* step with a left-click.

The **Properties View** is automatically updated:

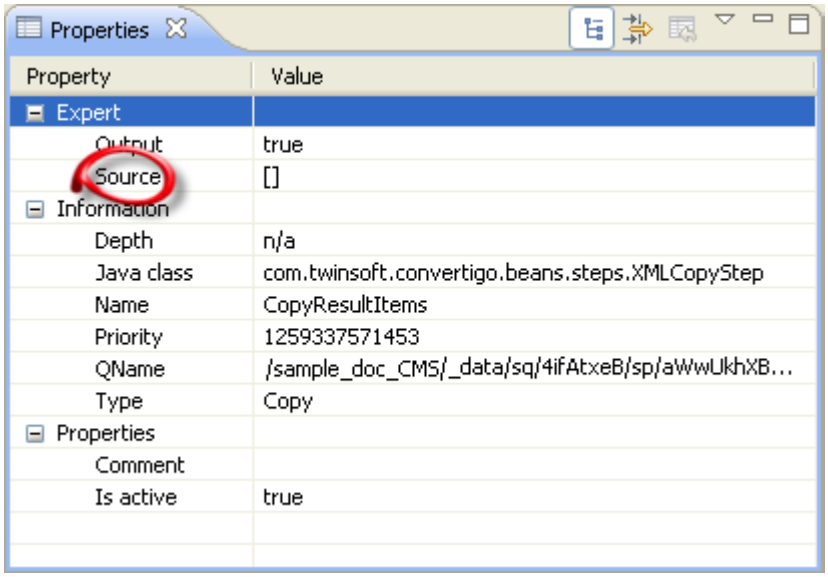


Figure 2 - 128: Copy step properties

7 Click in the **Value** column of the **Source** property.

A [...] button appears.

8 Click on the [...] button to launch the **Step Source** wizard.

The **Step Source** wizard is automatically launched.

9 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

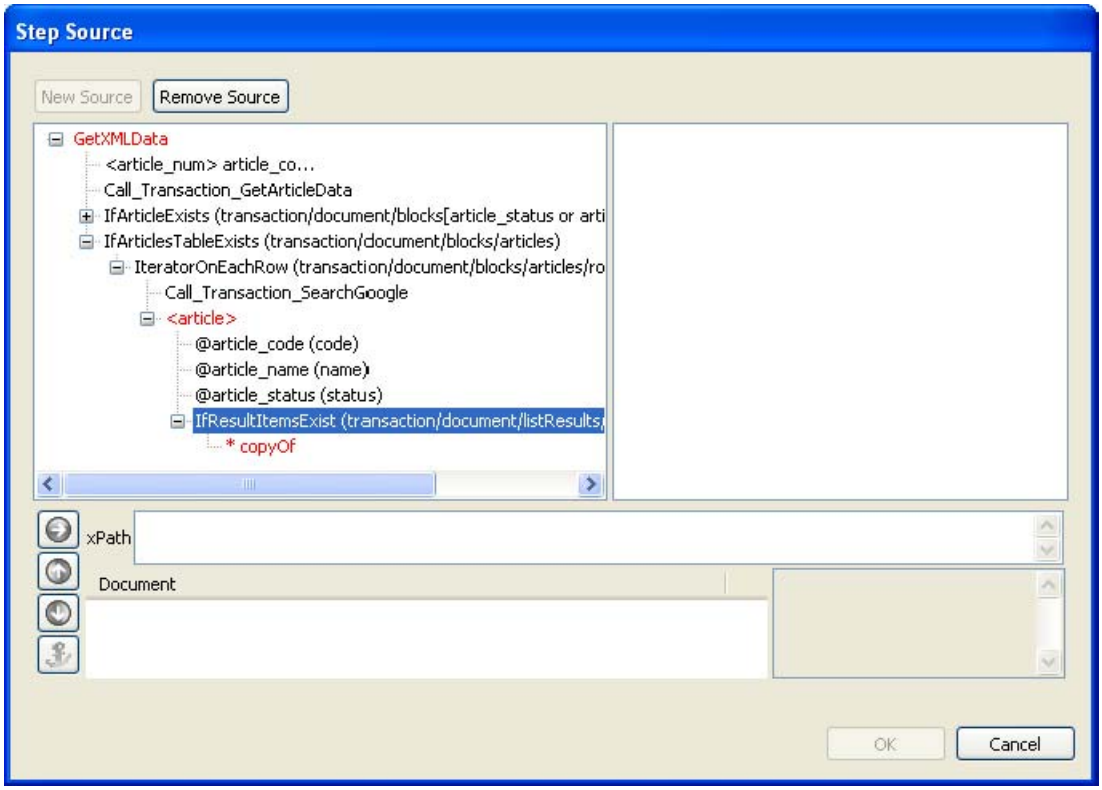


Figure 2 - 129: Setting of a Copy step source

The steps defined so far appear in the **Steps Tree Structure**.

We will set the `CopyOf` step source so that it points to the `resultItem` node of the `Call_Transaction_SearchGoogle` step XML schema.

As a result, all `resultItem` nodes (and associated child nodes) of the `searchGoogleWithLimit` transaction XML output - which is generated for each row of the `articles` XML table, based on the name tag included in each row - are copied.

10 In the **Steps Tree Structure**, select the `Call_Transaction_SearchGoogle` step to display its XML schema.

The selected step's **XML Output Schema** is displayed:

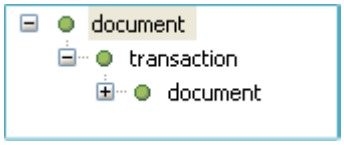


Figure 2 - 130: XML Schema of Call_transaction_SearchGoogle output

11 Expand the `document`, then `listResults` node including the `resultItem` node to be copied by clicking on ⊞:

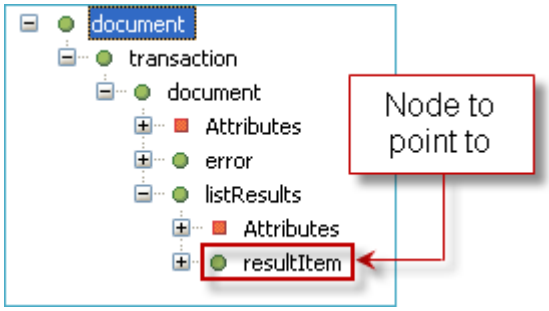


Figure 2 - 131: Source of CopyOf step

12 In the **XML Output Schema**, select the `resultItem` node with a left-click.

The XPath expression to this node is automatically generated in the **XPath Evaluator**:

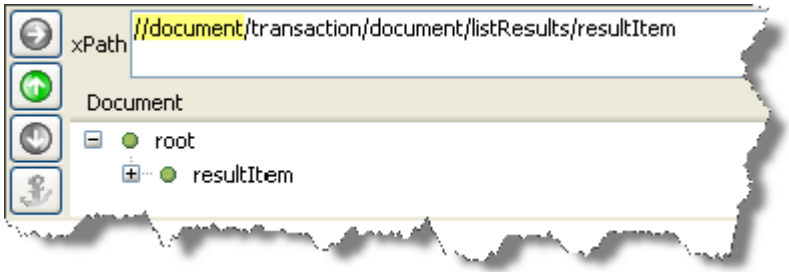


Figure 2 - 132: Generation of XPath to resulItems node

13 Click on **OK**.

The **Source** step property is automatically updated:

Figure 2 - 133: Source property automatically updated in Properties View

14 Save your project by clicking on  or by pressing `Ctrl + S`.

The *Copy* step is now created and properly set.

WHAT COMES NEXT?

The setting up of the sequence is now complete and the sequence is functional.

We can now execute the sequence and analyze results (see *"Executing a Sequence"* on page 2-138 and *"Analyzing the First Sequence XML Output"* on page 2 - 149).

2.5 Developing the Second Sequence

This section describes how to develop the second sequence or, in other words, how to set all steps and SQL transactions needed in the second sequence:

- First and Second Sequence - Redundant Project Settings
- Starting with the Second Sequence

2.5.1 First and Second Sequence - Redundant Project Settings

According to the second sequence description (see *"Second sequence description"* on page 2-5), one of the first steps of the second sequence consists in calling the `GetArticleData` CLI transaction, in order to use its generated XML as source for further steps.

This has already been done in the first sequence.

The first and second sequences actually share a number of similar project settings:

Table 2 - 10: First and second sequence similar settings

N.	Setting	Procedure
1	Creating the sequence	To create a new sequence variable
2	Creating the sequence variable	To create a new sequence variable
3	Creating and setting a *Call Transaction* step, `Call_Transaction_ GetArticleData` step.	To create and set a Call transaction step
4	Importing variables from the `GetArticleData` transaction	To import variables from a target transaction at step level
5	Creating and setting a JElement step serving as source for the Call_Transaction_GetArticleData step input variable	To create and set a jElement step serving as source for a step variable
6	Increasing the *jElement* step priority	To increase a step priority
7	Setting the *jElement* generated XML node as source for the `Call_Transaction_GetArticleData` step variable	To set a step's generated XML node as source for another step's variable
8	Creating and setting a step for checking the presence of required nodes in the XML output generated by the `Call_Transaction_GetArticleData` step[a]	To create and set an IfExistThenElse step

a. *In the context of the first sequence, setting number 8 requires the creation and setting of an IfExistThenElse step whereas it is an IfExist step in the second sequence. These steps have the same functionality in both sequences and the setting of their* ***Source*** *property is identical.*

2.5.2 Starting with the Second Sequence

To start with the setting the second step, please follow carefully procedures 1 to 7 of Table *"First and second sequence similar settings"* on page 2-80, for the second sequence named `InsertDataInBase`.

Once you are done with initial settings, pass on to the next procedure.

Before passing on to the next procedure, make sure that the sequence **Steps** folder contains the following two steps:

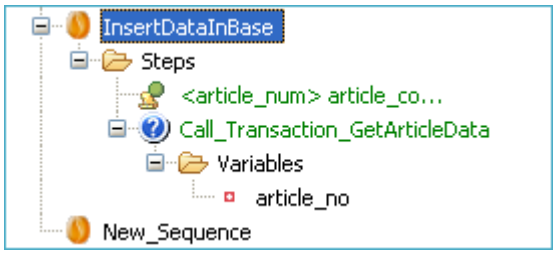


Figure 2 - 134: Start of second sequence

WHAT COMES NEXT?

The `GetArticleData` transaction call is properly defined through the `Call_Transaction_GetArticleData` step. Following steps can therefore be created, and their sources set as pointing towards the required nodes of the transaction **XML Output Schema**.

The following step consists in checking if, after the `GetArticleData` transaction has been called, the generated XML contains *at least* one of the following XML elements: `article_status`, `article_rsp` or `product_group`.

To this end, an *IfExist* step called `IfArticleDataExist` will be created and set.

The following procedure is similar to the first sequence procedure for creating and setting the `IfArticlesTableExist` step (see "To create and set an IfExist step" on page 2-47). Only the step name and source change. For documentation lightness purpose, the procedure is repeated here with a limited number of snapshots.

To create and set the `IfArticleDataExist` step

1 Right-click on the sequence (for example `InsertDataInBase`).

A contextual menu appears.

2 Select ***New > Step***.

The **New Step** wizard is automatically launched.

3 In the **Other steps** area, select **IfExist**.

4 Click on **Next**.

5 In the **Name** field that appears, type in the *IfExist* step name (for example `IfArticleDataExist`):

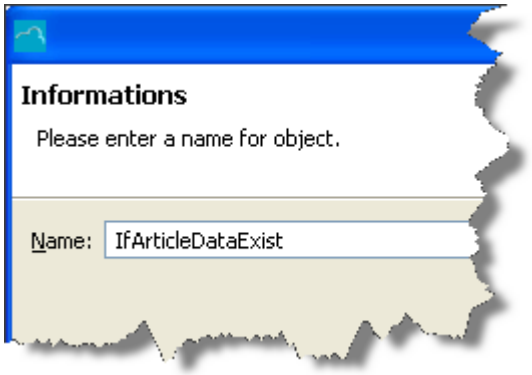


Figure 2 - 135: Entering the IfExist step name

6 Click on **Finish**.

The new step appears at the end of the sequence **Steps** folder in the **Projects View**:

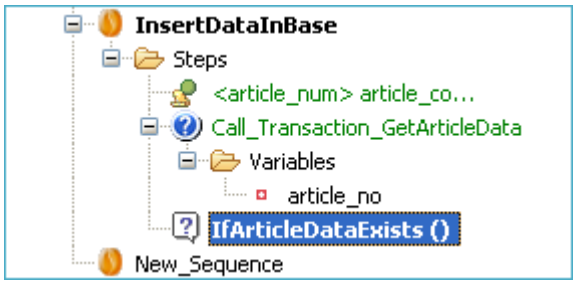


Figure 2 - 136: New IfExist step in Steps folder

Now that the *IfExist* step has been created, its source must be set. The step source needs to point towards XML nodes (`article_status`, `article_rsp` and `product_group`) to be checked within the XML output returned by the `GetArticleData` transaction (called by the preceding step).

7 Save your project by clicking on  or by pressing `Ctrl + S`.

8 Select the *IfExist* step with a left-click.

The **Properties View** is automatically updated.

9 In the **Properties View**, click in the **Value** column of the **Source** property.

A  button appears.

10 Click on the  button to launch the **Step Source** wizard.

The **Step Source** wizard is automatically launched.

11 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

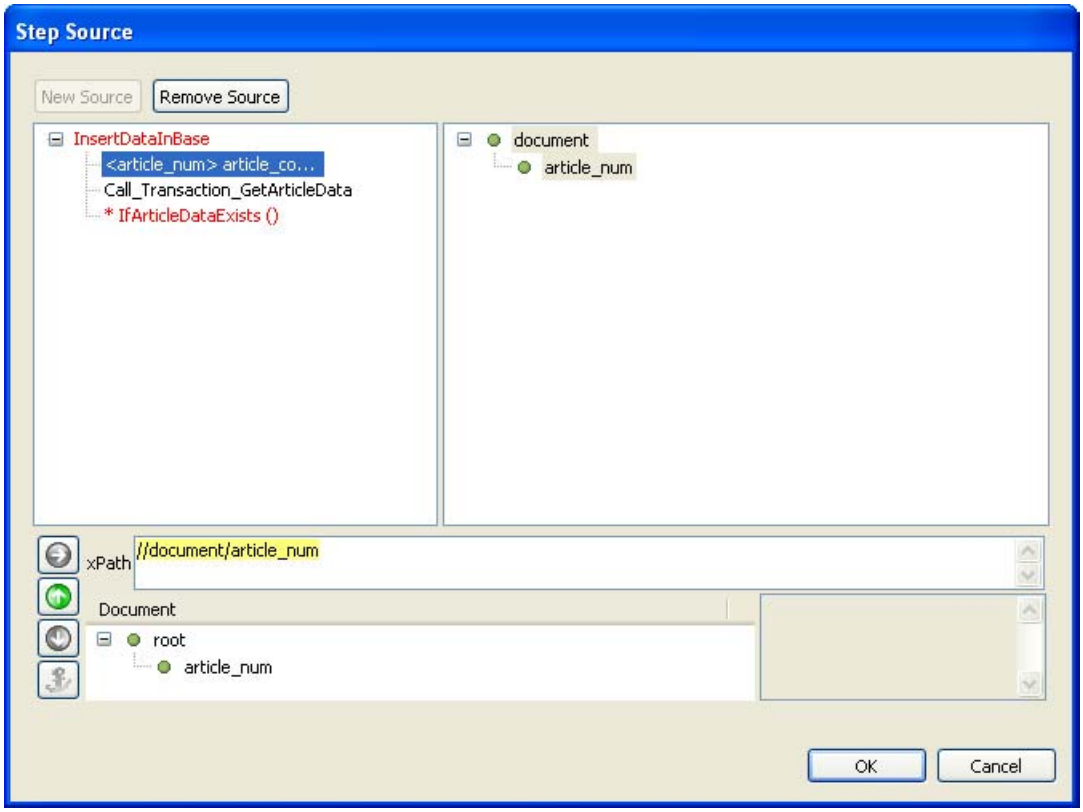


Figure 2 - 137: Setting of IfArticleDataExist step source

The previously defined steps appear in the **Steps Tree Structure**. At this point in the project, two steps can be selected: the *jElement* step and the *Call Transaction* step.

We need to check nodes in the XML generated by the `Call_Transaction_GetArticleData` step. This step must therefore be selected first to access its **XML Output Schema**.

12 In the **Steps Tree Structure**, select `Call_Transaction_GetArticleData` with a left-click.

The selected step's **XML Output Schema** is automatically displayed:

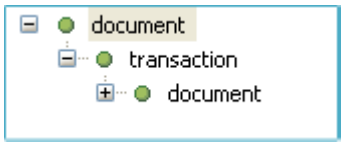


Figure 2 - 138: XML Schema of Call_transaction_GetArticleData step

13 Expand the `document`, then `blocks` node by clicking on ⊞ :

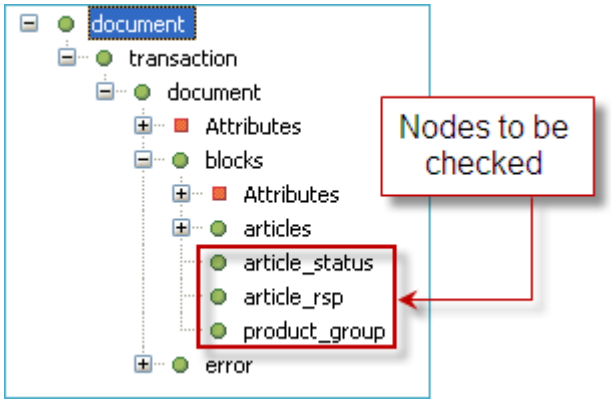


Figure 2 - 139: Nodes to check in the called transaction XML output

Selecting any node with a left-click automatically generates the XPath to the selected node in the **xPath** field of the **XPath Evaluator**.

However, we want to perform a check on the existence of *at least* one of these nodes.

To this end, we will:

- generate the XPath for one node,
- edit the XPath expression using an `or` operator to include the two other nodes to be checked.

For more information on XPath, see the XPath tutorial from W3Schools at http://www.w3schools.com/XPath

14 In the **XML Output Schema**, select the `article_status` node.

The XPath expression to this node is automatically generated in the **XPath Evaluator**:

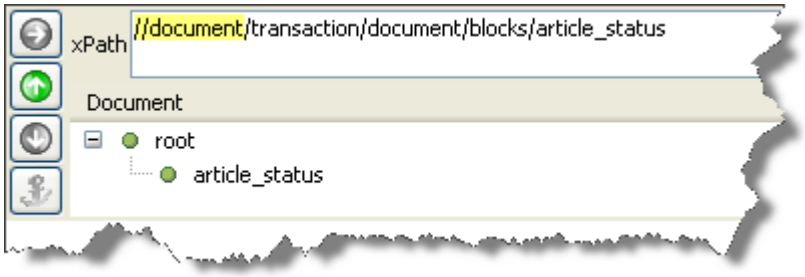


Figure 2 - 140: Generation of XPath to article_status node

15 Click in the **xPath** field of the **XPath Evaluator**.

16 Change the XPath expression to:

```
//document/transaction/document/blocks[article_status or
article_rsp or product_group]
```

This Xpath addresses a `blocks` element that contains at least a child node with tag name `article_status` or `rsp_article` or `product_group`. It matches if one of the above mentioned nodes exists in the *Call Transaction* step output DOM.

17 Press `Enter` to check that the execution of the XPath expression on the XML Schema

returns expected nodes.

The **Document** tree structure displays the result of the execution:

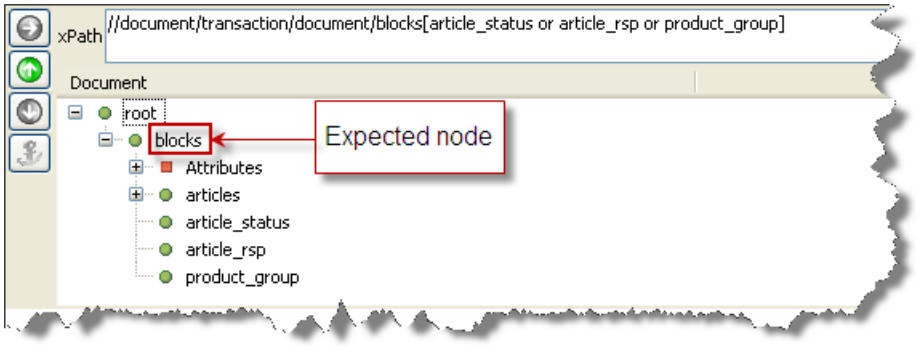


Figure 2 - 141: Display of XPath execution on XML Schema in Document tree structure

18 Click on **OK**.

The step source property is automatically updated.

19 Save your project by clicking on  or by pressing `Ctrl + S`.

WHAT COMES NEXT?

The step for checking the presence of required nodes in the `GetArticleData` transaction XML output is now set.

According to the project description (see *"Second sequence description"* on page 2-5), the next milestone in the project is now to insert into the `articles` table of the database:

- the article code (= value of the `article_code` variable),
- the article data retrieved from the transaction: status, rsp and product group (= content of generated XML `article_status`, `article_rsp` and `product_group` tags).

In CMS, inserting data into the `articles` table requires:

1. *Creating* an `InsertArticle` SQL transaction in SQL connector.
2. *Creating* a `Call_Transaction_InsertArticle` step at sequence level calling the `InsertArticle` SQL transaction with all variables needed imported from the `InsertArticle` target transaction into the step.
3. *Setting* the `Call_Transaction_InsertArticle` variables so that the source of each variable points towards proper nodes of the `GetArticleData` transaction XML output.

We will now create the `InsertArticle` SQL transaction, set its variables and **Query** property, extract its WSDL types and expose it as a *public method*.

To create and set an SQL transaction

1 Right-click on the connector (for example `DatabaseConnector`).

A contextual menu appears:

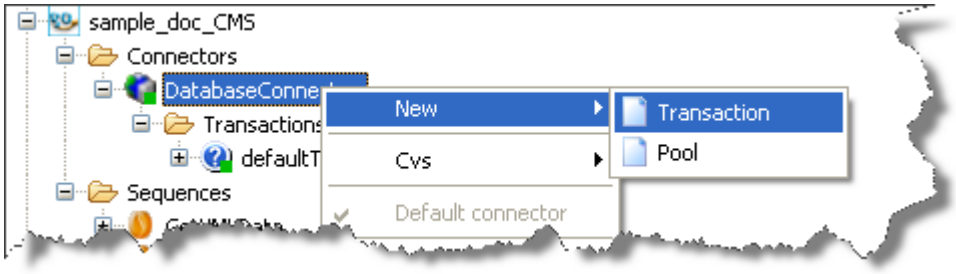


Figure 2 - 142: Creating a new transaction

2 Select ***New > Transaction***.

A **New Transaction** wizard is automatically launched.

3 Select **SQL Transaction**:

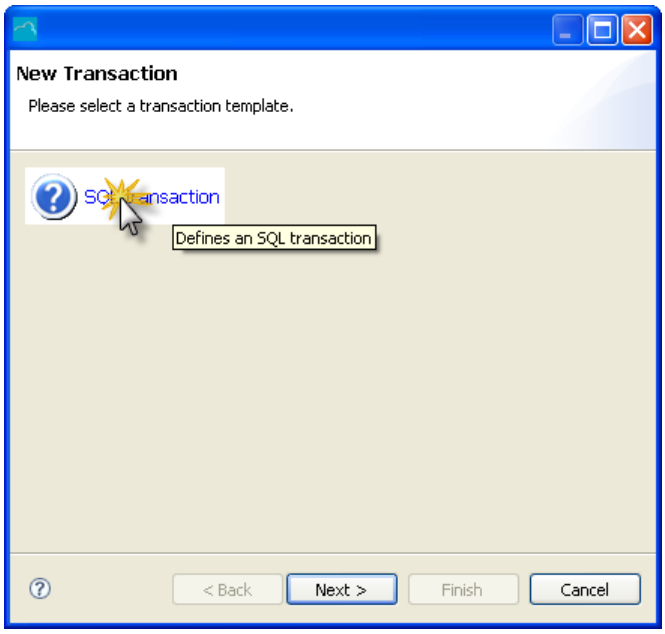


Figure 2 - 143: New Transaction wizard

4 Click on **Next**.

5 In the **Name** field that appears, type in the name of the transaction (for example `InsertArticle`):

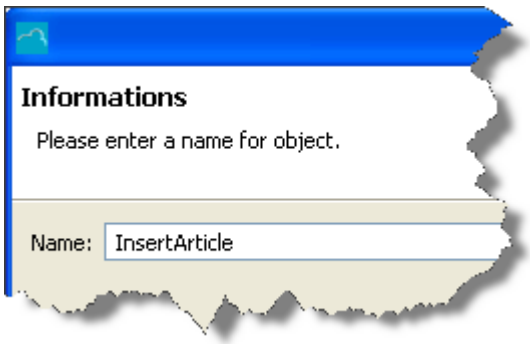


Figure 2 - 144: Giving a name to the new SQL transaction

6 Click on **Finish**.

The SQL transaction is now created. It appears unsaved in the **Transactions** folder of the project, under the connector:

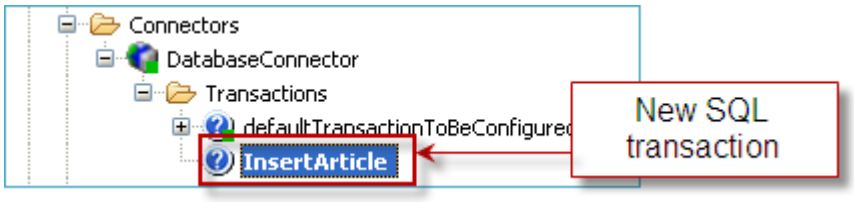


Figure 2 - 145: New SQL transaction in Transactions folder

We will now set the transaction main property: the **Query** property, which contains the transaction SQL query.

7 Select the SQL transaction with a left-click.

The **Properties View** is automatically updated:

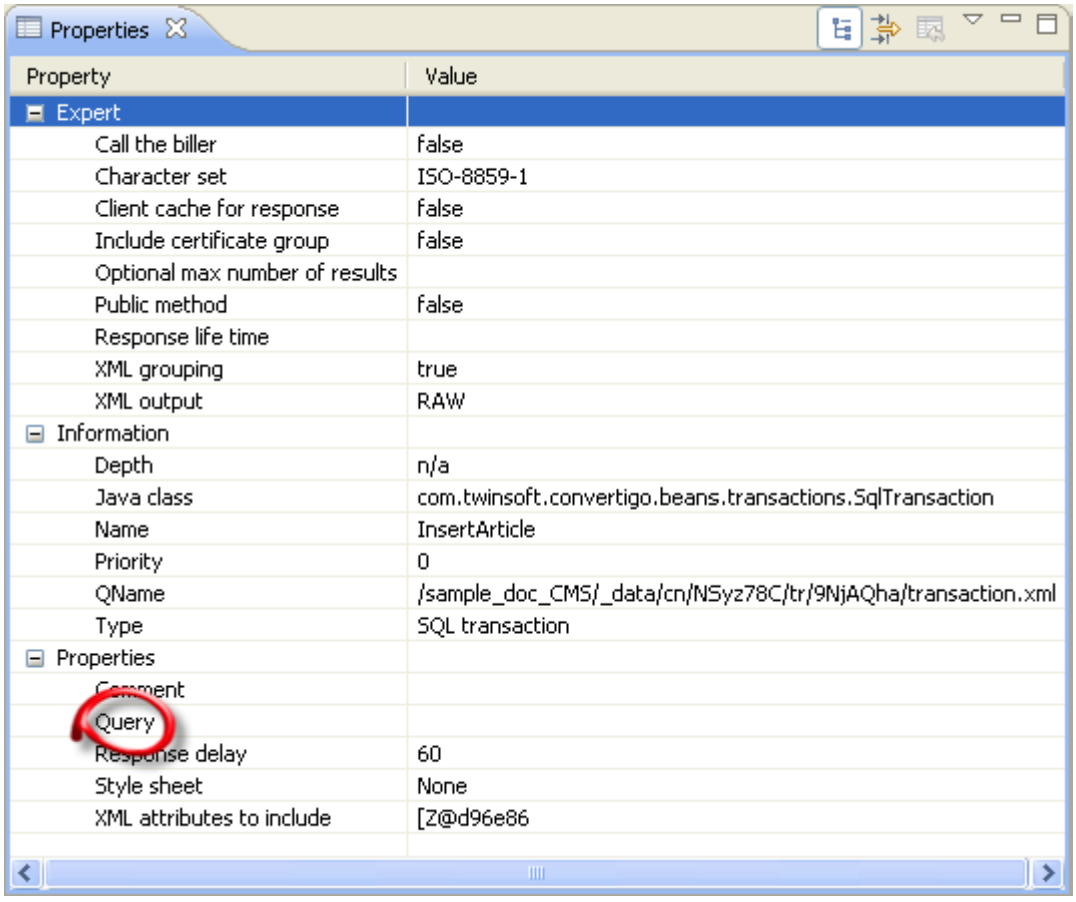


Figure 2 - 146: SQL transaction properties

8 Click in the **Value** column of the **Query** property.

A button appears.

9 Click on the button.

An **SQL Query** window opens:

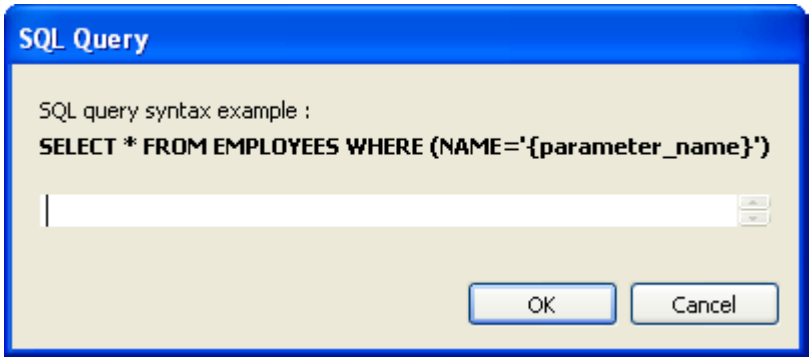


Figure 2 - 147: SQL query window

The information that we have for writing the insertion query is the following:

- data must be inserted into the `article_code`, `article_status`, `article_rsp`, and `article_product_group` columns of the `articles` table (see Table 2 - 1 on page 2-2),
- the value of these data being variable, they will be inserted in the form of variables (called respectively `a_code`, `a_status`, `a_rsp` and `a_product_group`),
- the SQL syntax for inserting data into a table using variable values is: `INSERT INTO tableName (tableColumn1, tableColumn2, ..., tableColumnX) VALUES ('{variableName1}', '{variableName2}',..., '{variableNameX}';`

The syntax for defining variable values to be inserted in queries (`{variableName}`) is specific to Convertigo.

Given this information, the value of the **Query** property must be set as:

```
INSERT INTO articles (article_code, article_status,
article_product_group, article_rsp, article_name)
VALUES('{a_code}','{a_status}', '{a_product_group}',
'{a_rsp}', '{a_name}');
```

Figure 2 - 148: SQL query for inserting required data into articles table

The `a_name` variable is not used at this point of the project but will be needed later.

10 Enter in the **SQL Query** window the required SQL query (see Figure 2 - 148).

11 Click on **OK**.

The transaction variables are automatically generated in the **Variables** folder of the transaction with values set between braces in the query:

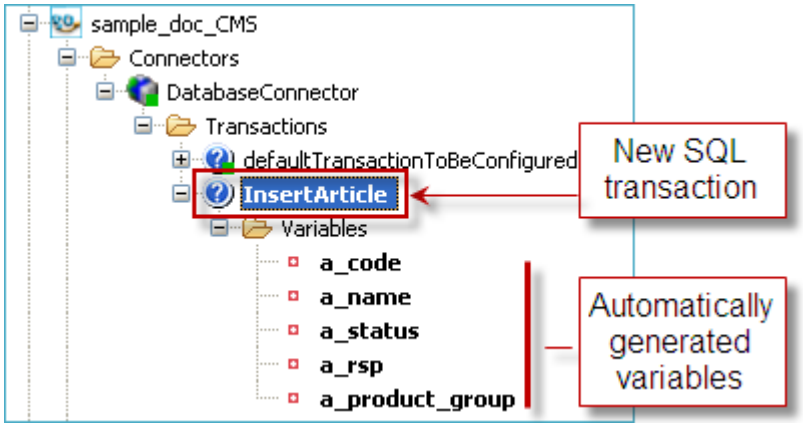


Figure 2 - 149: New SQL Transaction and generated variables

12 Save your project by clicking on or by pressing `Ctrl + S`.

The SQL transaction is now created and properly set.

WHAT COMES NEXT?

However, to be fully usable, an SQL transaction must be:

- *Tested* to check that there is no syntax error.
- *Set* as a public method.

After the SQL transaction has been tested and set as a public method, its XML schema must be updated to match the XML generated when testing the transaction.

Only then can a *Call Transaction* step be created and set for calling it.

TESTING AN SQL TRANSACTION AND UPDATING ITS XML SCHEMA

This sub-section describes how to:

1 *Test* an SQL transaction.

2 *Set* it as a public method.

3 *Update* its XML schema.

Prior to testing an SQL transaction, the SQL transaction variables must be set as fixed variables with default values.

This is done in the first steps of the following procedure.

To test an SQL transaction and extract its WSDL types (XML schema)

1 In the **Projects View**, select the first SQL transaction variable (for example `a_code`) with a left-click.

The **Properties View** is automatically updated:

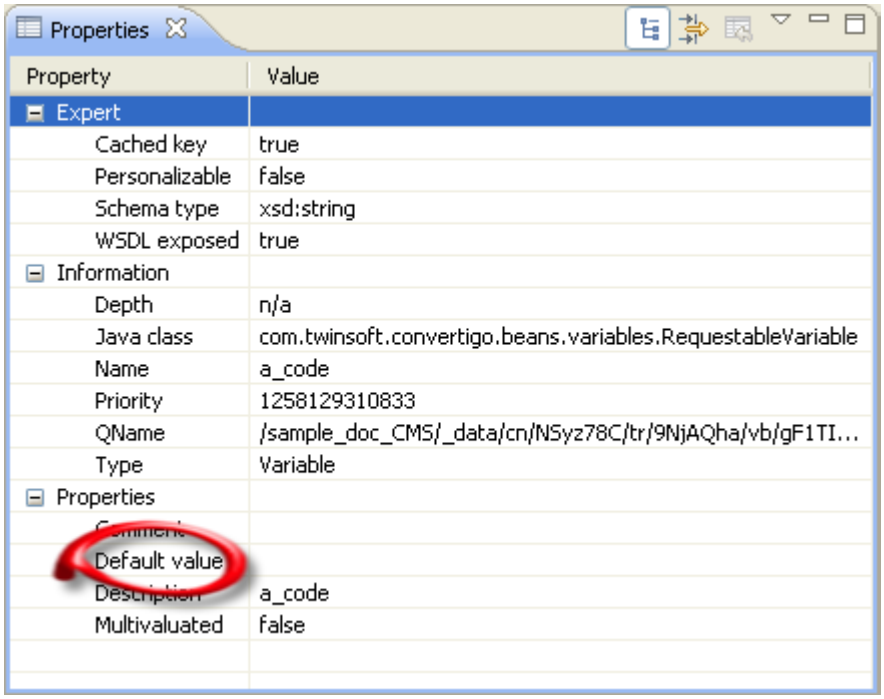


Figure 2 - 150: InsertArticle SQL variable properties

2 In the **Properties View**, click in the **Value** column of the **Default Value** property.

3 Type in a default value (for example `123456789`).

4 Press `Enter`.

5 Repeat points 1 to 4 of this procedure to assign default values to each variable of the SQL transaction (see following table for an example):

Table 2 - 11: SQL transaction variables default values

Variable	Default Value
`a_status`	`20`
`a_product_group`	`Groupe blanc`
`a_rsp`	`TTWISOF`
`a_name`	*leave blank*

6 Save your project by clicking on  or by pressing `Ctrl + S`.

SQL transaction variables are now set as fixed variables with default values.

We will now:

- *Open* the sequencer projet **Connector View** - this is where SQL transactions are displayed, along with the XML returned by the SQL transactions.
- *Execute* the SQL transaction - this will allow us to extract the SQL transaction XML schema.
- *Set* the SQL transaction as a *public method.*
- *Update* the SQL transaction XML schema.

7 In the **Projects View**, double-click on the project connector (for example `DatabaseConnector`).

The **Connector View** tab opens (see *"SQL Connector View"* on page 1-18) or, if already opened, appears in front of other tabs.

8 In the **Projects View**, right-click on the SQL transaction (for example `InsertArticle`).

A contextual menu appears:

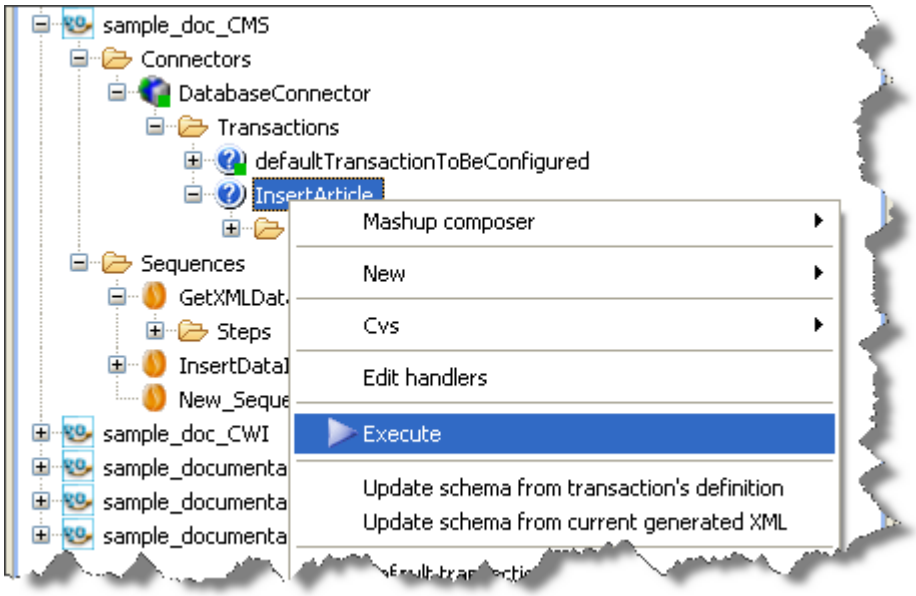


Figure 2 - 151: Executing an SQL transation

9 Select **Execute**.

You can also execute the transaction using the test platform. For more information, see "To execute a sequence using the test platform" on page 2-145.

The transaction starts.

The XML generated by the transaction appears in the **XML** output of the SQL **Connector View**:

Figure 2 - 152: XML generated by SQL transaction

The SQL transaction has returned an `<sql_output>` tag including the following content: `1 row(s) updated`.

This indicates that a row of the table included in the SQL query of the transaction (here, the `articles` table) has been updated.

To check that default data has been inserted in the table, query the table using a query browser application or with a Convertigo transaction.

Knowing that a row has been inserted in the table is a useful information when executing a sequence.

We can now reset all default values.

For each variable,

10 Click in the **Value** column of the **Default Value** property.

11 Delete the default value.

12 Press `Enter`.

For the generated information to be usable by the *Call Transaction* step calling the transaction, the XML schema of the generated XML output must be known.

To this end, we will set the SQL transaction as a *public method* then update the SQL transaction XML schema from the SQL transaction.

13 In the **Projects View**, select the SQL transaction with a left-click (for example `InsertArticle`).

The **Properties View** is automatically updated:

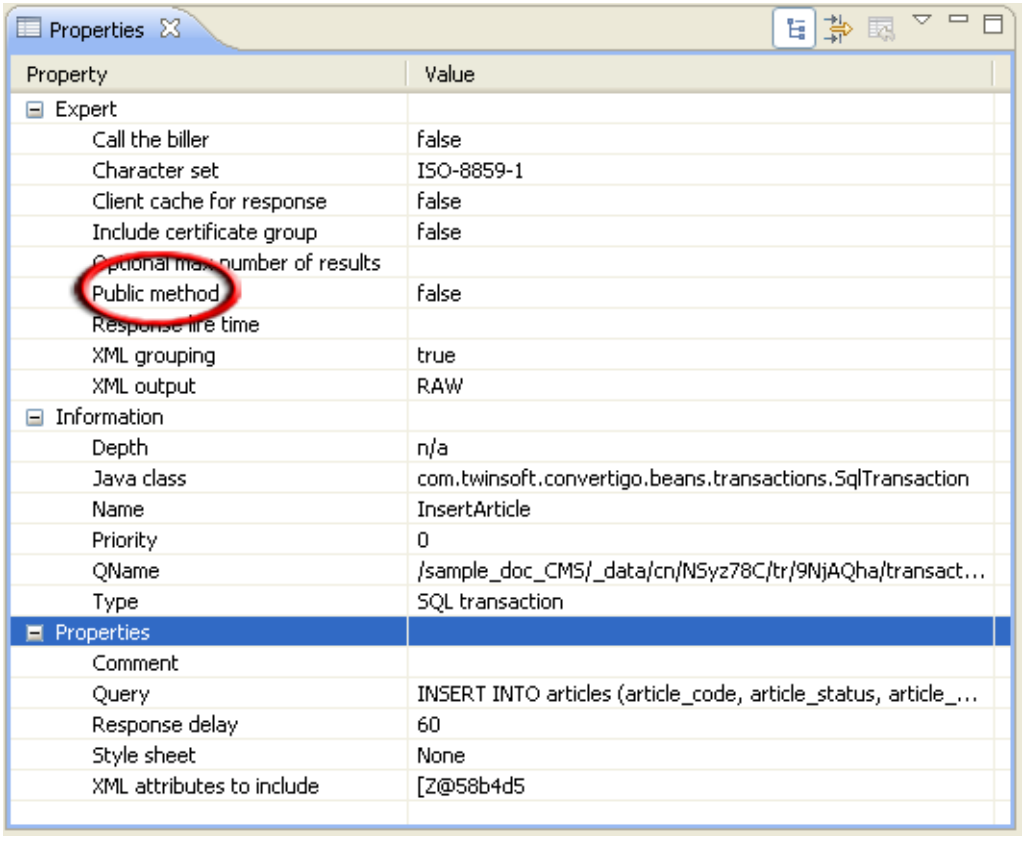


Figure 2 - 153: SQL transaction properties

14 Click in the **Value** column of the **Public Method** property.

The current value is highlighted and a symbol appears.

15 Click on .

A drop-down menu displays available **Output** values.

16 Select `true`.

The property is updated and the step appears bolded in the **Projects View**.

17 Press `Enter`.

18 Save your project by clicking on or by pressing `Ctrl + S`.

Ths SQL transaction is now set as a public method. We will now update its XML schema.

19 In the **Projects View**, right-click on the SQL transaction.

A contextual menu appears.

20 Select **Update schema from current generated XML**.

A confirmation window opens:

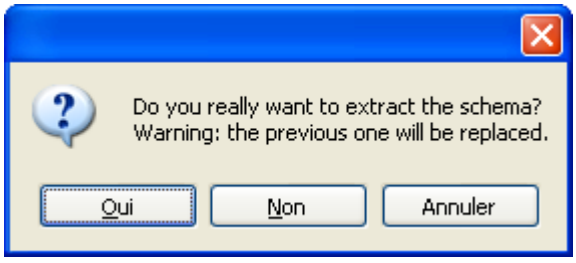


Figure 2 - 154: WSDL types extraction confirmation window

21 Click on **Yes**.

The SQL transaction XML schema is updated and the transaction appears bolded in the **Projects View**.

22 Save your project by clicking on or by pressing `Ctrl + S`.

The SQL transaction is tested, set as a public method and its XML schema is updated - the transaction is ready to be called by a *Call Transaction* step.

What comes next?

We will now:

1. *Create* a `Call_Transaction_InsertArticle` step calling the `InsertArticle` SQL transaction.
2. *Import* the `InsertArticle` target transaction variables into the `Call_Transaction_InsertArticle` step.
3. *Set* the `Call_Transaction_InsertArticle` variables so that the source of each variable points towards proper nodes of the `GetArticleData` transaction XML output.

The following procedure is similar to the first sequence procedure for calling the `GetArticleData` transaction (see "To create and set a Call transaction step" on page 2-18). For documentation lightness purpose, the procedure is repeated here with a limited number of snapshots.

The transaction call must be performed ***only if the article data exists***. This means that the `Call_Transaction_InsertArticle` step must be defined as a ***child*** of the `IfArticleDataExist` step.

To create and set the `Call_Transaction_InsertArticle` step

1. In the **Projects View**, right-click on the parent step (for example `IfArticleDataExist`).

   A contextual menu appears.

2. Select ***New > Step***.

   A **New Step** wizard is automatically launched.

3. In the **Other Steps** area of the window, select **Call Transaction** with a left-click.

4. Click on **Next**.

5. In the **Name** field that appears, type in the step name (for example `Call_Transaction_InsertArticle`).

6. Click on **Finish**.

   The new step appears in the **Steps** folder of the sequence:

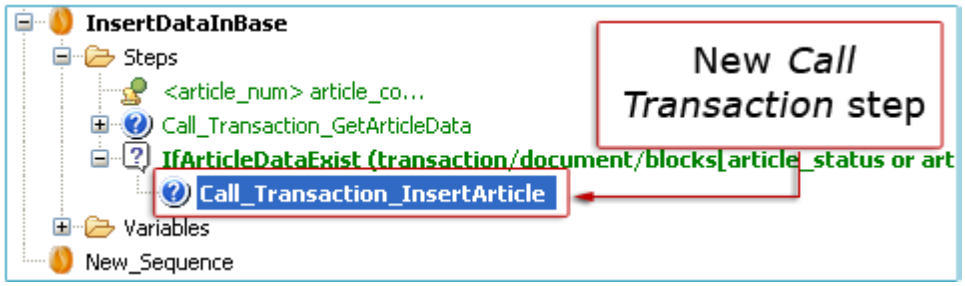


Figure 2 - 155: New Call Transaction step in Steps folder

7. Save your project by clicking on  or by pressing `Ctrl + S`.

   Now that the *Call Transaction* step has been created, we will set its three main interrelated parameters:

   - **Project** - in this case, the current project (`sample_doc_CMS`),
   - **Connector -** in this case, the `sample_doc_CMS` project's sole connector (`DatabaseConnector`),
   - **Transaction -** in this case, the `DatabaseConnector` connector's sole transaction (`InsertArticle`).

8. In the **Projects View**, select the step (for example `Call_Transaction_InsertArticle`).

   The **Properties View** is automatically updated. The default value of the **Project**

property (`sample_doc_CMS`) and **Connector** property (`DatabaseConnector`) can be left as are.

9 In the **Properties View**, click in the **Value** column of the **Transaction** property.

The current value is highlighted and a drop-down symbol appears.

10 Click on .

11 Select the transaction (for example `InsertArticle`):

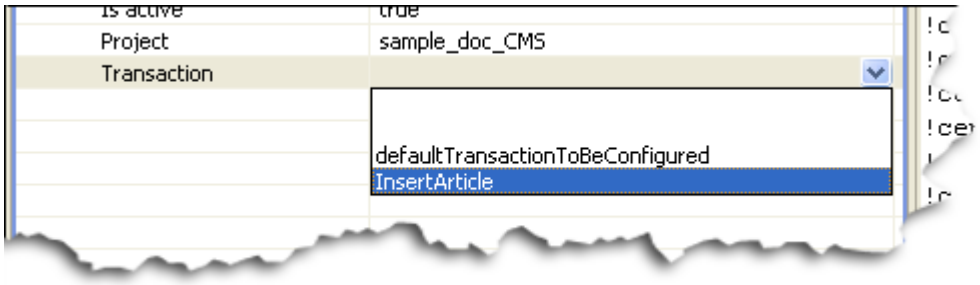


Figure 2 - 156: Setting the Transaction property of the Call Transaction step

12 Press `Enter`.

13 Save your project by clicking on or by pressing `Ctrl + S`.

Since the `InsertArticle` transaction expects a number of input variables, we will now import its variables .

The procedure for importing variables has already been described. For more information, see "To import variables from a target transaction at step level" on page 2-22.

14 Right-click on the step (for example `Call_Transaction_InsertArticle`).

A contextual menu appears:

15 Select **Import variables from target transaction**.

Once variables have been imported from the target transaction:

- the step appears as "changed and unsaved" (green bolded characters, see Figure 1 - 9),
- variables appear in a **Variables** sub-folder (together with the default value imported from the target transaction between brackets, when applicable):

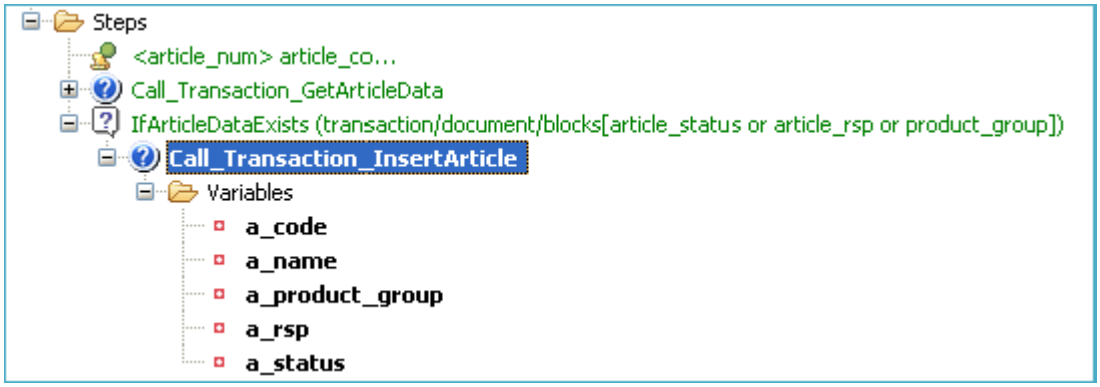


Figure 2 - 157: Variables imported from target transaction (and default values)

The *Call Transaction* step is now created and its variables are imported. These variables now need to be set.

WHAT COMES NEXT?

Imported variables now need to be set so that their source points towards the relevant nodes of the XML output generated by:

- the *jElement* first step;
- the `Call_Transaction_GetArticleData` second step.

The table below describes the nodes towards which sources of the `Call_Transaction_InsertArticle` variables must point:

Table 2 - 12: `Call_Transaction_InsertArticle` step - Imported variable sources

The source of variable...	...must point to XML node...	...generated by...
`a_code`	`article_num`	*jElement* step (first step)
`a_status`	`article_status`	call to the `GetArticleData` transaction (second step)
`a_rsp`	`article_rsp`	
`a_product_group`	`product_group`	

These sources will now be set using the **Step Source** wizard.

To set the variable sources of a Call Transaction step using the Step Source wizard

1. Select the *Call Transaction* step variable (for example `a_code`) with a left-click.

   The **Properties View** is automatically updated.

2. In the **Properties View**, click in the **Value** column of the **Source** property.

   A button appears.

3. Click on the button.

   The **Step Source** wizard is automatically launched.

4. Click on **New Source**.

   The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

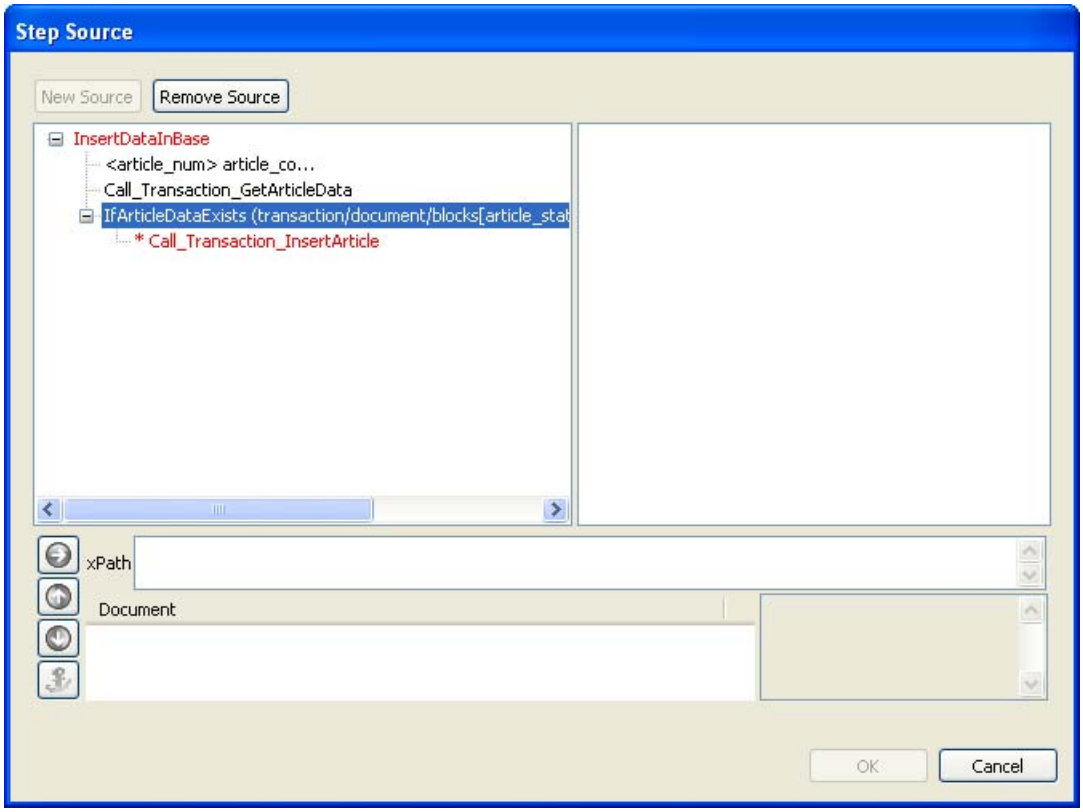


Figure 2 - 158: Setting of a_code variable source

The **Steps Tree Structure** displays all steps defined so far.

The value of the `a_code` must be sourced from the `article_num` node generated by the first *jElement* step. We will therefore select the *jElement* step in the **Steps Tree Structure**, then select the `article_num` node in the **XML Output Schema**.

5 In the **Steps Tree Structure**, select the `<article_num> article_code` step to display its XML schema.

The selected step's **XML Output Schema** is automatically displayed:

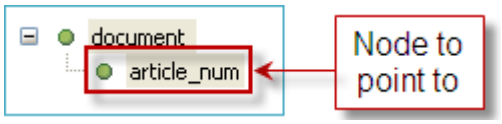


Figure 2 - 159: XML Output Schema of jElement step

This **XML Output Schema** contains only one node called after the value of the **Node name** property set for the step (see Table *"jElement main properties"* on page 2-25): `article_num`.

6 In the **XML Output Schema**, select the `article_num` node:

The XPath to this node is automatically generated in the **xPath** field of the **XPath Evaluator** and the result of this XPath on the XML schema is displayed in the **Document** tree structure:

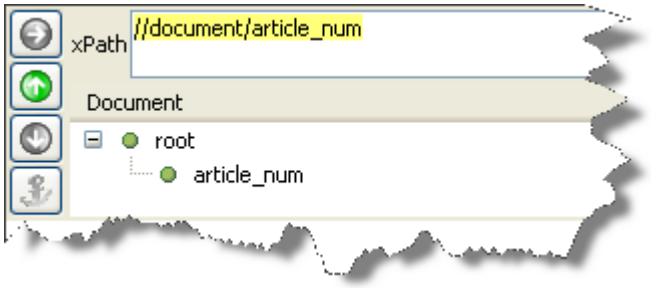


Figure 2 - 160: Generation of XPath to article_num node

7 Click on **OK**.

The variable source property is automatically updated.

8 Save your project by clicking on or by pressing `Ctrl + S`.

The `a_code` variable source is now properly set.

We will now set the second step variable, `a_product_group`.

The `a_name` variable is not used at this point of the project but will be needed later in the project. In this step, an empty value will be sent to the transaction when executed.

9 Select the *Call Transaction* step variable (`a_product_group`) with a left-click.

The **Properties View** is automatically updated.

10 In the **Properties View**, click in the **Value** column of the **Source** property.

A button appears.

11 Click on the button.

The **Step Source** wizard is automatically launched.

12 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16)

13 Click on **New Source**.

The source of the `a_product_group` variable must point to the `product_group` XML node generated by the `Call_Transaction_GetArticleData` step.

We will therefore select the `Call_Transaction_GetArticleData` step in the **Step Tree Structure**, then select the `product_group` in the **XML Output Schema**.

14 In the **Steps Tree Structure**, select the `Call_Transaction_GetArticleData` step with a left-click.

The selected step's **XML Output Schema** is automatically displayed:

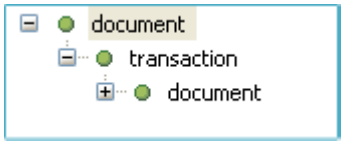


Figure 2 - 161: XML Schema of Call_transaction_GetArticleData step

15 Expand the `document`, then `blocks` node by clicking on ⊞:

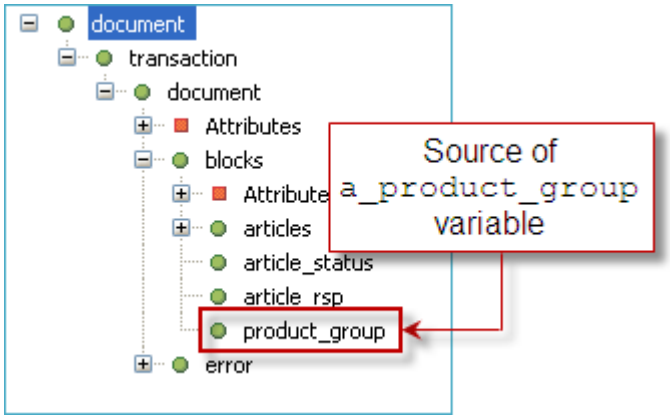


Figure 2 - 162: Source node for the statut variable

16 Select the `product_group` node with a left-click.

The XPath to this node is automatically generated in the **XPath Evaluator**:

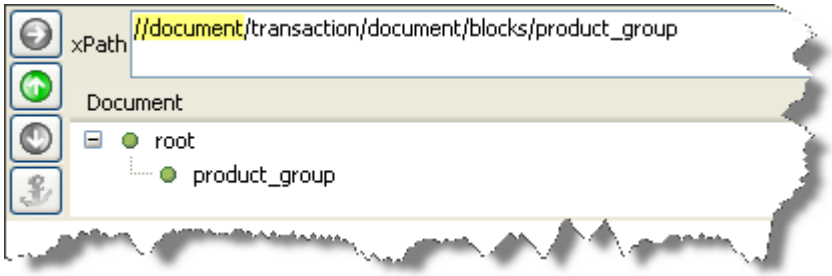


Figure 2 - 163: XPath to product_group node

17 Click on **OK**.

The `product_group` variable source is now properly set.

18 Repeat points 9 to 17 to set the remaining two variable sources (`a_rsp` and `a_status`, see Table *"`Call_Transaction_InsertArticle` step - Imported variable sources"* on page 2-96).

19 Save your project by clicking on  or by pressing `Ctrl + S`.

The sources of all needed variables are now properly set.

WHAT COMES NEXT?

Points 1 to 3 of the second sequence description (see *"Second sequence description"* on page 2-5) are now implemented in terms of Convertigo *steps*.

According to the sequence description, we now need to:

1 *Check* in the XML generated by the `GetArticleData` transaction the presence of the

`articles` XML table. This check is performed using an *IfExist* step (*see "To create and set the `IfArticleDataExist` step"* on page 2-81).

2 If so, *iterate* on each row of the `articles` table using an *Iterator* step and, for each row:

- insert the article code, name and status into the `articles` table using the previously set `InsertArticle` SQL transaction (see *"To create and set an SQL transaction"* on page 2-85) called by a *Call Transaction* step (see *"To create and set a Call transaction step"* on page 2-18),
- call the `searchGoogleWithLimit` CWI transaction using a *Call Transaction* step and iterate on each result usind an *Iterator* step and, for each result:
- insert returned Web pages and URLs into the `web_sites` table by creating an `InsertWebSite` SQL transaction called by a *Call Transaction* step.

The table below sums up the different CMS objects (steps and SQL transactions) to be created and set in this second part of the second sequence:

Table 2 - 13: CMS objects needed in the second part of the second sequence

CMS object type	Object name	Description
IfExist step	`IfArticlesTableExists`	Checks in the XML output generated by `GetArticleData` CLI transaction the presence of `articles` nodes.
	`IfListResultsTableExists`	Checks in the XML output generated by `searchGoogleWithLimit` transaction the presence of `listResults` nodes.
Call transaction step	`Call_Transaction_InsertArticle`	Calls CMS project-specific `InsertArticle` SQL transaction.
	`Call_Transaction_searchGoogle`	Calls CWI project-specific `searchGoogleWithLimit` transaction
	`Call_Transaction_InsertWebSite`	Calls CMS project-specific `InsertWebSite` SQL transaction
Iterator step	`IteratorOnEachRow`	Iterates on each `row` node of the `articles` XML table generated by the `GetArticleData` transaction.
	`IteratorOnEachResultItem`	Iterates on each `resultItem` node of the `listresults` XML table generated by the `searchGoogleWithLimit` CWI transaction.
SQL transaction	`InsertWebSite`	For each article found in the `articles` XML table, insert article code, URL and page title returned by `searchGoogleWithLimit` transaction in `web_sites` database table.
	`InsertArticle` (already created and set)	For each article found in the `articles` XML table, insert article code, article name and article status in `articles` database table.

The following procedures show how to create and set these ojects as they are needed in the sequence.

The next step needed in the sequence is the `IfArticlesTableExists` step. The purpose of this step is to check the presence of `articles` nodes in the XML output generated by the `GetArticleData` transaction.

To do so, we will create an *IfExist* step called `IfArticlesTableExists`, the source of which will point towards the XML output generated by the `Call_Transaction_GetArticleData` step.

The following procedure is similar to the first sequence procedure for creating and setting the `IfArticlesTableExists` step (see "To create and set an IfExist step" on page 2-47). Only the step name and source change. For documentation lightness purpose, the procedure is repeated here with a limited number of snapshots.

To create and set the `IfArticlesTableExists` step

1. Right-click on the sequence.

   A contextual menu appears.

2. Select ***New > Step***.

   The **New Step** wizard is automatically launched.

3. In the **Other steps** area, select **IfExist**.

4. Click on **Next**.

5. In the **Name** field that appears, type in the *IfExist* step name (for example `IfArticlesTableExists`):

6. Click on **Finish**.

   The new step appears at the end of the Steps Tree Structure in the **Projects View**:

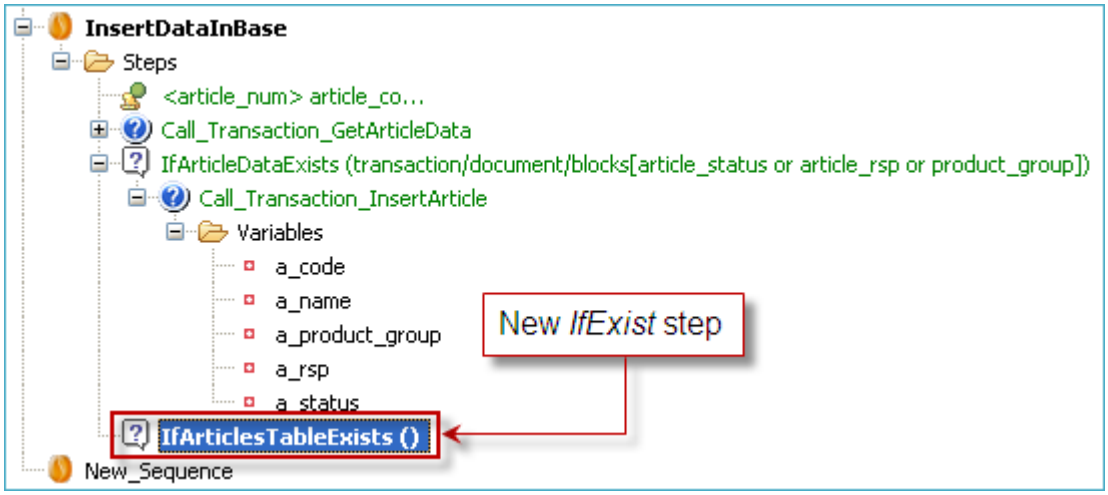


Figure 2 - 164: New IfExist step in Steps folder

Both *IfExist* steps stand alone. They are not correlated and can therefore appear at the same level in the sequence.

Now that the *ifExist* step has been created, its source must be set. The step source needs to point towards the XML node (`articles`) to be checked within the XML output returned by the `GetArticleData` transaction (called by the `Call_Transaction_GetArticleData` step).

7. Save your project by clicking on  or by pressing `Ctrl + S`.

8 Select the *IfExist* step with a left-click.

The **Properties View** is automatically updated.

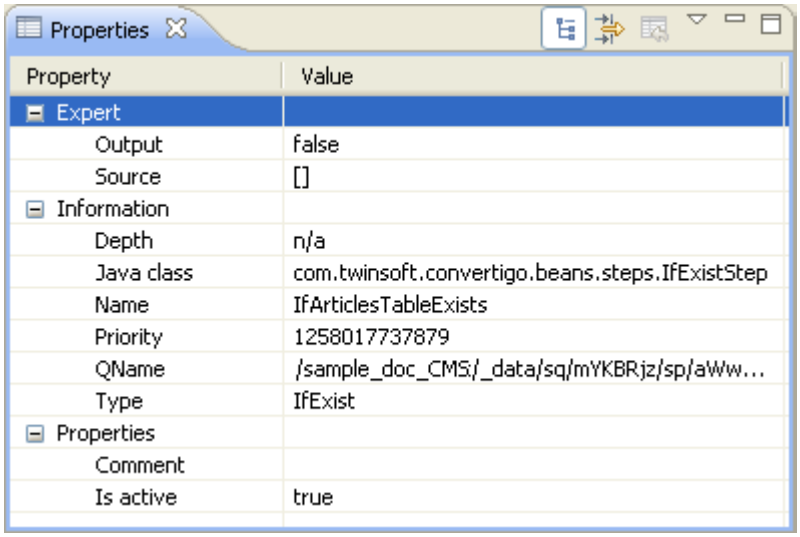


Figure 2 - 165: IfExist step properties

9 Click in the **Value** column of the **Source** property.

A button appears.

10 Click on the button to launch the **Step Source** wizard.

The **Step Source** wizard is automatically launched.

11 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

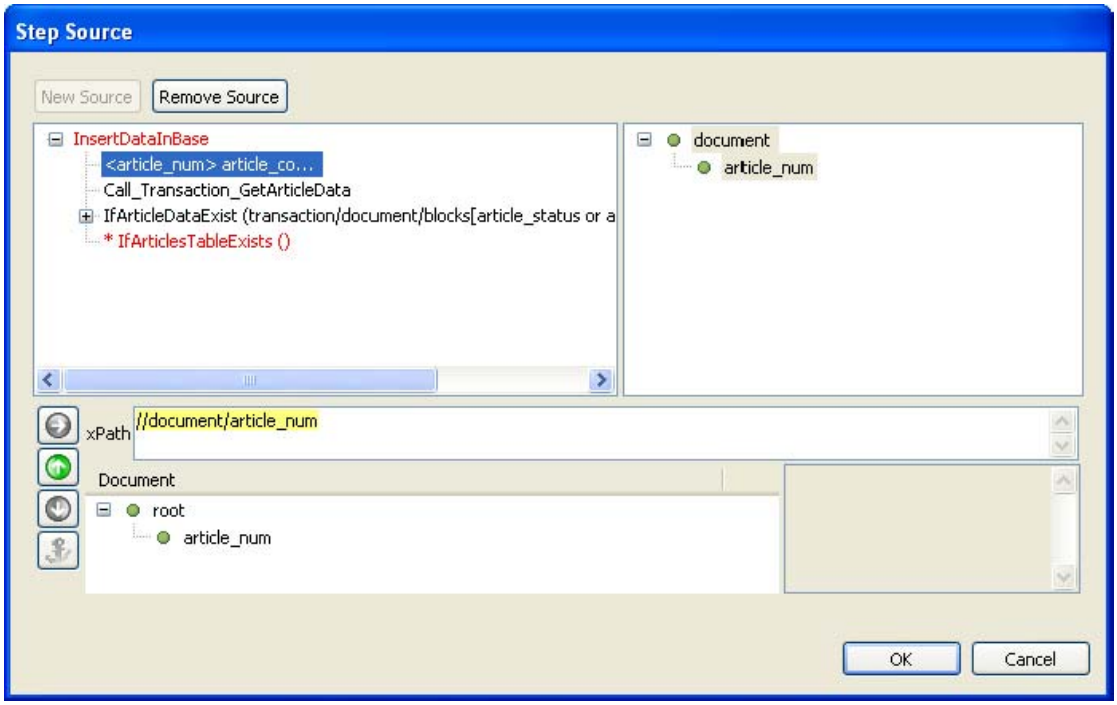


Figure 2 - 166: Setting of IfArticlesTableExists step source

The *IfExist* check must be performed on the XML schema of the output generated when calling the `GetArticleData` transaction. The step to be selected is therefore `Call_Transaction_GetArticleData`.

12 In the **Steps Tree Structure**, select the required step (for example `Call_Transaction_GetArticleData`) with a left-click.

The selected step's **XML Output Schema** is displayed:

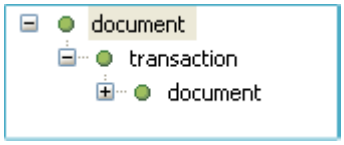


Figure 2 - 167: XML Schema of Call_transaction_GetArticleData step

13 Expand the `document`, then `blocks` node by clicking on ⊞:

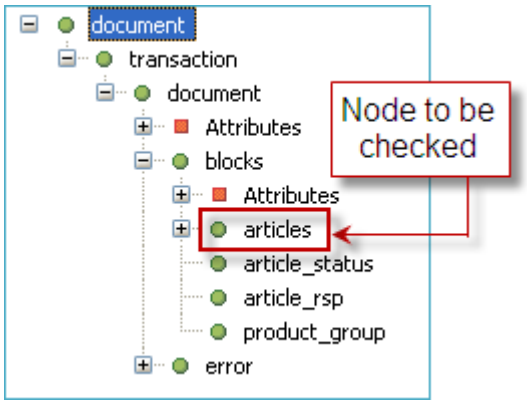


Figure 2 - 168: Node to check in the called transaction XML output

14 In the **XML Output Schema**, select the node to be checked (`articles`).

15 The XPath expression to this node is automatically generated in the **XPath Evaluator**:

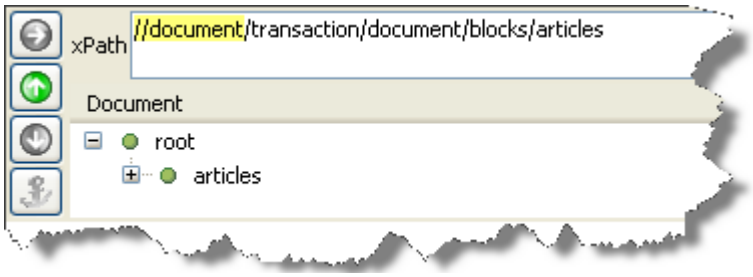


Figure 2 - 169: Generation of XPath to articles node

16 Click on **OK**.

The **Source** step property is automatically updated:

Figure 2 - 170: Source property automatically updated in Properties View

17 Save your project by clicking on  or by pressing `Ctrl + S`.

The *IfExist* step is now created and properly set.

WHAT COMES NEXT?

The following milestone in the project consists in iterating on each row of the `articles` XML table (for an example of `articles` XML table, refer to the *"Starting with Convertigo Legacy Integrator" Quick Guide*) to perform operations on these rows (insert row data into database table, call CWI transaction using row data, etc.).

To this end, we will now create and set an *Iterator* step.

The following procedure is similar to the first sequence procedure for creating and setting the `IteratorOnEachRow` step (see "To create and set an IfExist step" on page 2-47). Only the step name and source change. For documentation lightness purpose, the procedure is repeated here with a limited number of snapshots.

The iteration must be performed ***only if the articles table exists***. This means that the `IteratorOnEachRow` step must be defined as a ***child*** of the `IfArticlesTableExists` step.

To create and set an Iterator step

1. In the **Projects View**, right-click on the parent step (for example the `IfArticlesTableExists` step).

   A contextual menu appears.

2. Select ***New > Step***.

   A **New Step** wizard is automatically launched:

3. In the **Other Steps** area of the window, select **Iterator** with a left-click.

4. Click on **Next**.

5. In the **Name** field that appears, type in the step name (for example `IteratorOnEachRow`).

6. Click on **Finish**.

   The new step appears in the **Steps** folder of the sequence:

Figure 2 - 171: New Iterator step in Steps folder

7 Save your project by clicking on or by pressing `Ctrl + S`.

Now that the *Iterator* step has been created, we will set its **Source** property so that it points towards the required node (`row`) of the `Call_Transaction_GetArticleData` step XML schema.

8 Select the *Iterator* step (for example `IteratorOnEachRow`) with a left-click.

The **Properties View** is automatically updated:

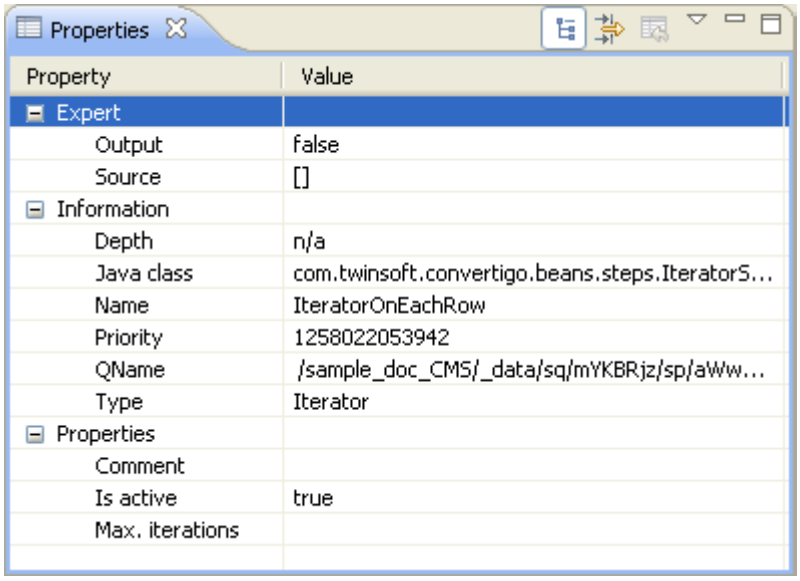


Figure 2 - 172: Iterator step properties

9 In the **Properties View**, click in the **Value** column of the **Source** property.

A button appears.

10 Click on the button.

The **Step Source** wizard is automatically launched.

11 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

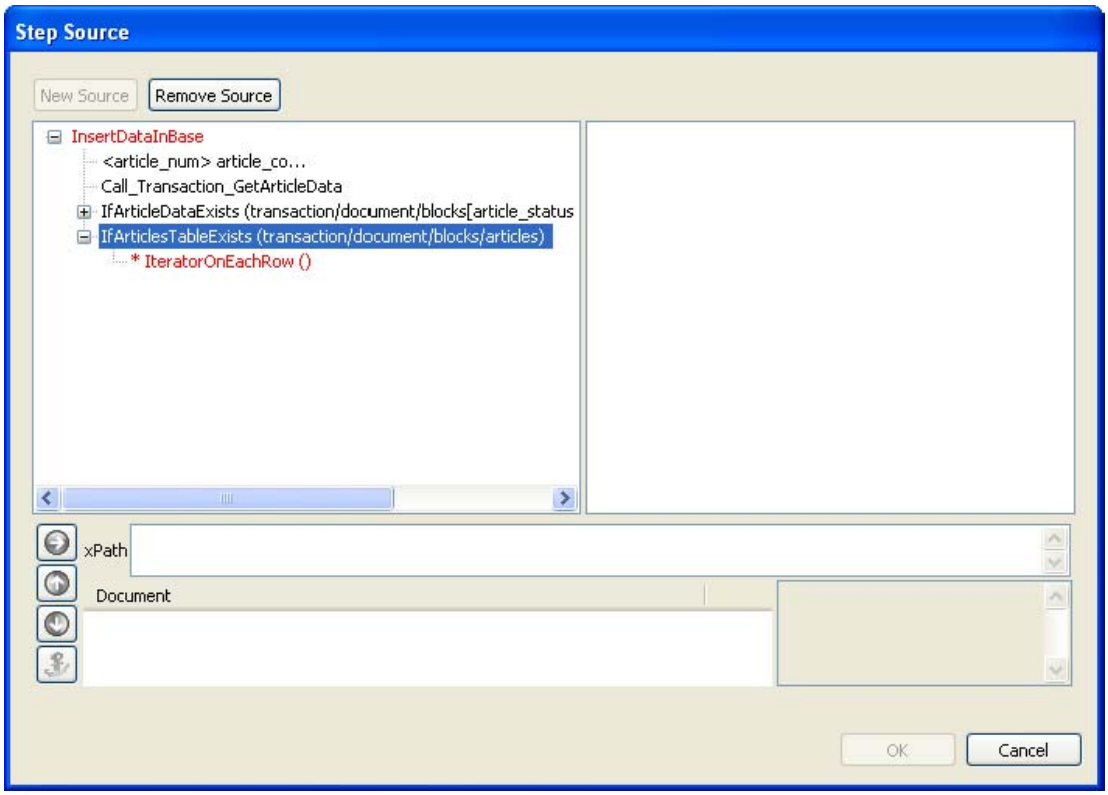


Figure 2 - 173: Generating Iterator step source

The iteration must be performed on each `row` node of the articles XML table included in the output generated when calling the `GetArticleData` transaction. The step to be selected is therefore `Call_Transaction_GetArticleData`.

12 In the **Steps Tree Structure**, select the required step (for example `Call_Transaction_GetArticleData`) with a left-click.

The selected step's **XML Output Schema** is displayed:

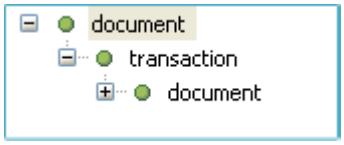


Figure 2 - 174: XML Schema of Call_transaction_GetArticleData step

13 Expand the `document`, `blocks` and `articles` node by clicking on ⊞:

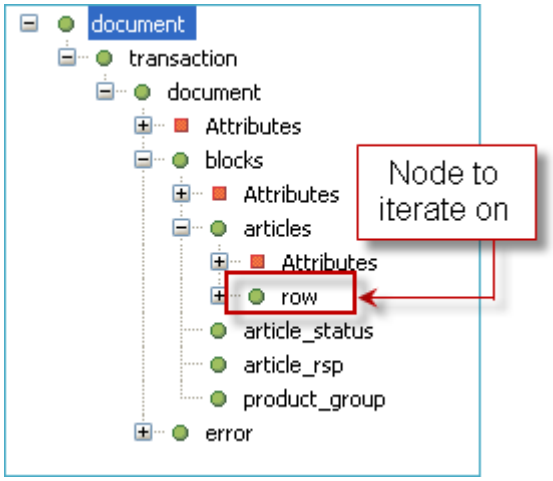


Figure 2 - 175: Node to iterate on in the called transaction XML output

14 In the **XML Output Schema**, select the `row` node.

*One `row` node only appears in the **XML Schema** because this is schema. When executing the `GetArticleData` transaction, several `row` nodes will be generated.*

15 The XPath expression to this node is automatically generated in the **XPath Evaluator**:

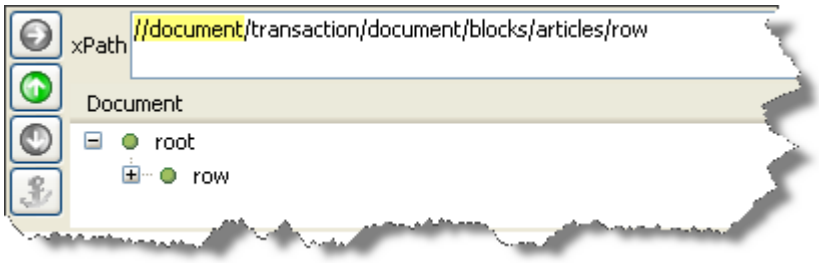


Figure 2 - 176: Generation of XPath to row node

16 Click on **OK**.

The **Source** step property is automatically updated:

Figure 2 - 177: Source property automatically updated in Properties View

17 Save your project by clicking on  or by pressing `Ctrl + S`.

The *Iterator* step is now created and properly set.

WHAT COMES NEXT?

Each `row` node is parent to three nodes: `code`, `name` and `status`. The `IteratorOnEachRow` step will therefore serve as source for child steps such as:

- `Call_Transaction_InsertArticle`, since the source of this step variables must point to the `code`, `name` and `status` nodes of ***the currently iterated row***, for the `InsertArticle` transaction to insert the content of these nodes into the `articles` table,

- `Call_Transaction_searchGoogle`, since the source of this step variables must point to the name node of ***the currently iterated row***, for the `searchGoogleWithLimit` transaction to use the content of this node as input variable (search keyword).

The following step consists in creating the `Call_Transaction_InsertArticle` step.

The following procedure is similar to the previously described procedure for creating and setting a first `Call_Transaction_InsertArticle` step (see "To create and set a Call transaction step" on page 2-18). Only the node name and source change. For documentation lightness purpose, the procedure is repeated here with a limited number of snapshots.

To create and set the second `Call_Transaction_InsertArticle` step

1. In the **Projects View**, right-click on the parent step (`IteratorOnEachRow`).

   A contextual menu appears.

2. Select ***New > Step***.

   A **New Step** wizard is automatically launched.

3. In the **Other Steps** area of the window, select **Call Transaction** with a left-click.

4. Click on **Next**.

5. In the **Name** field that appears, type in the step name (for example `Call_Transaction_InsertArticle`).

6. Click on **Finish**.

   The new step appears in the **Steps** folder of the project.

Figure 2 - 178: New Call Transaction step in Steps folder

7. Save your project by clicking on  or by pressing `Ctrl + S`.

   Now that the *Call Transaction* step has been created, we will set its three main interrelated parameters:

- **Project** - in this case, the current project (`sample_doc_CMS`),
- **Connector -** in this case, the `sample_doc_CMS` project's sole connector (`DatabaseConnector`),
- **Transaction -** in this case, the `DatabaseConnector` connector's sole transaction (`InsertArticle`).

8 In the **Projects View**, select the step.

The **Properties View** is automatically updated. The default value of the **Project** property (`sample_doc_CMS`) and **Connector** property (`DatabaseConnector`) can be left as are.

9 In the **Properties View**, click in the **Value** column of the **Transaction** property.

The current value is highlighted and a drop-down symbol appears.

10 Click on .

11 Select the transaction (for example `InsertArticle`):

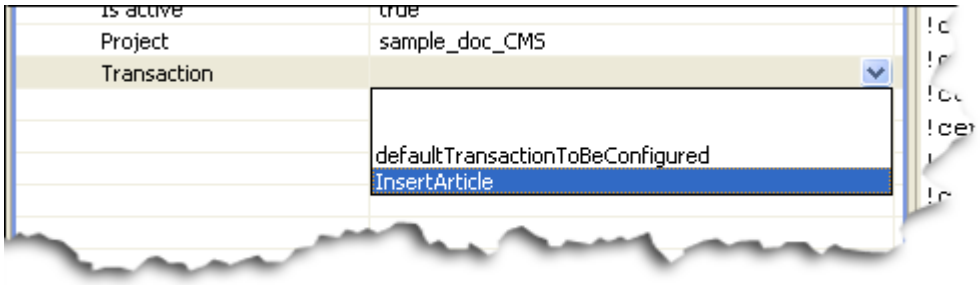


Figure 2 - 179: Setting the Transaction property of the Call Transaction step

12 Press `Enter`.

13 Save your project by clicking on or by pressing `Ctrl + S`.

The step is now set to call the `InsertArticle` target transaction.

Since the `InsertArticle` transaction expects a number of input variables, we will now import these variables into the required *Call Transaction* step.

The procedure for importing variables has already been described. For more information, see "To import variables from a target transaction at step level" on page 2-22.

14 Right-click on the step (for example `Call_Transaction_InsertArticle`).

A contextual menu appears:

15 Select **Import variables from target transaction**.

Once variables have been imported from the target transaction:

- the step appears as "changed and unsaved" (green bolded characters, see Figure 1 - 9),
- variables appear in a **Variables** sub-folder (together with the default value imported

from the target transaction between brackets, when applicable):

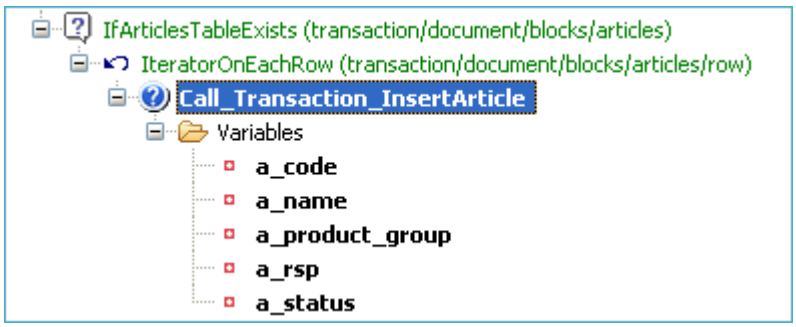


Figure 2 - 180: Variables imported from target transaction (and default values)

The *Call Transaction* step is now created and its variables are imported. These variables now need to be set.

What comes next?

Imported variables now need to be set so that their sources point towards the relevant nodes of each `articles` XML table row children nodes (`code`, `status` and `name`).

The table below describes the nodes towards which sources of the `Call_Transaction_InsertArticle` variables must point:

Table 2 - 14: `Call_Transaction_InsertArticle` step - Imported variable sources

The source of variable...	...must point to XML node...[a]	...generated by...
`a_article_code`	`code`	call to the `GetArticleData` transaction (second step)
`a_status`	`status`	
`a_name`	`name`	

a. *These nodes are children of the `articles` XML tabl `row` nodes generated by the mentioned step.*

These sources will now be set using the **Step Source** wizard.

The following procedure is similar to the procedure for creating and setting the variable sources of the first `Call_Transaction_InsertArticle` step (see "To set the variable sources of a Call Transaction step using the Step Source wizard" on page 2-96). For documentation lightness purpose, the procedure is repeated here with a limited number of snapshots.

To set the variable sources of the `Call_Transaction_InsertArticle` step using the Step Source wizard

1. Select the second `Call_Transaction_InsertArticle` step variable (for example `a_code`) with a left-click.

   The **Properties View** is automatically updated.

2. In the **Properties View**, click in the **Value** column of the **Source** property.

   A  button appears.

3. Click on the  button.

The **Step Source** wizard is automatically launched.

4 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

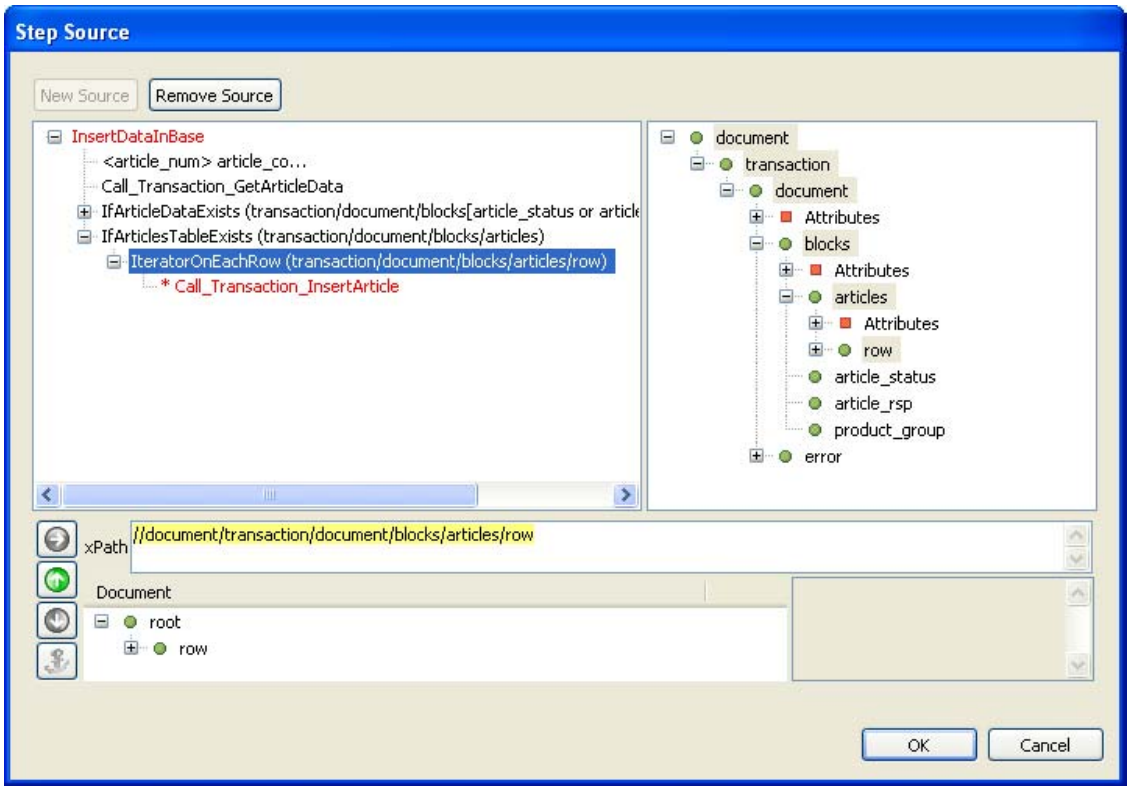


Figure 2 - 181: Setting of a_article_code variable source

The **Steps Tree Structure** displays all steps defined so far. The preceding step (*Iterator* step) is selected by default.

The `a_article_code` variable source must point towards the `code` node of the ***currently processed*** `row` node (ongoing iteration) in the `articles` XML table. The step selected by default (`IteratorOnEachRow`) is therefore correct, since its **XML Output Schema** corresponds to the currently processed `row` node (source of the `IteratorOnEachRow` step).

5 In the **XML Output Schema**, expand the `row` node by clicking on ⊞ :

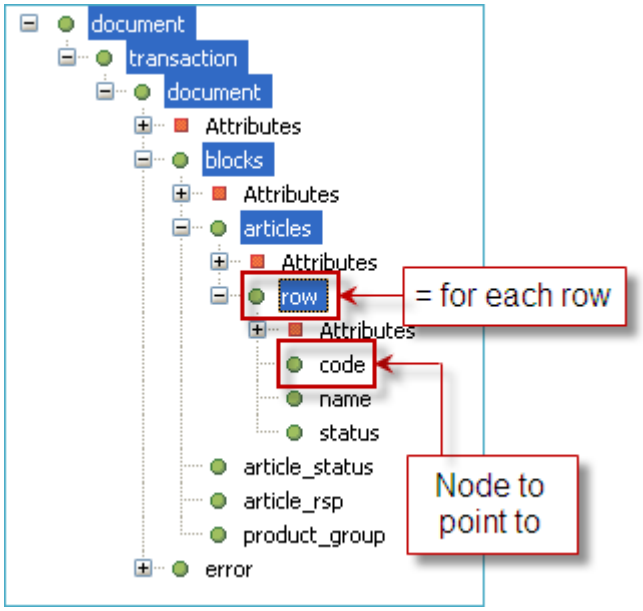


Figure 2 - 182: Source node for the a_article_code variable

6 Select the `code` node with a left-click.

The XPath to this node is automatically generated in the **XPath Evaluator**:

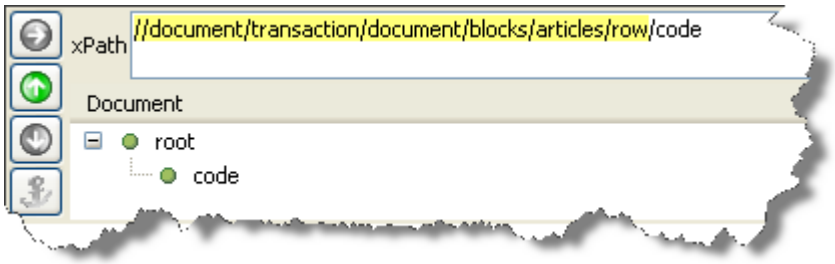


Figure 2 - 183: XPath to code node

The `a_article_code` variable source is now properly set.

7 Click on **OK**.

The variable source is updated.

8 Repeat points 1 to 7 to set sources for the remaining two variables (`a_status` and `a_name`, see Table *"`Call_Transaction_InsertArticle` step - Imported variable sources"* on page 2-110).

These two variables must point towards the `name` and `status` nodes in the **XML Output Schema**:

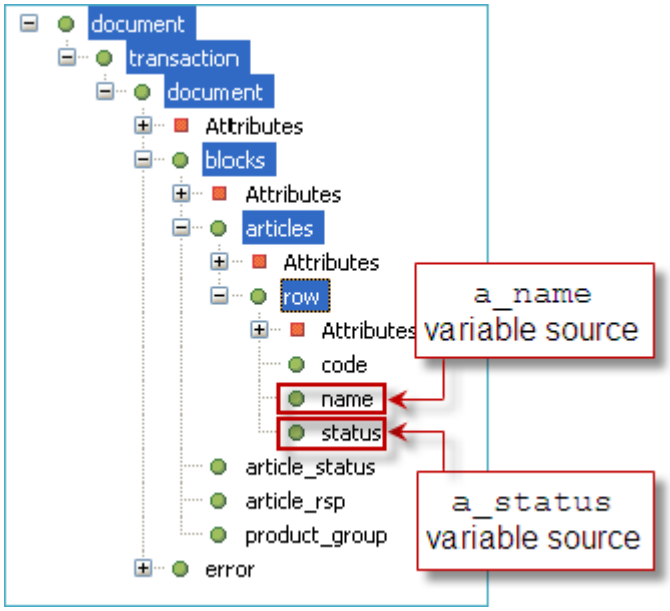


Figure 2 - 184: Sources of a_status and a_name variables

9 The sources of all variables are now properly set.

The `a_rsp` and `a_product_group` variables are not used at this point of the project. In this step, an empty value will be sent to the transaction when executed.

10 Save your project by clicking on  or by pressing `Ctrl + S`.

The second `Call_Transaction_InsertArticle` is now set.

WHAT COMES NEXT?

So far, only data generated by a CLI (Convertigo Legacy Integration) transaction have been processed.

We want now to use one of these data as input variable for a CWI (Convertigo Web Integration) transaction.

To this end, we will create another *Call Transaction* step (`Call_Transaction_SearchGoogle`) for calling the `searchGoogleWithLimit` transaction.

This step is defined as follows:

- it is called for each article of the `articles` XML table generated by the `GetArticleData` transaction,
- it calls the `searchGoogleWithLimit` CWI transaction, in order to use its generated XML as source for further steps and to get more data about each article,
- it uses as input variables the transaction input variables, `keyword` and `maxPages` (these variables must be imported from the target transaction). The source of the `keyword` step variable must point to the `name` node of the row currently processed by the `IteratorOnEachRow` step (meaning that the `Call_Transaction_SearchGoogle`

step is child to the `IteratorOnEachRow` step).

We will now create and set this step.

The following procedure is similar to the first sequence procedure for creating and setting the same Call Transaction step, `Call_Transaction_SearchGoogle` step (see "To create and set the `Call_Transaction_SearchGoogle` step" on page 2-57). For documentation lightness purpose, the procedure is repeated here with a limited number of snapshots.

To create and set the `Call_Transaction_SearchGoogle` step

1. In the **Projects View**, right-click on the parent step (for example `IteratorOnEachRow`).

   A contextual menu appears.

2. Select ***New > Step***.

   A **New Step** wizard is automatically launched.

3. In the **Other Steps** area of the window, select **Call Transaction** with a left-click.

4. Click on **Next**.

5. In the **Name** field that appears, type in the step name (for example `Call_Transaction_SearchGoogle`):

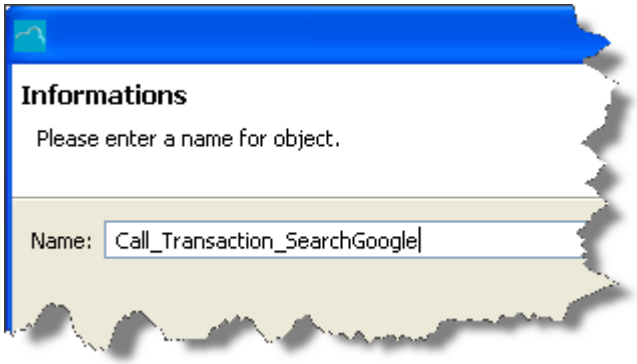


Figure 2 - 185: Giving a name to the step

6. Click on **Finish**.

   The new step appears in the **Steps** folder of the project:

Figure 2 - 186: New Call Transaction step in Steps folder

Now that the *Call Transaction* step has been created, we will set its three main interrelated parameters:

- **Project** - in this case, the CWI project (`sample_documentation_CWI`),
- **Connector -** in this case, the `sample_documentation_CWI` project's sole connector (`GoogleConnector`),
- **Transaction -** in this case, the `searchGoogleWithLimit` transaction.

7 In the **Projects View**, select the step.

The **Properties View** is automatically updated.

8 Click in the **Value** column of the **Project** property.

The current value (`sample_doc_CMS`) is highlighted and a drop-down symbol appears.

9 Click on .

10 Select the project containing the transaction to be called (for example `sample_documentation_CWI` for the `searchGoogleWithLimit` transaction):

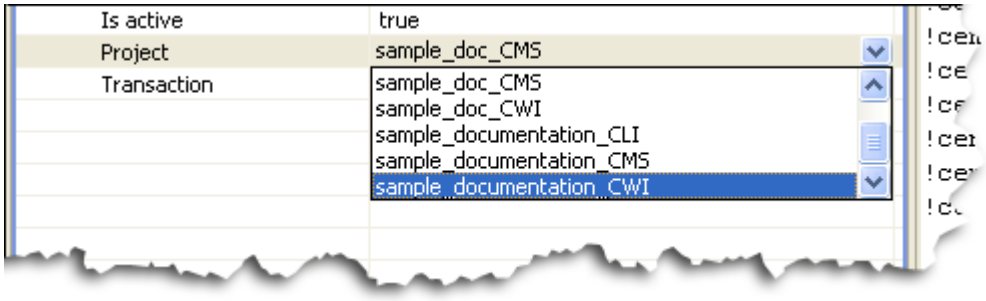


Figure 2 - 187: Setting the Project property of the Call Transaction step

11 Press `Enter`.

The values of the **Connector** and **Transaction** properties are updated.

12 Click in the **Value** column of the **Connector** property.

The current value is highlighted and a drop-down symbol appears.

13 Click on .

14 Select the connector containing the transaction to be called (for example `GoogleConnector` for the `searchGoogleWithLimit` transaction):

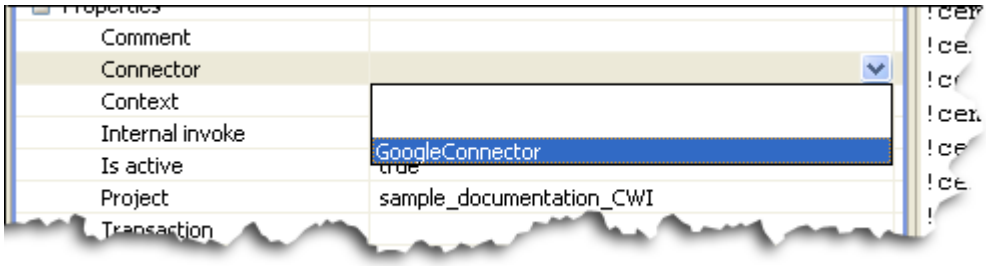


Figure 2 - 188: Setting the Connector property of the Call Transaction step

15 Press `Enter`.

The value of the **Transaction** property is updated.

16 Click in the **Value** column of the **Transaction** property.

The current value is highlighted and a drop-down symbol appears.

17 Click on .

18 Select the transaction (for example `searchGoogleWithLimit`):

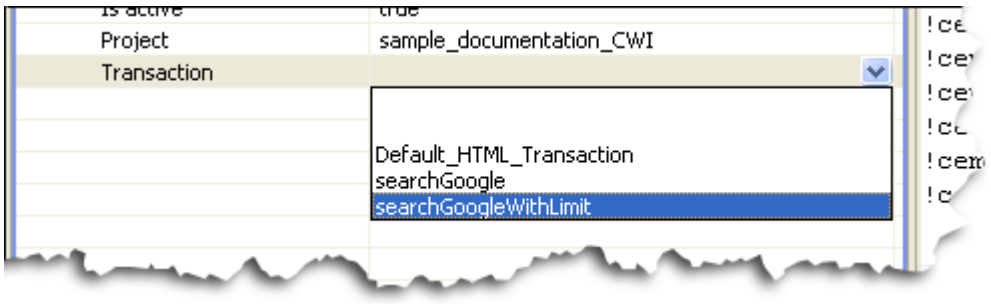


Figure 2 - 189: Setting the Transaction property of the Call Transaction step

19 Press `Enter`.

20 Save your project by clicking on or by pressing `Ctrl + S`.

The step is now set to call the `searchGoogleWithLimit` target transaction. Since the transaction expects a number of input variables, we will now import these variables from the `searchGoogleWithLimit` target transaction.

The procedure for importing variables has already been described. For more information, see "To import variables from a target transaction at step level" on page 2-22.

21 Right-click on the step (for example `Call_Transaction_SearchGoogle`).

A contextual menu appears:

22 Select **Import variables from target transaction**.

Once variables have been imported from the target transaction:

- the step appears as "changed and unsaved" (green bolded characters, see Figure 1 - 9),
- variables appear in a **Variables** sub-folder (together with the default value imported from the target transaction between brackets, when applicable):

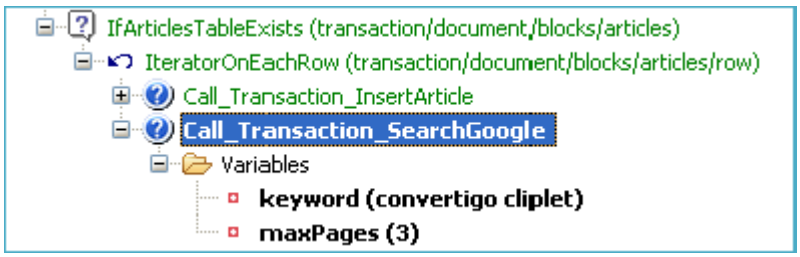


Figure 2 - 190: Variables imported from target transaction (and default values)

The *Call Transaction* step is now created and its variables are imported. These variables now need to be set.

The `keyword` variable source needs to point towards the `name` node of the `articles` XML table row currently processed by the `IteratorOnEachRow` step.

This source will now be set using the **Step Source** wizard.

To set the variable sources of the `Call_Transaction_SearchGoogle` step using the Step Source wizard

1 Select the *Call Transaction* step `keyword` variable with a left-click.

The **Properties View** is automatically updated:

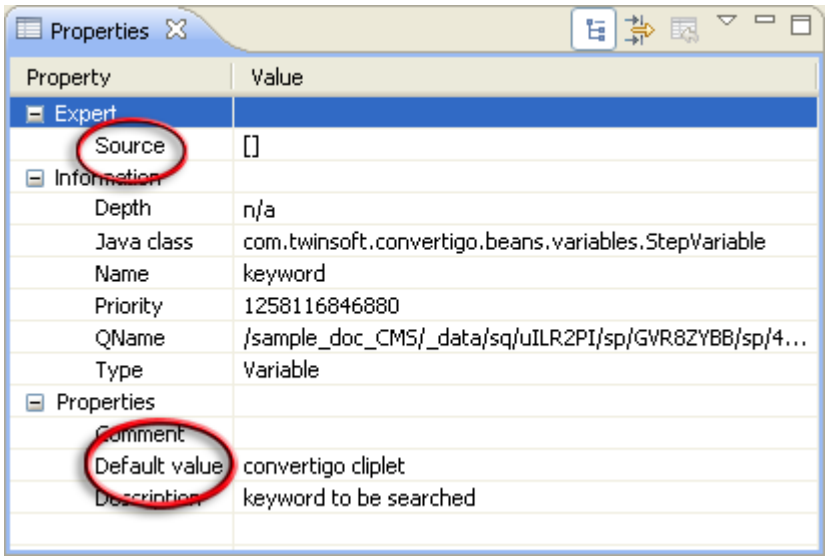


Figure 2 - 191: Properties of Call Transaction step keyword variable

2 In the **Properties View**, click in the **Value** column of the **Default Value** property.

The default value (imported from the transaction) is highlighted.

3 If applicable, delete the variable default value (because the variable value is sourced from the *Iterator* step).

4 Press `Enter`.

5 Click in the **Value** column of the **Source** property.

The value is highlighted and a  button appears to the right of the field.

6 Click on the  button.

The **Step Source** wizard is automatically launched.

For more information on the Step Source wizard, see Table "Step Source wizard description" on page 1-16.

7 Click on **New Source** in the upper left corner of the window.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

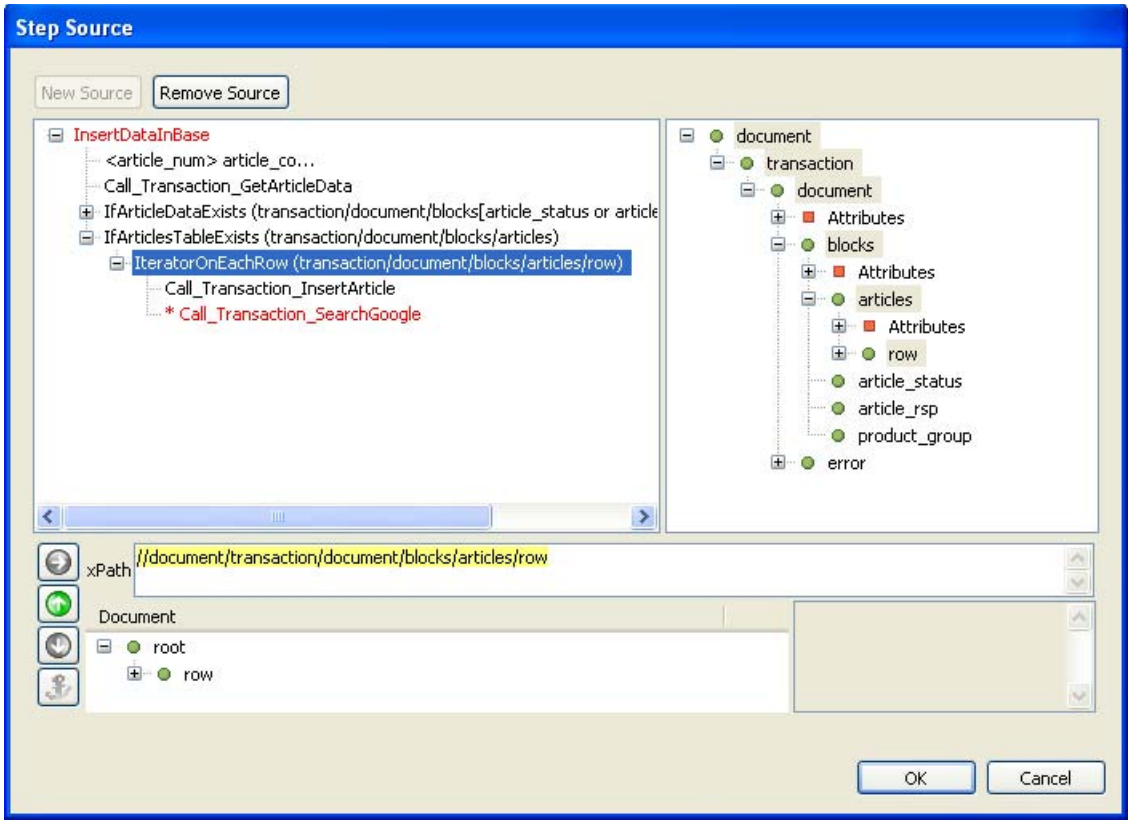


Figure 2 - 192: Setting of keyword variable source

The **Steps Tree Structure** displays all steps defined so far. The *Iterator* step is selected by default.

The `keyword` variable source must point towards the `name` node of the ***currently processed*** `row` node (ongoing iteration) in the `articles` XML table. The step selected by default (`IteratorOnEachRow`) is therefore correct, since its **XML Output Schema** corresponds to the currently processed row (source of the `IteratorOnEachRow` step).

In the **XML Output Schema**, the `row` node as well as its parent nodes. This corresponds to the pregenerated *Iterator* step source XPath (highlighted in yellow in the **xPath** field of the **XPath Evaluator**).

8 In the **XML Output Schema**, expand the `row` node by clicking on ⊞:

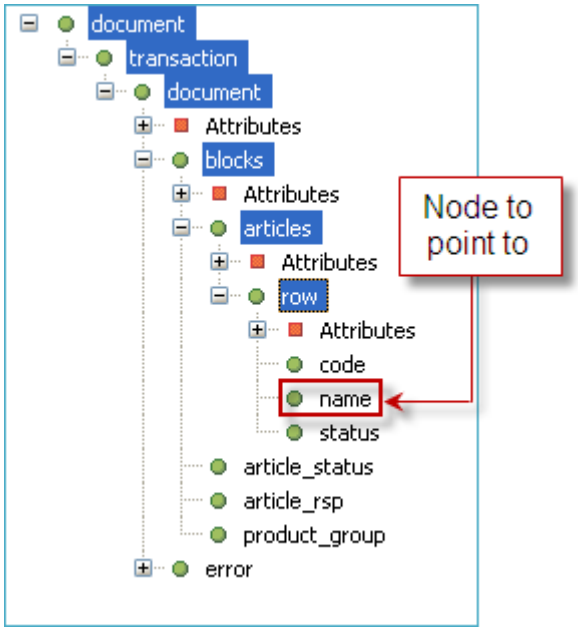


Figure 2 - 193: Source node for the keyword variable

9 Select the `name` node with a left-click.

The XPath to this node is automatically generated in the **XPath Evaluator**:

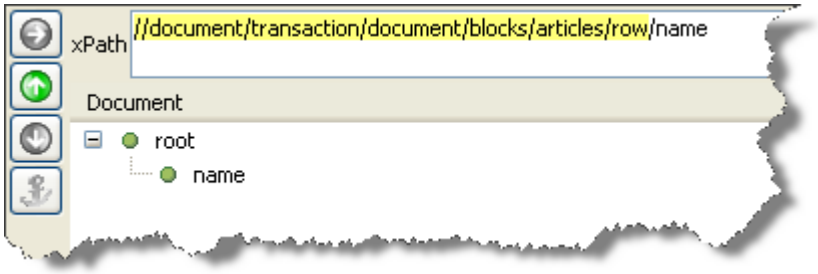


Figure 2 - 194: XPath to name node

The `keyword` variable source is now properly set.

10 Click on **OK**.

The variable source value is automatically updated in the **Properties View**.

We will now set the `maxPages` variable. Unlike other variables set so far, the `maxPages` variable can be given a fixed integer value. No source will be set for this variable, only a default value.

1 Select the *Call Transaction* step `maxPages` variable with a left-click.

The **Properties View** is automatically updated:

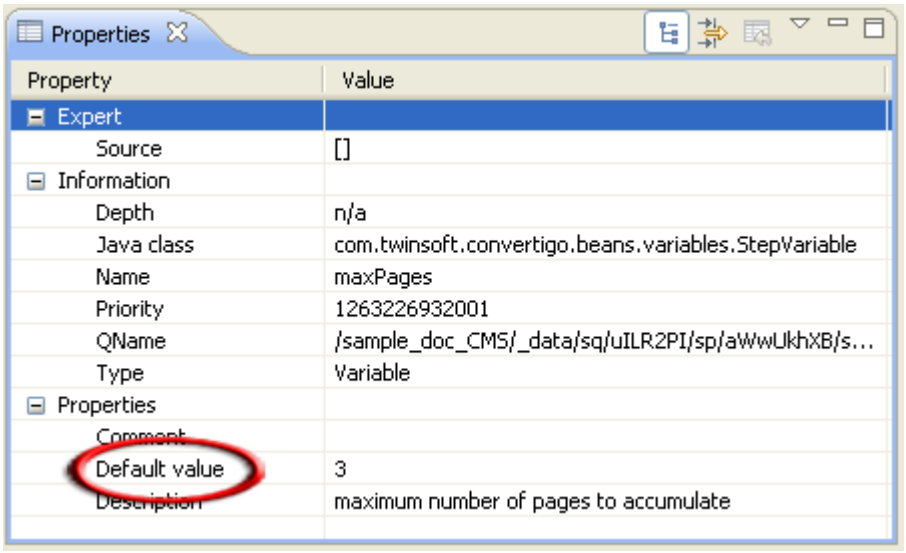


Figure 2 - 195: Properties of Call Transaction step maxPages variable

2 In the **Properties View**, click in the **Value** column of the **Default Value** property.

The default value (imported from the transaction) is highlighted.

3 Type in the required value (2 is a correct value in this project) or leave it unchanged.

4 Press `Enter`.

The variable is updated.

5 Save your project by clicking on  or by pressing `Ctrl + S`.

The `Call_Transaction_SearchGoogle` is now properly set to:

- call the `searchGoogleWithLimit` CWI transaction,
- enter as keyword the value of each `name` node returned by the `GetArticleData` transaction,
- limit the number of pages returned by Google to 2.

What comes next?

When called, the `searchGoogleWithLimit` transaction generates an XML table defined as follows:

- the generated XML table is called `listResults`;
- it contains as many `resultItem` rows as result items returned by the Google search engine
- each `resultItem` row has two columns: `title` and `url`.

For more information on the `searchGoogleWithLimit` transaction (which is similar to the `searchGoogle` transaction), refer to the "Starting with Convertigo Web Integrator" Quick Guide.

According to the project description (see *"Second sequence description"* on page 2-5), next steps now consist in:

1 *Checking* the presence of the `listResults` table in the XML output generated when calling the `searchGoogleWithLimit` transaction.

2 If the `listResults` is present, iterating on each `resultItem` row and inserting into the `web_sites` table the content of the:

- `code` node (included in the `articles` XML table generated by the CLI `GetArticleData` transaction);
- `url` and `title` nodes (included in the `listResults` XML table generated by the CWI `searchGoogleWithLimit` transaction);

by:

- *creating* and *setting* an `InsertWebSite` SQL transaction,
- *creating* and *setting* a `Call_InsertWebSite` step.

Point 1 of this procedure requires setting up a step similar, in a way, to the *IfExist* step created for checking the presence of the `articles` XML table generated when calling the `GetArticleData` transaction (see *"To create and set the `IfArticlesTableExists` step"* on page 2-101), so a third *IfExist* step will be needed in the sequence.

But we also know that a second SQL transaction, `InsertWebSite`, needs to be called for inserting article code, url and title information into the `web_sites` database table.

We will first set this second SQL transaction.

To create and set the `InsertWebSite` SQL transaction

1 Right-click on the connector (for example `DatabaseConnector`).

A contextual menu appears:

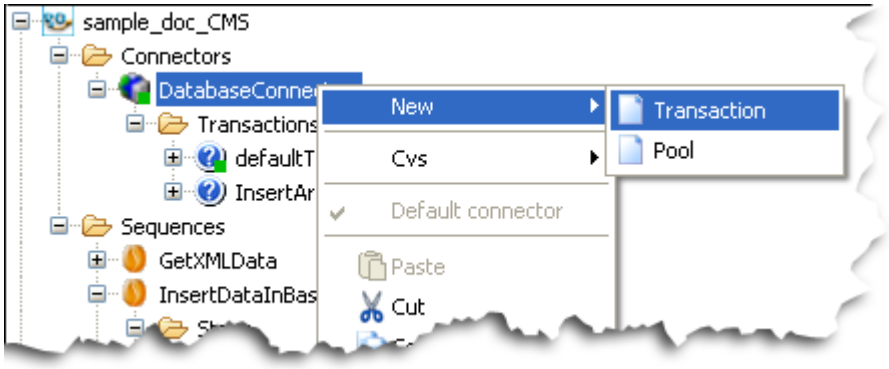


Figure 2 - 196: Creating a new transaction

2 Select ***New > Transaction***.

A **New Transaction** wizard is automatically launched.

3 Select **SQL Transaction**:

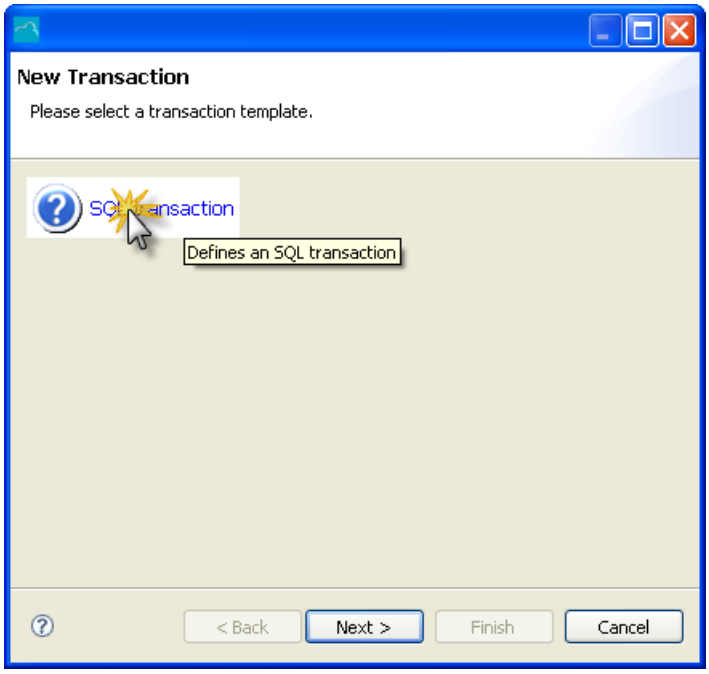


Figure 2 - 197: New Transaction wizard

4 Click on **Next**.

5 In the **Name** field that appears, type in the name of the transaction (for example `InsertWebSite`):

Figure 2 - 198: Giving a name to the new SQL transaction

6 Click on **Finish**.

The SQL transaction is now created. It appears unsaved in the **Transactions** folder of the connector:

Figure 2 - 199: New SQL transaction in Transactions folder

We will now set the transaction main property: the **Query** property, which contains the

transaction SQL query.

7 Select the SQL transaction with a left-click.

The **Properties View** is automatically updated:

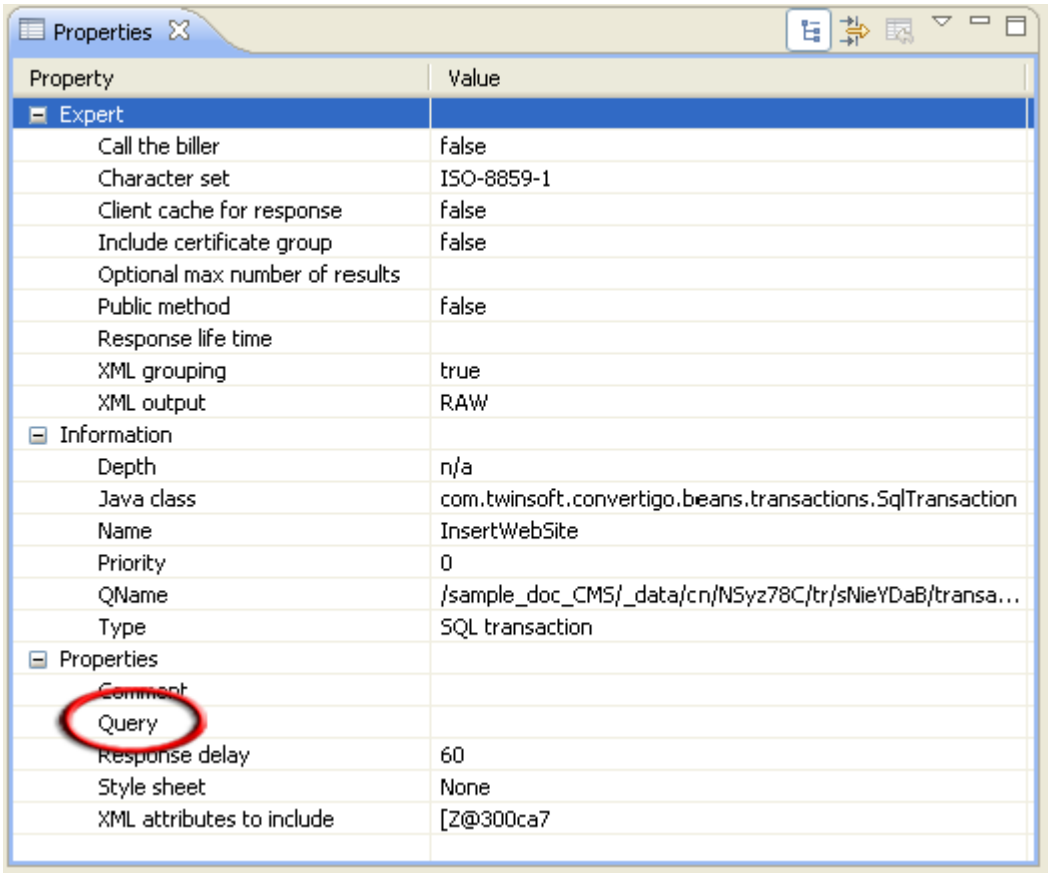


Figure 2 - 200: SQL transaction properties

8 Click in the **Value** column of the **Query** property.

A button appears.

9 Click on the button.

An **SQL Query** window opens:

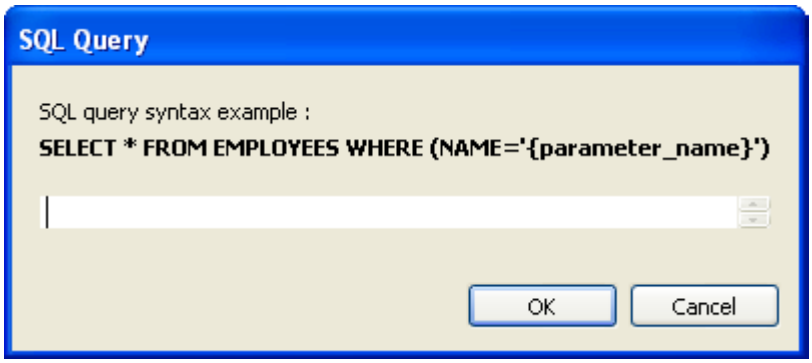


Figure 2 - 201: SQL query window

The information that we have for writing the insertion query is the following:

- data must be inserted into the `article_code`, `web_site_url` and

`web_site_title` columns of the `web_sites` table (see Table 2 - 2 on page 2-3),

- the value of these data being variable, they will be inserted in the form of variables (called respectively `a_code`, `ws_url` and `ws_title`) , the source of which will point towards the `code` XML node generated by the called `searchGoogleWithLimit` transaction and the `url` and `title` XML nodes generated by the called `GetArticleData` transaction,
- the SQL syntax for inserting data into a table using variable values is: `INSERT INTO tableName (tableColumn1, tableColumn2, ..., tableColumnX) VALUES ('{variableName1}', '{variableName2}',..., '{variableNameX}';`

Given this information, the value of the **Query** property must be set as:

```
INSERT INTO web_sites (article_code, web_site_url, web_site_title)

VALUES('{a_code}', '{ws_url}', '{ws_title}');
```

Figure 2 - 202: SQL query for inserting required data into web_sites table

10 Enter in the **SQL Query** window the required SQL query (see Figure 2 - 202).

11 Click on **OK**.

The transaction variables are automatically generated in the **Variables** folder of the transaction:

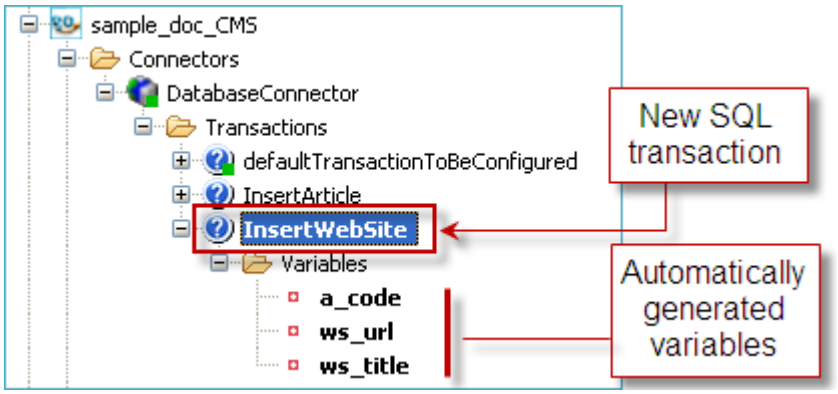


Figure 2 - 203: New SQL Transaction and generated variables

12 Save your project by clicking on  or by pressing `Ctrl + S`.

Now that the SQL transaction is created and properly set, we will test it, set it as a public method and update its XML schema so that the SQL transaction is fully usable by *CallTransaction* steps.

13 Test the SQL transaction, set it as a public method and update its XML schema using the suitable procedure (see *"Testing an SQL Transaction and Updating its XML Schema"* on page 2-89).

The `InsertWebSite` SQL transaction is now tested and set as a public method, and its XML schema has been updated.

WHAT COMES NEXT?

We will now switch back to the sequence and create an *IfExist* step (called `IfListResultsTableExists`) for checking the presence of the `listResults` table in the XML output generated when calling the `searchGoogleWithLimit` transaction.

The following procedure is similar to the previously described procedure for checking the presence of the `articles` XML table generated when calling the `GetArticleData` transaction (see "To create and set the `IfArticlesTableExists` step" on page 2-101). For documentation lightness purpose, the procedure is repeated here with a limited number of snapshots.

To create and set the `IfListResultsTableExists` step

1. Right-click on the parent step (for example `IteratorOnEachRow`).

   A contextual menu appears.

2. Select ***New > Step***.

   The **New Step** wizard is automatically launched.

3. In the **Other steps** area, select **IfExist**.

4. Click on **Next**.

5. In the **Name** field that appears, type in the *IfExist* step name (for example `IfListResultsTableExists`).

6. Click on **Finish**.

   The new step appears at the end of the sequence **Steps** folder in the **Projects View**:

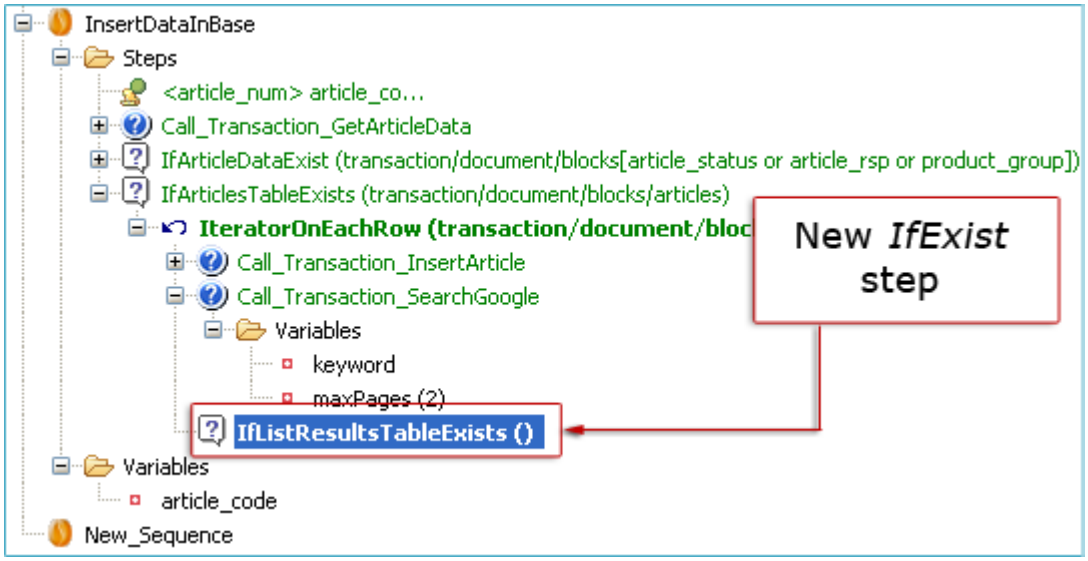


Figure 2 - 204: New IfExist step in Steps folder

7. Save your project by clicking on  or by pressing `Ctrl + S`.

   Now that the *ifExist* step has been created, its source must be set.

   The step source needs to point towards the XML node (`listResults`) to be checked

within the XML output returned by the `searchGoogleWithLimit` transaction (called by the `Call_Transaction_searchGoogleWithLimit` step).

We will now set the step source.

8 Select the *IfExist* step with a left-click.

The **Properties View** is automatically updated.

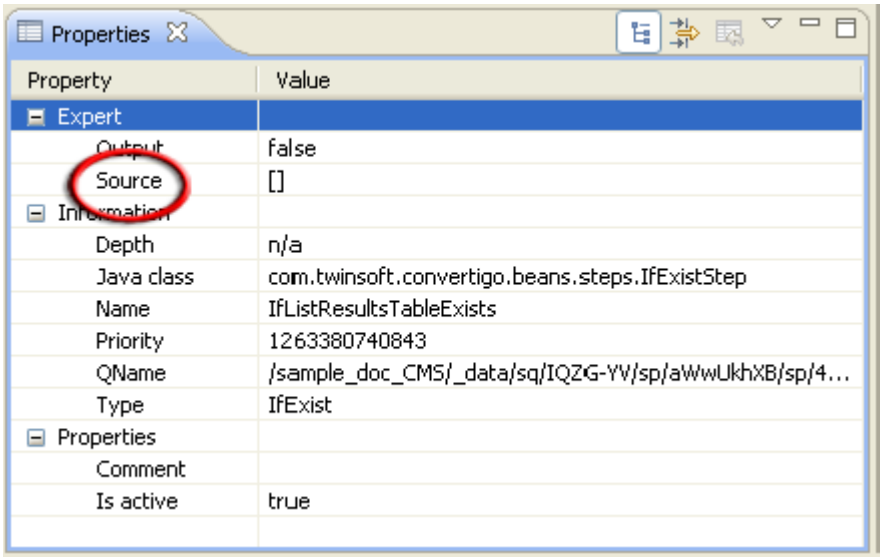


Figure 2 - 205: IfExist step properties

9 In the **Properties View**, click in the **Value** column of the **Source** property.

A button appears.

10 Click on the button to launch the **Step Source** wizard.

The **Step Source** wizard is automatically launched.

11 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

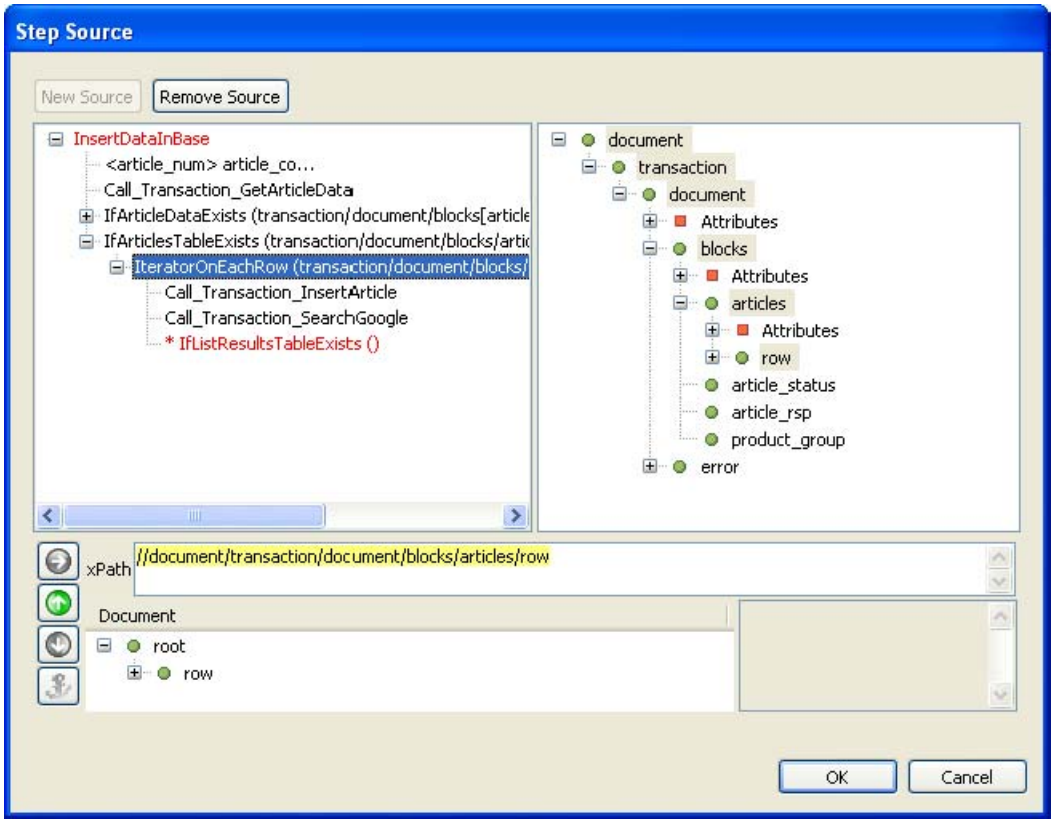


Figure 2 - 206: Setting of IfListresultsTableExists step source

The *IfExist* check must be performed on the XML schema output generated when calling the `searchGoogleWithLimit` transaction. The step to be selected is therefore `Call_Transaction_searchGoogle`.

12 In the **Steps Tree Structure**, select the required step (for example `Call_Transaction_searchGoogle`) with a left-click.

The selected step's **XML Output Schema** is displayed:

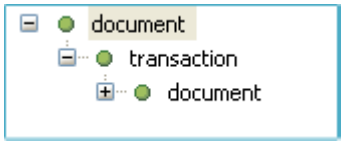


Figure 2 - 207: XML Schema of Call_transaction_SearchGoogle output

13 In the **XML Output Schema**, expand the `document` node, then select the node to be checked (`listResults`):

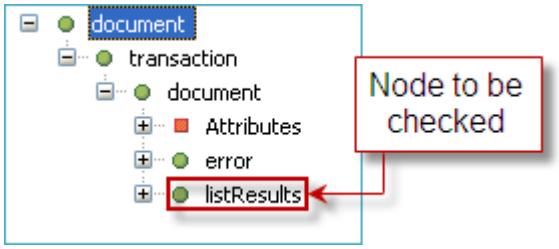


Figure 2 - 208: Node to check in the called transaction XML output

14 The XPath expression to this node is automatically generated in the **XPath Evaluator**:

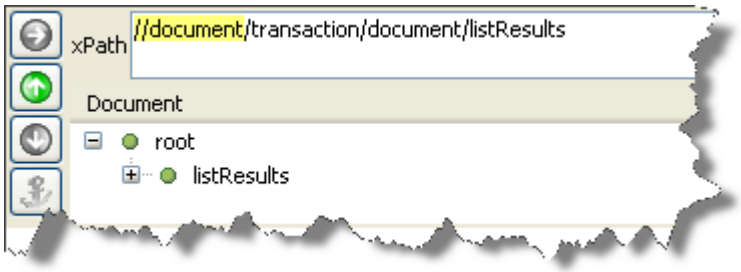


Figure 2 - 209: Generation of XPath to listResults node

15 Click on **OK**.

The **Source** step property is automatically updated:

Source	[1258382563518, ./transaction/document/listResults]

Figure 2 - 210: Source property automatically updated in Properties View

16 Save your project by clicking on  or by pressing `Ctrl + S`.

The *IfExist* step is now created and properly set.

WHAT COMES NEXT?

The following milestone in the project consists in iterating on each row of the `listResults` XML table (for an example of `listresults` table, refer to the *"My First Convertigo Mashup Sequencer Project" Quick Guide*) to subsequently perform operations on these rows (for example insert row data into database table by calling an SQL transaction).

To this end, we will now create and set an *Iterator* step called `IteratorOnEachResultItem`.

This *Iterator* step iterates on each `resultItem` row of the `listresults` XML table generated when executing the `Call_Transaction_SearchGoogle`. Following steps then use this step as a source for subsequent operations.

The following procedure is similar to the procedure for creating and setting the `IteratorOnEachRow` step (see "To create and set an Iterator step" on page 2-104). For documentation lightness purpose, the procedure is repeated here with a limited number of snapshots.

To create and set the `IteratorOnEachResultItem` step

1 In the **Projects View**, right-click on the parent step of the *Iterator* step (for example `IfListResultsTableExists`).

A contextual menu appears.

2 Select ***New > Step***.

A **New Step** wizard is automatically launched:

3 In the **Other Steps** area of the window, select **Iterator** with a left-click.

4 Click on **Next**.

5 In the **Name** field that appears, type in the step name (for example `IteratorOnEachResultItem`):

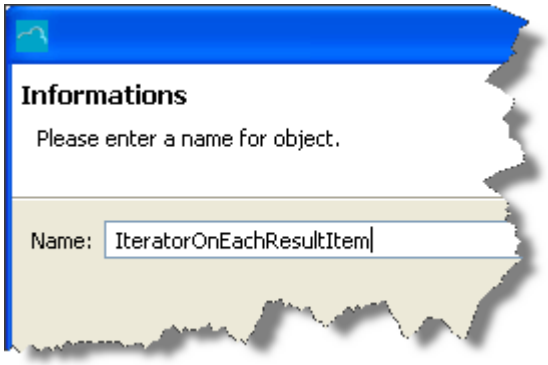


Figure 2 - 211: Giving a name to the step

6 Click on **Finish**.

The new step appears in the **Steps** folder of the project as a child of the `IfListResultsTableExists` step, upon which it depends (no iteration is needed if no `listResults` XML table exists):

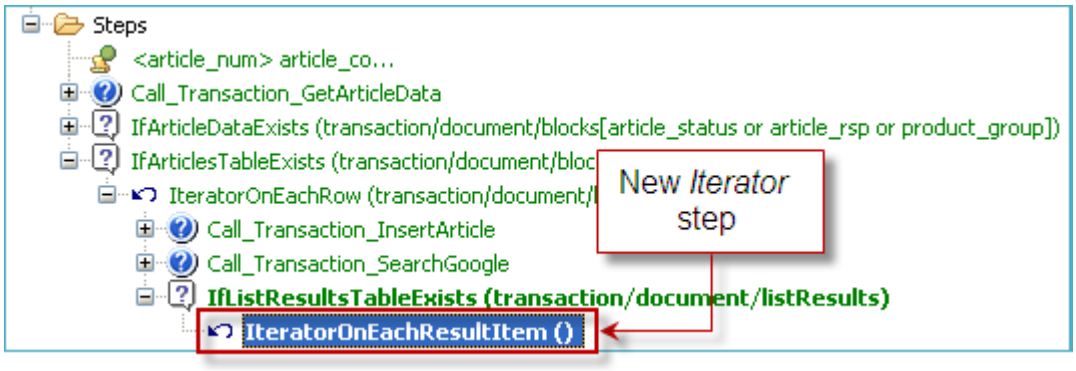


Figure 2 - 212: New Iterator step in Steps folder

Now that the *Iterator* step has been created, we will set its **Source** property so that it points towards the required node (`resultItem`) of the `Call_Transaction_SearchGoogle` step XML schema.

7 Select the `IteratorOnEachResultItem` step with a left-click.

The **Properties View** is automatically updated:

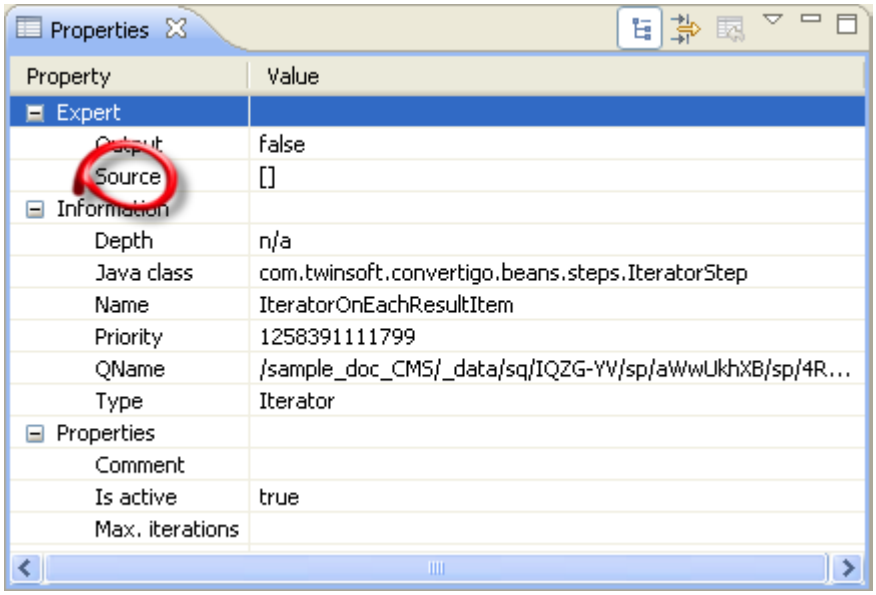


Figure 2 - 213: Iterator step properties

8 In the **Properties View**, click in the **Value** column of the **Source** property.

A [...] button appears.

9 Click on the [...] button.

The **Step Source** wizard is automatically launched.

10 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

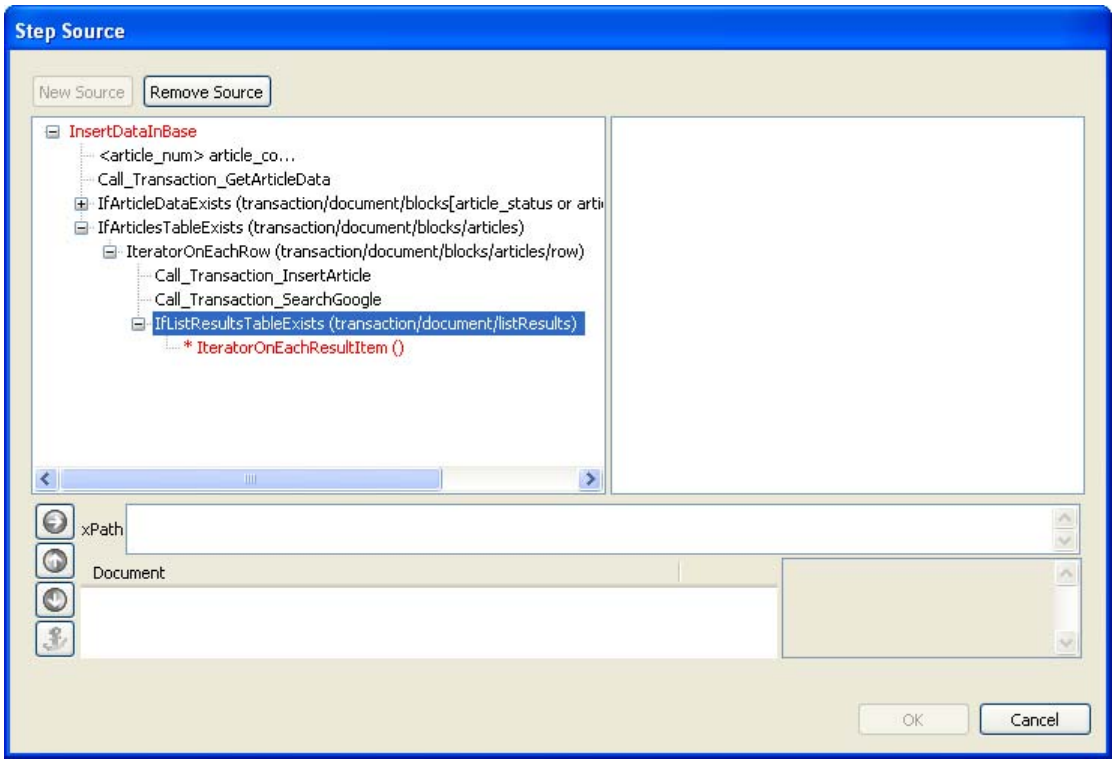


Figure 2 - 214: Generating iterator step source

The iteration must be performed on each `resulItem` node of the `listResults` XML table included in the output generated when calling the `searchGoogleWithLimit` transaction. The step to be selected is therefore `Call_Transaction_searchGoogle`.

11 In the **Steps Tree Structure**, select the required step (for example `Call_Transaction_searchGoogle`) with a left-click.

The selected step's **XML Output Schema** is displayed:

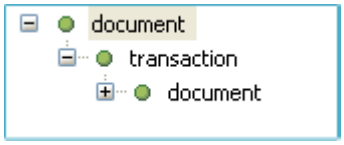


Figure 2 - 215: XML Schema of Call_transaction_searchGoogle step

12 Expand the `document`, then `listResults` node including the `resultItem` nodes to be selected by clicking on ⊞:

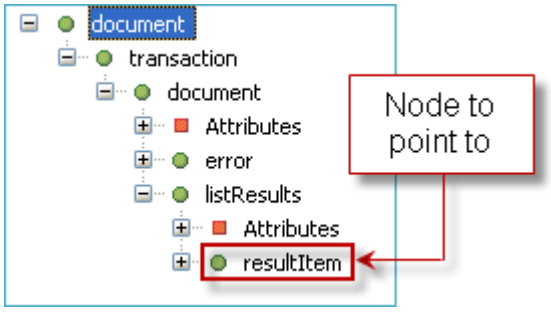


Figure 2 - 216: Source of Iterator step

13 In the **XML Output Schema**, select the node to point to (`resultItem`).

14 The XPath expression to this node is automatically generated in the **XPath Evaluator**:

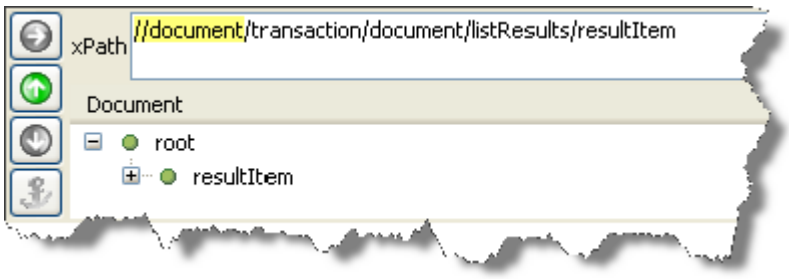


Figure 2 - 217: Generation of XPath to resulItem node

15 Click on **OK**.

The **Source** step property is automatically updated:

Figure 2 - 218: Source property automatically updated in Properties View

16 Save your project by clicking on  or by pressing `Ctrl + S`.

WHAT COMES NEXT?

Each `resultItem` node being parent to two nodes - `url` and `title` - the `IteratorOnEachResultItem` step will therefore serve as a source for its child step, `Call_Transaction_InsertWebSite`.

The source of the `Call_Transaction_InsertWebSite` step variables must indeed point to the `code` node of ***the currently iterated row*** and to the `url` and `title` nodes of ***the currently iterated resultItem***, for the `InsertWebSite` transaction to insert the content of these nodes into the `web_sites` table.

The following step therefore consists in creating the `Call_Transaction_InsertWebSite` step.

The following procedure is similar to the first sequence procedure for creating and setting another Call Transaction step, `Call_Transaction_SearchGoogle` step (see "To create and set the `Call_Transaction_SearchGoogle` step" on page 2-57). For documentation lightness purpose, the procedure is repeated here with a limited number of snapshots.

To create and set the `Call_Transaction_InsertWebSite` step

1. In the **Projects View**, right-click on the parent step (`IteratorOnEachResultItem`).

   A contextual menu appears.

2. Select ***New > Step***.

   A **New Step** wizard is automatically launched.

3. In the **Other Steps** area of the window, select **Call Transaction** with a left-click.

4. Click on **Next**.

5. In the **Name** field that appears, type in the step name (for example `Call_Transaction_InsertWebSite`).

6. Click on **Finish**.

   The new step appears in the **Steps** folder of the project.

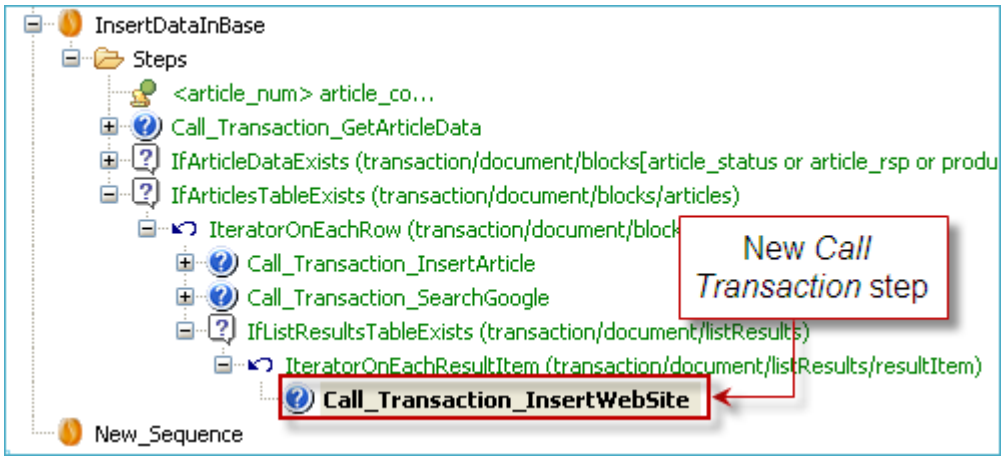


Figure 2 - 219: New Call Transaction step in Steps folder

7. Save your project by clicking on  or by pressing `Ctrl + S`.

Now that the *Call Transaction* step has been created, we will set its three main interrelated parameters:

- **Project** - in this case, the current project (`sample_doc_CMS`),
- **Connector -** in this case, the `sample_doc_CMS` project's sole connector (`DatabaseConnector`),
- **Transaction -** in this case, the second transaction of `DatabaseConnector` connector (`InsertWebSite`).

8 In the **Projects View**, select the step.

The **Properties View** is automatically updated. The default value of the **Project** property (`sample_doc_CMS`) and **Connector** property (`DatabaseConnector`) can be left as are.

9 In the **Properties View**, click in the **Value** column of the **Transaction** property.

The current value is highlighted and a drop-down symbol appears.

10 Click on .

11 Select the transaction (for example `InsertWebSite`):

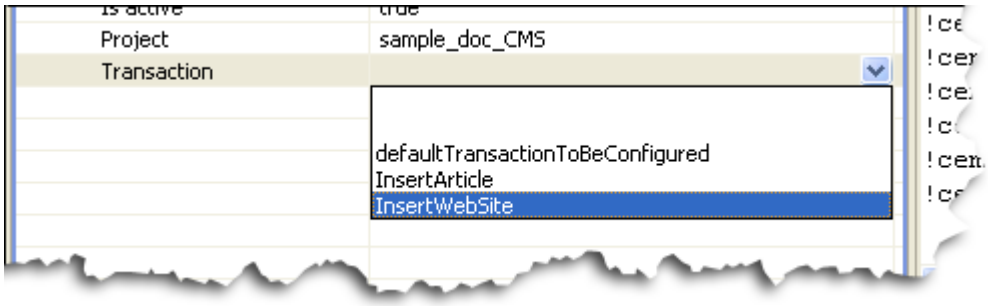


Figure 2 - 220: Setting the Transaction property of the Call Transaction step

12 Press `Enter`.

13 Save your project by clicking on or by pressing `Ctrl + S`.

The step is now set to call the `InsertWebSite` target SQL transaction.

Since the transaction expects a number of input variables, we will now import these variables from the `InsertWebSite` target SQL transaction.

The procedure for importing variables has already been described. For more information, see "To import variables from a target transaction at step level" on page 2-22.

14 Right-click on the step (for example `Call_Transaction_InsertWebSite`).

A contextual menu appears:

15 Select **Import variables from target transaction**.

Once variables have been imported from the target transaction:

- the step appears as "changed and unsaved" (green bolded characters, see Figure 1

- 9),

- variables appear in a **Variables** sub-folder (together with the default value imported from the target transaction between brackets, when applicable):

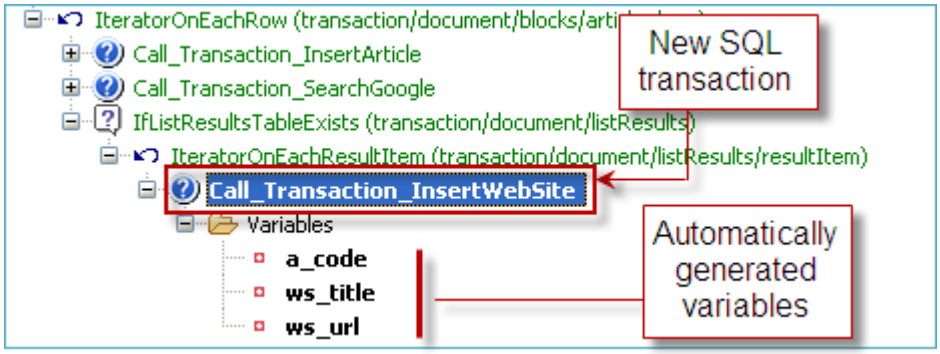


Figure 2 - 221: Variables imported from target transaction (and default values)

The *Call Transaction* step is now created and its variables are imported. These variables now need to be set.

Imported variables now need to be set so that their sources point towards:

- the `code` node of ***each iterated*** `articles` XML table `row` node generated by the `Call_Transaction_GetArticleData` step;
- the `url` and `title` nodes of ***each iterated*** `listresults` XML table `resultItem` node generated by the `Call_Transaction_searchGoogle` step.

The table below describes the nodes towards which sources of the `Call_Transaction_InsertWebSite` variables must point:

Table 2 - 15: `Call_Transaction_InsertWebSite` step - Imported variable sources

The source of variable...	...must point to XML node...	...of the XML output generated by step...
`a_code`	`articles/row/code`	calling the `GetArticleData` transaction
`ws_url`	`listResults/resultItem/url`	calling the `searchGoogleWithLimit` transaction
`ws_title`	`listResults/resultItem/title`	

These sources will now be set using the **Step Source** wizard.

The procedure for setting the variable sources of a Call Transaction step has already been described (see "To set the variable sources of a Call Transaction step using the Step Source wizard" on page 2-96). For documentation lightness purpose, the procedure is repeated here with a limited number of snapshots.

To set the variable sources of the `Call_Transaction_InsertWebSite` step using the Step Source wizard

1 Select the first *Call Transaction* step variable (`a_code`) with a left-click.

The **Properties View** is automatically updated.

2 In the **Properties View**, click in the **Value** column of the **Source** property.

A button appears.

The **Step Source** wizard is automatically launched.

3 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

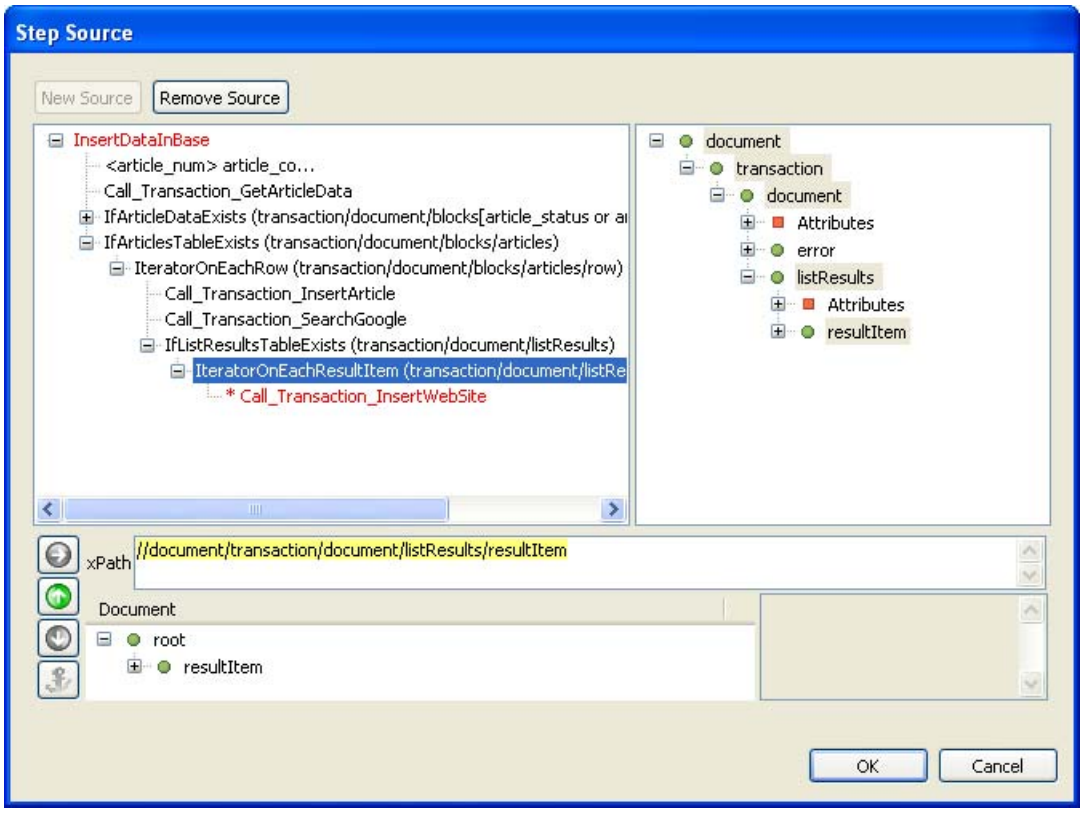


Figure 2 - 222: Setting of a_code variable source

The **Steps Tree Structure** displays all steps defined so far. The preceding step (`IteratorOnEachResultItem` step) is selected by default.

As mentioned in Table *"`Call_Transaction_InsertArticle` step - Imported variable sources"* on page 2-96, the `a_code` variable source must point towards the `code` node of the ***currently processed*** `row` node (ongoing iteration) in the `articles` XML table generated by the `GetArticleData` transaction.

We must therefore select the `IteratorOnEachRow` step, then select the `code` node.

4 In the **Steps Tree Structure**, select the `IteratorOnEachRow` step.

The selected step's **XML Output Schema** is displayed:

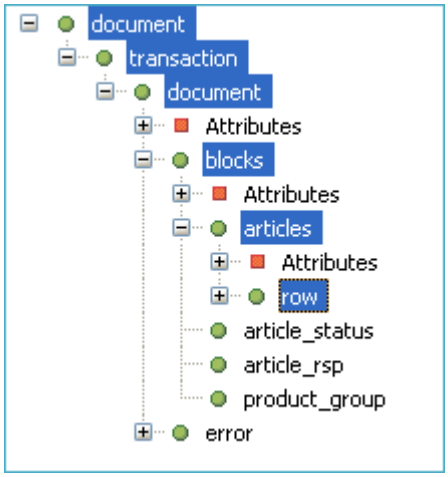


Figure 2 - 223: IteratorOnEachRow step XML Schema

5 In the **XML Output Schema**, expand the `row` node by clicking on ⊞:

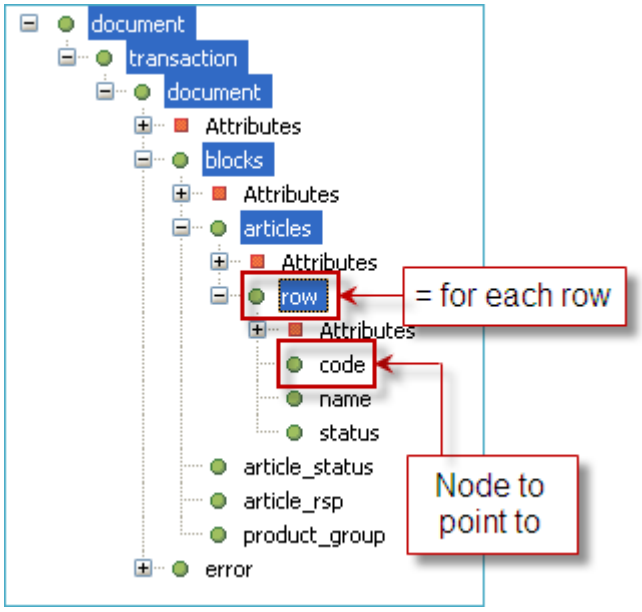


Figure 2 - 224: Source node for the code variable

6 Select the `code` node with a left-click.

The XPath to this node is automatically generated in the **XPath Evaluator**:

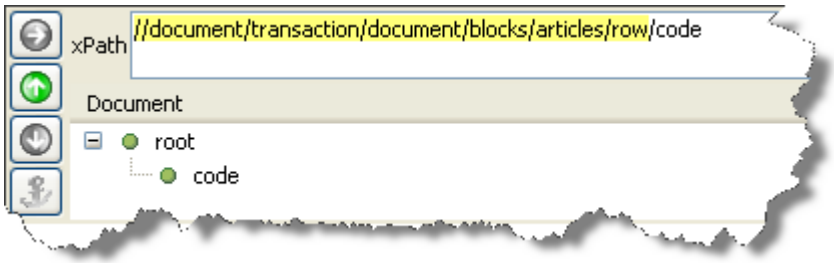


Figure 2 - 225: XPath to code node

The `a_article_code` variable source is now properly set.

7 Click on **OK**.

The variable source is automatically updated in the **Transactions variable** window.

We will now set the source of the other two variables: `url` and `title`.

8 Select the second *Call Transaction* step variable (`ws_title`) with a left-click.

The **Properties View** is automatically updated.

9 In the **Properties View**, click in the **Value** column of the **Source** property.

A  button appears.

The **Step Source** wizard is automatically launched.

10 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

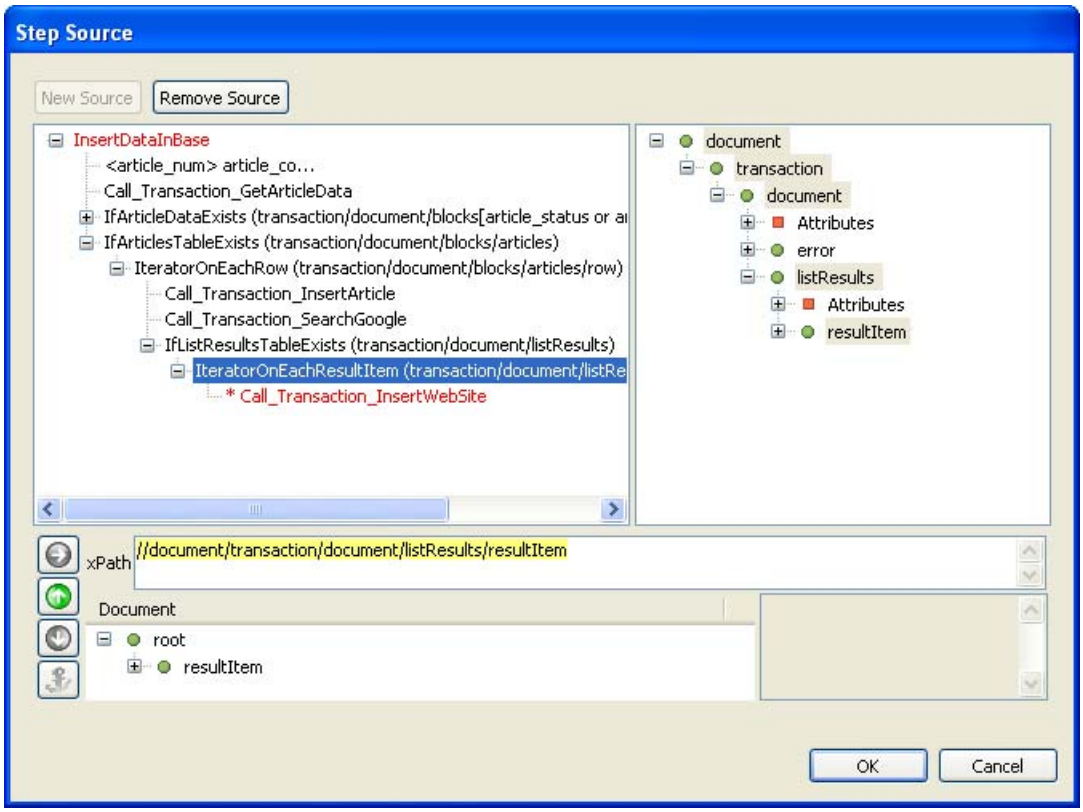


Figure 2 - 226: Setting of ws_title variable source

The **Steps Tree Structure** displays all steps defined so far. The preceding step (`IteratorOnEachResultItem` step) is selected by default.

As mentioned in Table *"`Call_Transaction_InsertArticle` step - Imported variable sources"* on page 2-96, the `ws_title` variable source must point towards the `title` node of the ***currently processed*** `resultItem` row (ongoing iteration) in the `listResults` XML table. The step selected by default (`IteratorOnEachResultItem`) is therefore correct, since its **XML Output Schema** corresponds to the currently processed `resultItem` row (source of `IteratorOnEachResultItem` step).

11 In the **XML Output Schema**, expand the `resultItem` node:

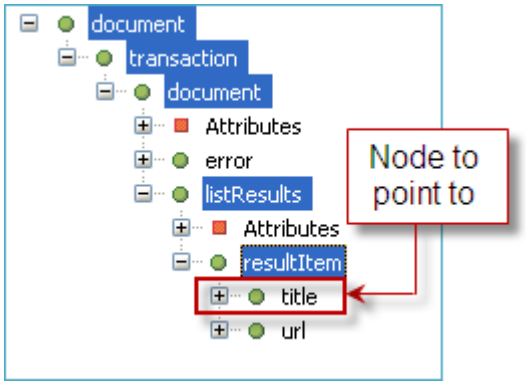


Figure 2 - 227: Source node for the ws_title variable

12 Select the `title` node.

The XPath to this node is automatically generated in the **XPath Evaluator**:

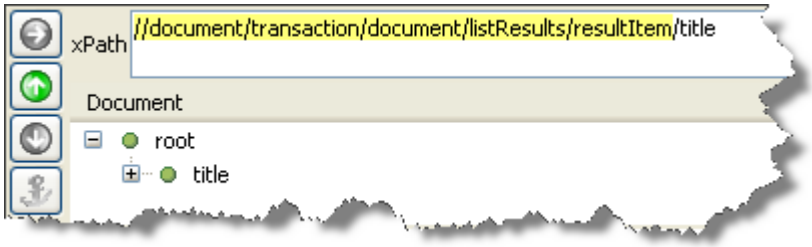


Figure 2 - 228: XPath to title node

The `ws_title` variable source is now properly set.

13 Click on **OK**.

The variable source is automatically updated in the **Transactions variable** window.

14 Repeat points 8 to 13 to set up the source of the remaining variable: `ws_url`, selecting the `url` node in the **XML Output Schema**.

15 Save your project by clicking on  or by pressing `Ctrl + S`.

The `Call_Transaction_InsertWebSite` is now set.

WHAT COMES NEXT?

The setting up of the sequence is now complete and the sequence is functional.

We will now execute the sequence and analyze results.

While executing the sequence, we will also notice that we cannot know, in this state of the project, whether SQL insertions ended normally or in error.

This is possible by setting *check steps* based on the XML output schema of SQL transactions (see *"Adding Steps to Check Database Transactions"* on page 2-162).

2.6 Executing a Sequence

Both sequences are now set up, and ready to be executed either:

- from the Studio using the **Execute** command;

- by calling a properly formatted URL from a Web browser to access a test platform included in the project.

Executing a sequence requires:

- *Limiting* the number of iterations of each *Iterator* step.

It is recommended to limit the maximum number of iterations when executing the sequence for testing purpose. It reduces the sequence execution time.

- *Opening* all necessary tabs in the Studio:
  - CLI **Connector View** (`sample_documentation_CLI demo5250Connector`) - for the legacy transaction to be executed;
  - CWI **Connector View** (`sample_documentation_CWI GoogleConnector`) - for the Web transaction to be executed,
  - CMS **Sequences Views** (`sample_doc_CMS GetXmlData` and `sample_doc_CMS InsertDataInBase`) - for the sequence to be executed
  - CMS **Connector View** (`sample_doc_CMS DatabaseConnector`) - for SQL transactions to be executed.

*Procedures for executing a sequence are similar whatever the sequence (first or second). In this section, procedures are based on the **second** sequence.*

You will find more information in the following sections:

- Preparing the Sequence and Studio
- Executing the Sequence from the Studio
- Executing the Sequence from the Test Platform
- Executing the Sequence

2.6.1 Preparing the Sequence and Studio

This section describes how to limit the number of iterations for testing purpose and which views to open before executing the sequence.

To limit the number of iterations (Iterator steps)

For each *Iterator* step in your sequence:

1 In the **Projects View**, select the required *Iterator* step (for example `IteratorOnEachRow`) with a left-click.

The **Properties View** is automatically updated:

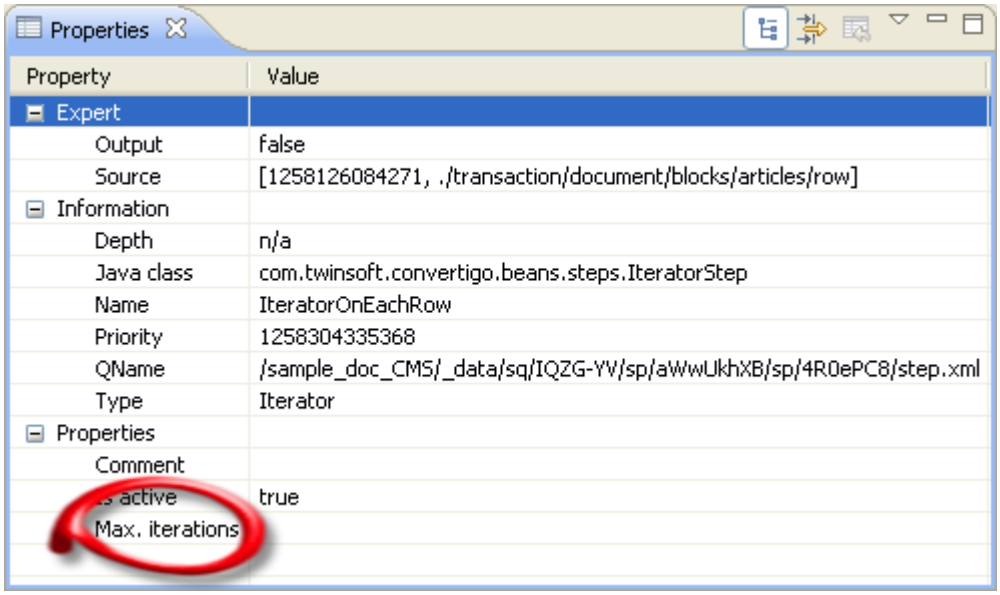


Figure 2 - 229: Iterator step properties

2 Click in the **Value** column of the **Max. iterations** property.

The value is highlighted.

3 Enter a value (3 for example).

4 Press `Enter`.

The property is updated and the step appears bolded in the **Projects View**.

5 Save your project by clicking on  or by pressing `Ctrl + S`.

Remember to reset this property to zero before executing the sequence live.

To open all required tabs prior to executing the sequence

If the CLI **Connector View** is not open:

1 In the **Projects View**, expand the CLI project (for example `sample_documentation_CLI`) **Connectors** folder.

2 Double-click on the CLI connector (for example `demo5250Connector`) to open the CLI **Connector View**:

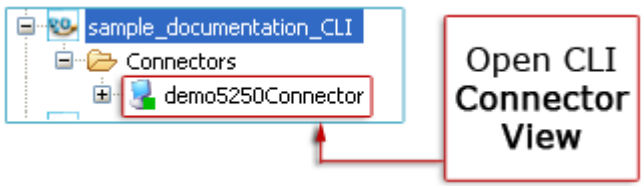


Figure 2 - 230: Opening CLI Connector View

The Connector opens in a new tab (for example `sample_documentation_CLI demo5250Connector`).

3 In the **Projects View**, expand the CLI connector (for example `demo5250Connector`)

Traces folder.

4 Launch the required trace by double-clicking on it:

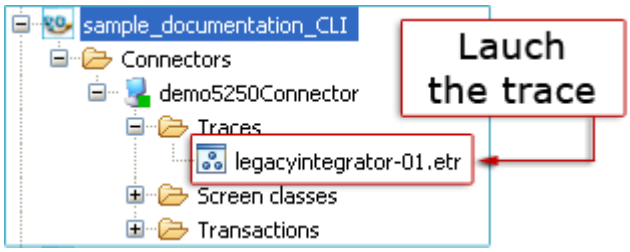


Figure 2 - 231: Launching the CLI projet trace

The trace is automatically launched:

Figure 2 - 232: CLI trace first screen

If the CWI **Connector View** is not open:

5 In the **Projects View**, expand the CWI project (for example `sample_documentation_CWI`) **Connectors** folder.

6 Double-click on the CWI connector (for example `GoogleConnector`) to open the CWI **Connector View**:

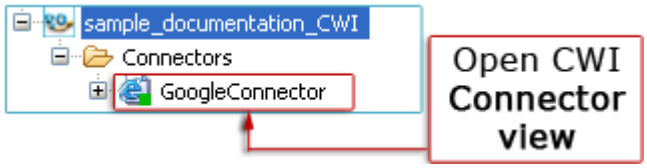


Figure 2 - 233: Opening CWI Connector View

The Connector opens in a new tab (for example `sample_documentation_CWI GoogleConnector`).

Convertigo automatically connects to the HTML page defined for the connector (for example `www.google.com`), which is diplayed in the **Web Browser**:

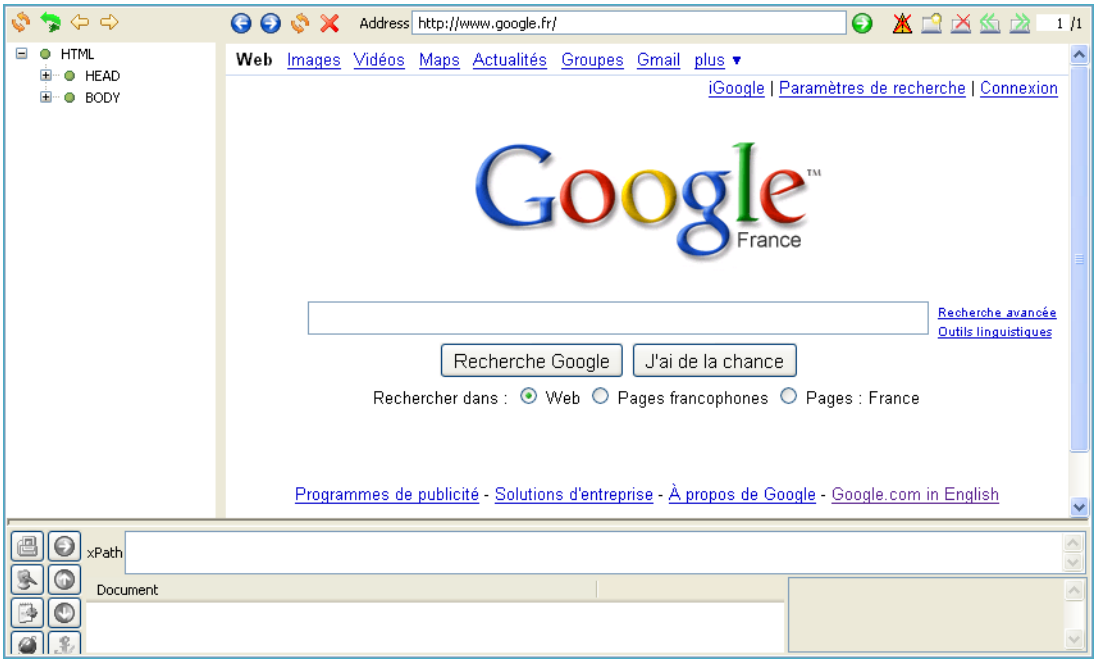


Figure 2 - 234: CWI project Connector View (DOM Tree, Web Browser, XPathEvaluator)

If the CMS SQL **Connector View** is not open:

7 In the **Projects View**, double-click on the CMS SQL connector (for example `DatabaseConnector`) to open the CMS **Connector View** (see *"SQL Connector View"* on page 1-18):

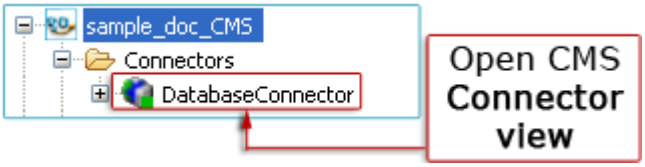


Figure 2 - 235: Opening CMS Connector View

The Connector opens in a new tab (for example `sample_doc_CMS DatabaseConnector`):

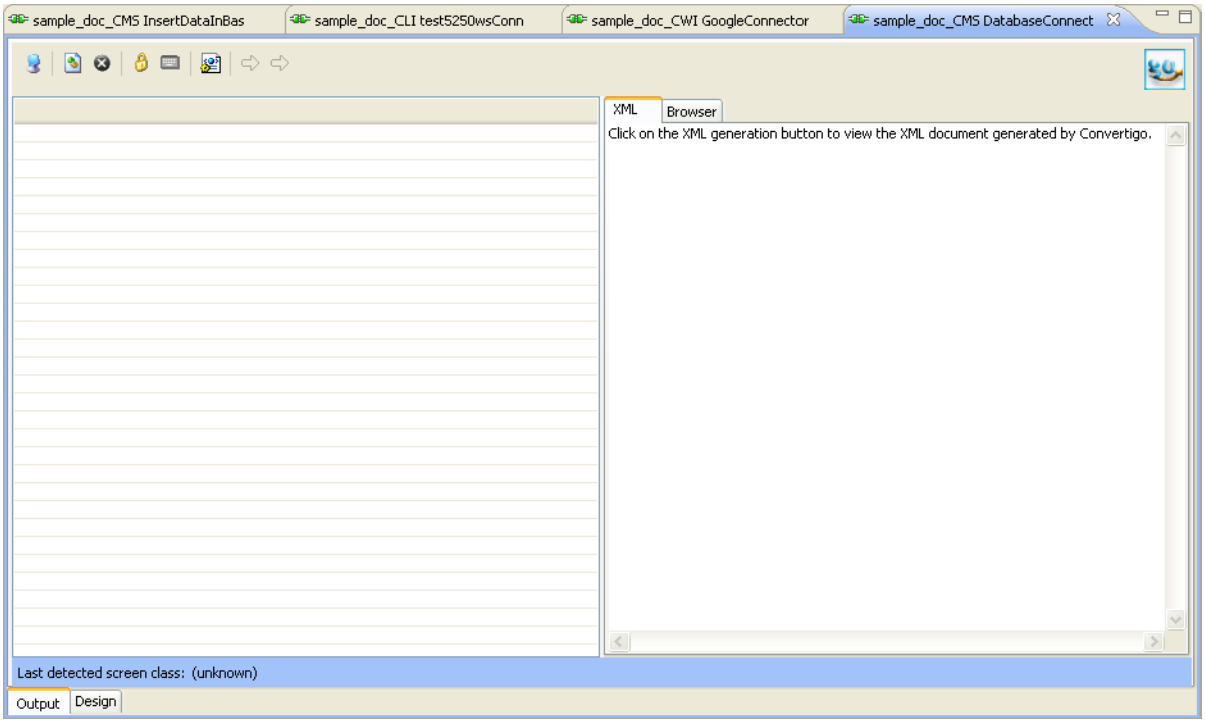


Figure 2 - 236: CMS project Connector View

8 In the **Projects View**, double-click on the CMS sequence (for example `InsertDataInBase`) to open the **Sequence View** (see *"Sequence View"* on page 1-19).

The `sample_doc_CMS InsertDataInBase` tab opens in the **Sequence View**.

All required tabs are now open.

You can execute the sequence:

- from the Studio - see *"Executing the Sequence from the Studio"* on page 2-143,
- from the test platform - see *"Executing the Sequence from the Test Platform"* on page 2-144.

2.6.2 Executing the Sequence from the Studio

This section describes how to execute the sequence from the Studio using the **Execute** command.

To execute the sequence from the Studio, the input variable defined for the sequence (`article_code`, see *"To create a new sequence variable"* on page 2-15) must be given a default value.

To execute a sequence from the Studio

1 In the **Projects View**, left-click on the sequence variable (`article_code`):

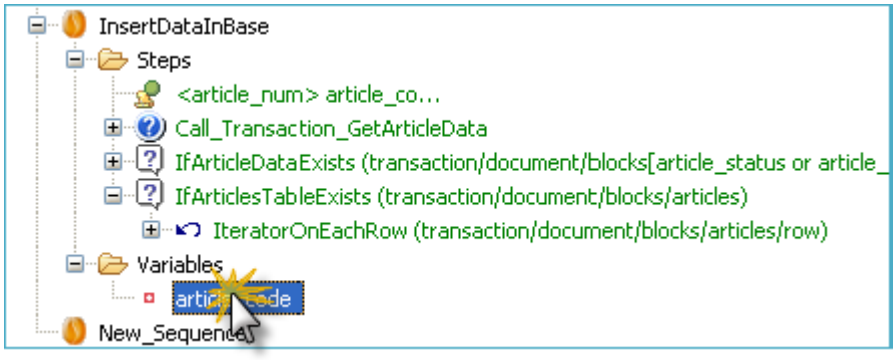


Figure 2 - 237: Selecting the sequence variable

The **Properties View** is automatically updated.

2 In the **Properties View**, click in the **Value** column of the **Default Value** property.

3 Enter a default value (for example `123456789`).

4 Press `Enter`.

The sequence variable is now set as a fixed variable with a default value.

5 Save your project by clicking on  or by pressing `Ctrl + S`.

6 In the **Projects View,** right-click on the sequence (for example `InsertDataInBase`).

A contextual menu appears:

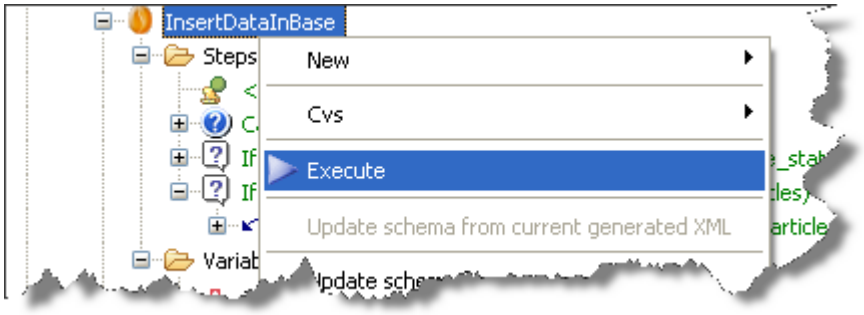


Figure 2 - 238: Sequence contextual menu

7 Select **Execute**.

The sequence starts.

*For more information on the sequence execution and its results in the different tabs of the Studio's **Connector and Sequence Views**, see "Executing the Sequence" on page 2-147).*

2.6.3 Executing the Sequence from the Test Platform

This section describes how to execute the sequence from the test platform using any Web Browser.

Unlike the previous procedure, executing the sequence from the test platform does not require setting the sequence input variable prior to the execution, since variables are set directly in the platform.

To execute a sequence using the test platform

1 Launch a standard Web browser.

2 In the URL address field, type in the following URL:

```
http://<HostName>:<ConvertigoPortNumber>/convertigo/projects/
<ProjectName>
```

For example, in our project:

- *HostName* = `localhost`;
- *ConvertigoPortNumber* = `18080`;
- *ProjectName* = `sample_doc_CMS`;

The test platform opens:

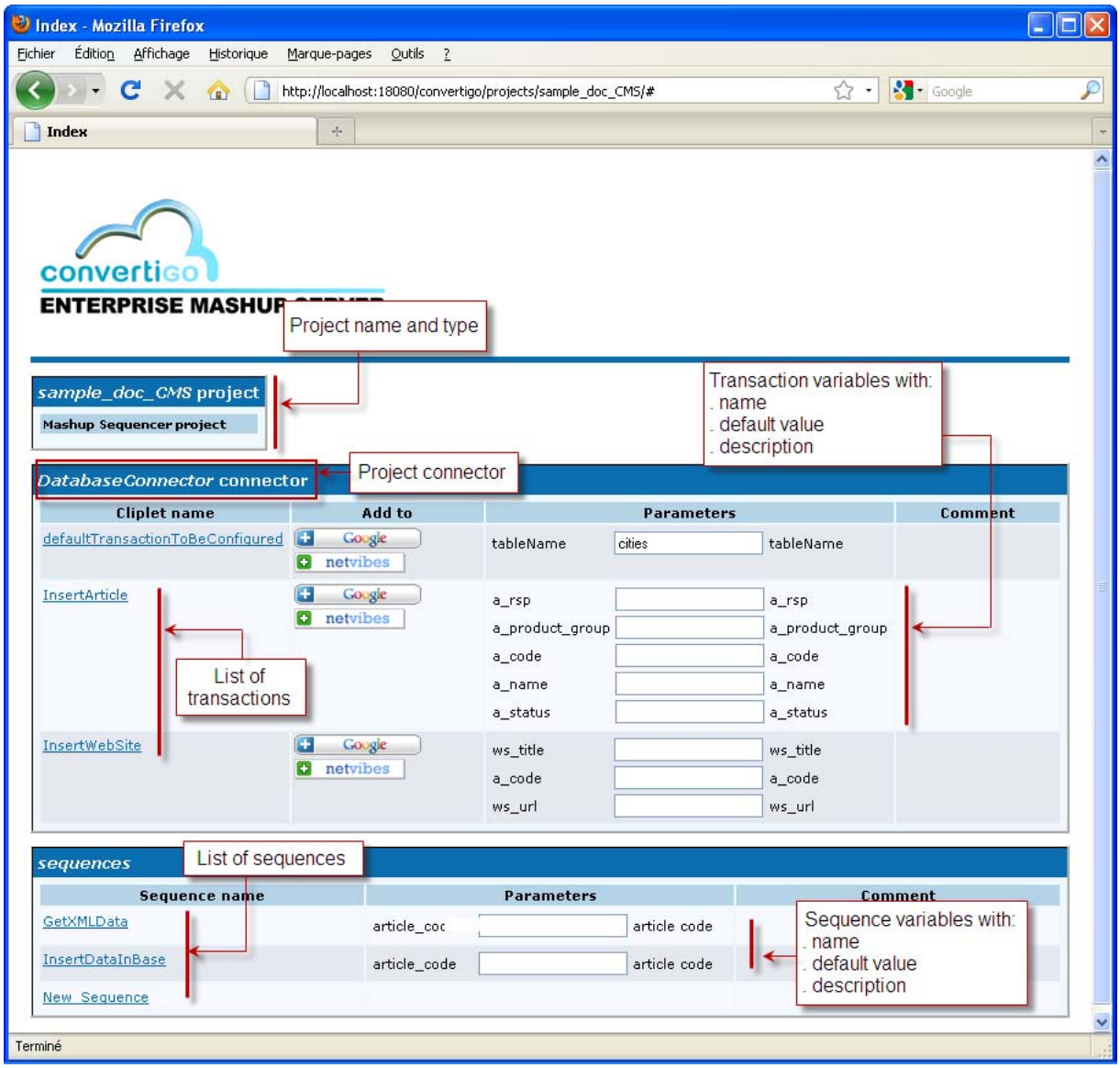


Figure 2 - 239: Test platform

It displays:

- in a first table:
  - the **project name** (in our example, `sample_doc_CMS`);
  - the **project type** (for example a Mashup Sequencing project);
- in a second table:
  - the ***connector*** defined for the project (in our example, `DatabaseConnector`);
  - all ***transactions*** defined for the connector (in our example, the default transaction, `InsertArticle` and `InsertWebSite`) and their *variables*.

If several connectors were defined in the project, the test platform displays as many connector tables as defined connectors.

- in a third table, the ***sequences*** defined for the project (in our example, the default sequence, the `GetXmlData` sequence and the `InsertDataInBase` sequence) and their *variables*.

3 In the **sequences** table, enter an article code in the **article_code** field (**Parameters** column of the `InsertDataInBase` sequence): for example `123456789`.

4 Click on the `InsertDataInBase` link.

The sequence starts. You can switch to the Studio to see steps being processed and results appear in opened tabs.

*For more information on the sequence execution and its results in the different tabs of the Studio **Connector and Sequence Views**, see "Executing the Sequence" on page 2-147).*

Once the transaction has been completed, the sequence XML output is displayed in an **Execution result** window in the test platform:

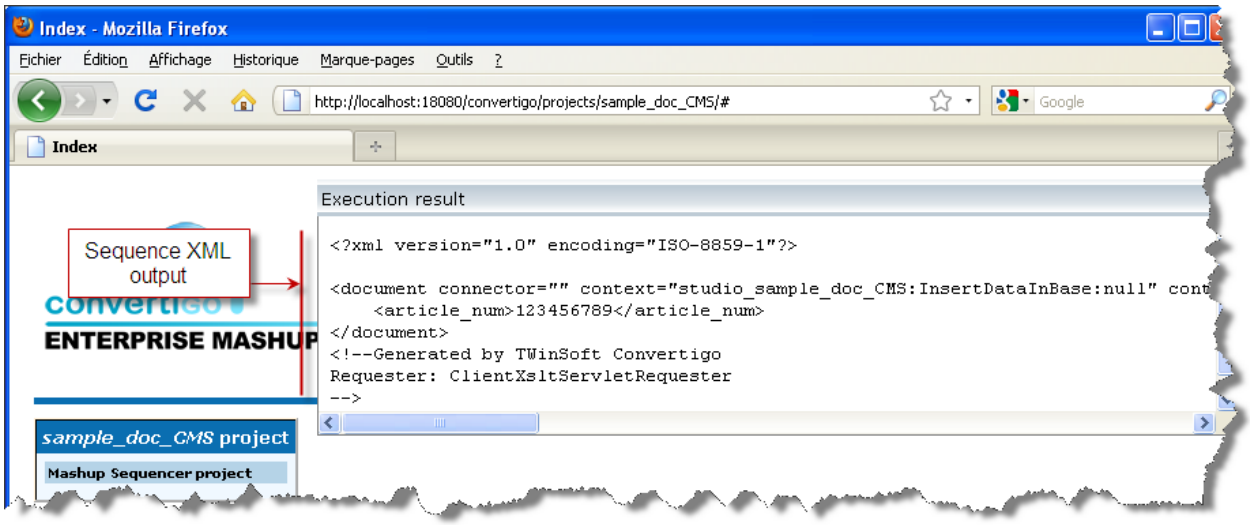


Figure 2 - 240: Sequence XML output

This XML output is similar to the XML output displayed in the Studio's **Sequence View XML**

tab after the sequence has been executed from the Studio using the **Execute** command (see Figure 2 - 250).

2.6.4 Executing the Sequence

This section describes the information displayed in the opened views prior to executing the sequence.

Objects processed in real time (either steps in CMS and statements in CWI) appear bolded one after another in the **Projects View**:

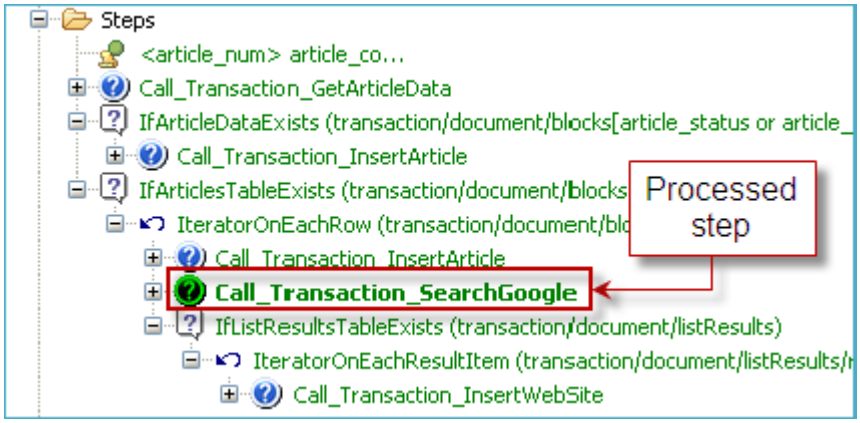


Figure 2 - 241: Step currently processed

*If steps do not appear bolded when executing the sequence, click on **Window > Preferences** in the Studio toolbar, expand the tree structure to **Convertigo > Studio** then check the **Highlight detected objects in tree** checkbox.*

Whatever the sequence, once the `Call_Transaction_GetArticleData` step has been executed, the XML output generated by the CLI transaction (`GetArticleData`) is displayed in the CLI **Connector View**:

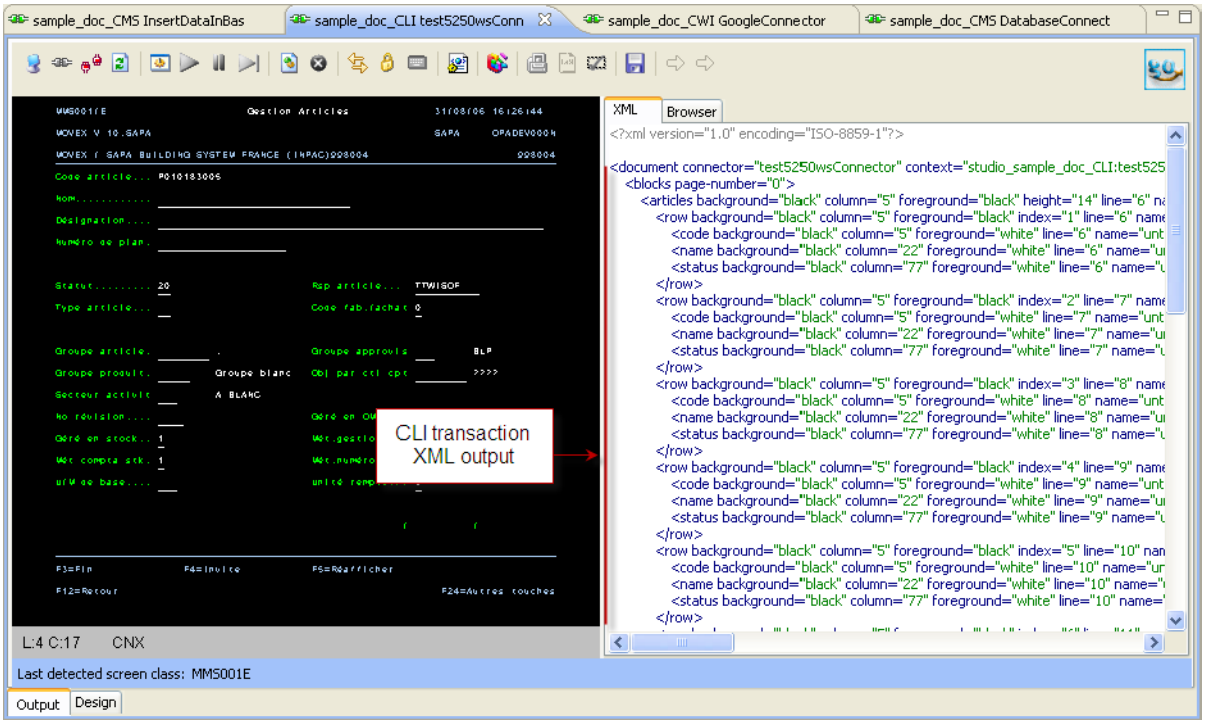


Figure 2 - 242: CLI transaction XML output

While the sequence is being executed, the **Web Browser** (CWI **Connector View**) displays Google search operations:

Figure 2 - 243: CWI searchGoogleWithLimit transaction

Search keywords appearing in the Google search field come from name nodes generated by the CLI `GetArticleData` transaction (see Figure 2 - 242).

Then, depending on whether the sequence was executed from the studio (see *"Executing the Sequence from the Studio"* on page 2-143) or from the test platform (see *"Executing the Sequence from the Studio"* on page 2-143), the XML output is displayed either:

- in the **XML** tab of the **Sequence View** or
- in an **Execution result** window automatically opening in the external Web Browser from

which the sequence has been executed.

2.7 Analyzing the First Sequence XML Output

This section describes the XML output generated by the first sequence:

- Introduction
- XML Output Analyzis

2.7.1 Introduction

After the sequence has been executed, the sequence XML output is displayed either:

- in the **XML** tab of the **Sequence View** (sequence executed from the Studio):

Figure 2 - 244: First sequence XML output in XML tab of Sequence View

- in the **Execution result** window in the Web Browser (sequence executed from the test platform):

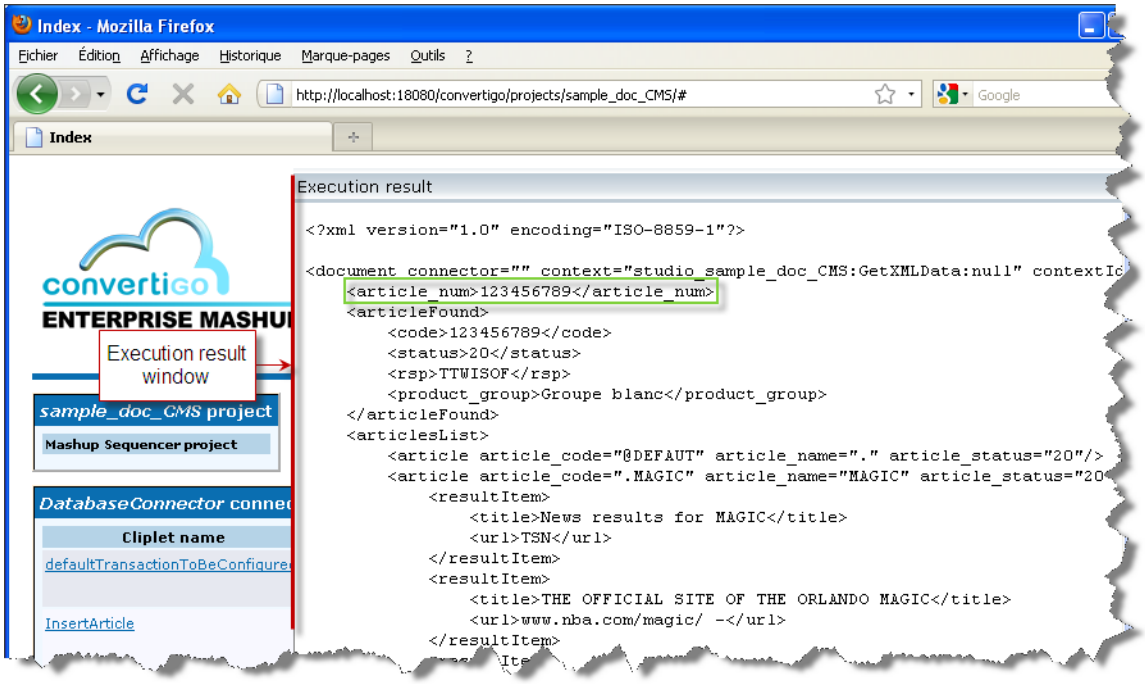


Figure 2 - 245: Second sequence XML output in Execution result window of Web Browser

The first line displayed after the header is the portion of sequence XML output generated by the first step (*jElement,* see Figure 2 - 245).

As mentioned when describing the **Step Source** wizard, some steps generate XML, others do not. Whether a step generates or not XML depends on the value of its **Output** property. In the case of the *jElement* step, it is set to `true` by default.

Knowing the article code through this isolated first line is unwanted since the article code is already included in the following `<articleFound>` XML element.

To remove this line, we will set the *jElement* step's **Output** property to `false` and reexecute the sequence.

To set a steps' Output property to `false`

1 In the **Projects View**, select the step with a left-click (for example the *jElement* step).

The **Properties View** is automatically updated:

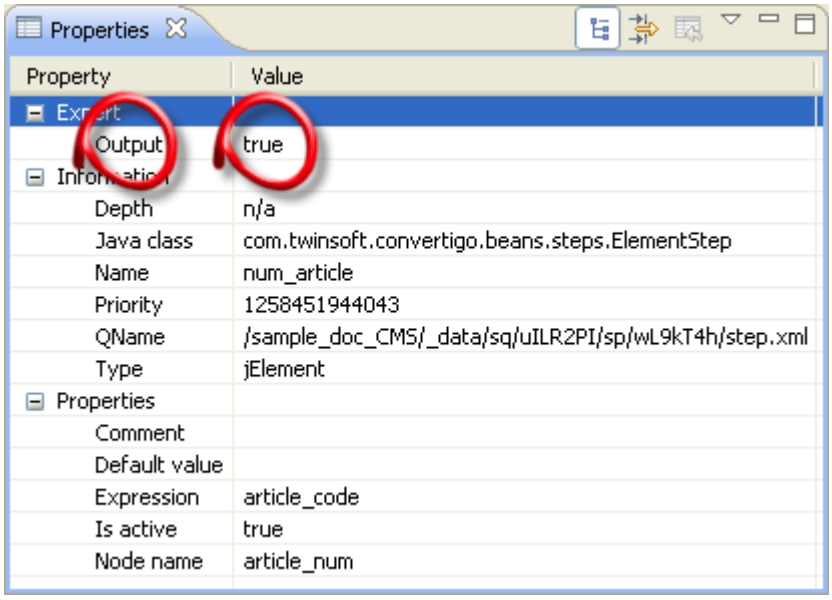


Figure 2 - 246: jElement step properties

2 Click in the **Value** column of the **Output** property.

The current value is highlighted and a symbol appears.

3 Click on .

A drop-down menu displays available **Output** values.

4 Select `false`.

5 Press `Enter`.

The property is updated and the step appears bolded in the **Projects View**.

6 Save your project by clicking on or by pressing `Ctrl + S`.

7 After the **Output** property has been set to `false`, execute the sequence again (for example from the Studio, see *"To execute a sequence from the Studio"* on page 2-143).

*Prior to executing the sequence again, remember to reset the CLI connector by clicking on the **Disconnect the connector icon** , then on the **Connect the connector** icon*

At the end of the sequence execution, the **Sequence View** displays the complete sequence XML output (if executed from the Studio):

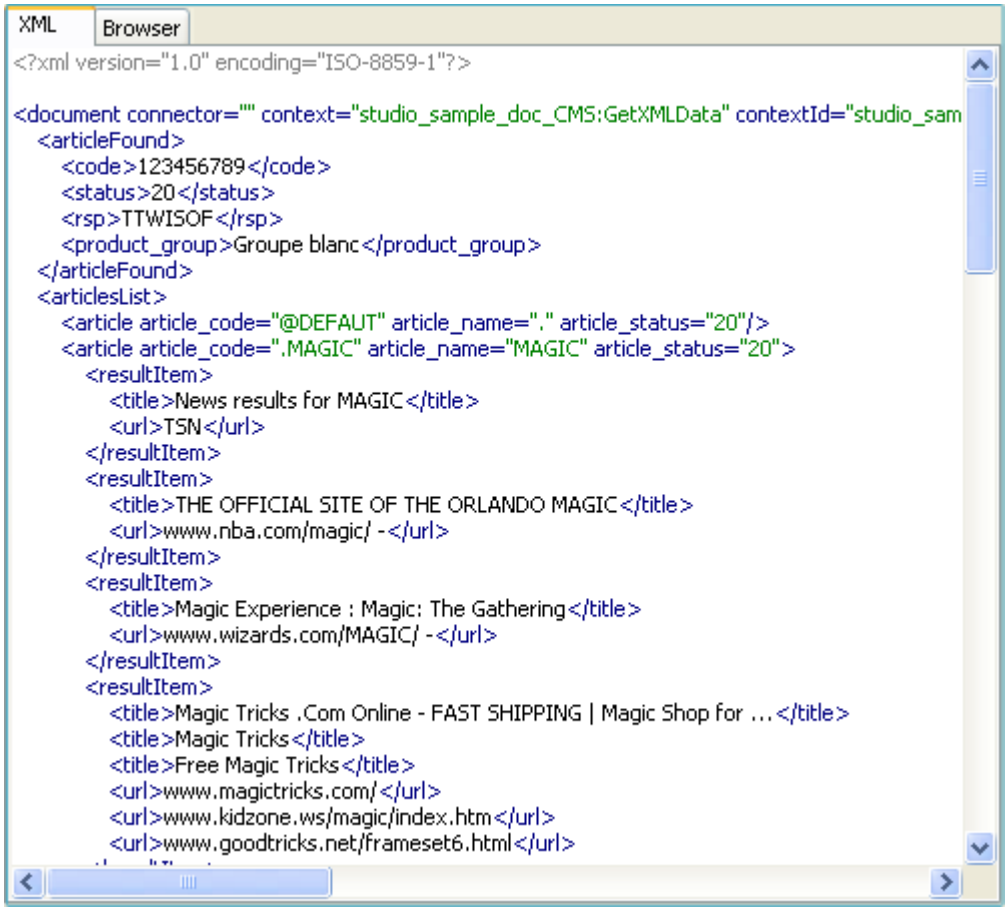


Figure 2 - 247: First Sequence XML output in Sequence View

We will now analyze the sequence XML output.

2.7.2 XML Output Analyzis

This sub-section analyzes in detail the XML output generated by the first sequence.

In the context of the first sequence of this *Quick Guide* project, the sequence XML output includes the XML output generated by:

- the first *Complex* step generating a complex `<articleFound>` XML element together with four simple XML elements (`code_article`, `statut_article`, `rsp_article` and `product_group`) if an article is found in the XML output returned by the `GetArticleData` transaction:

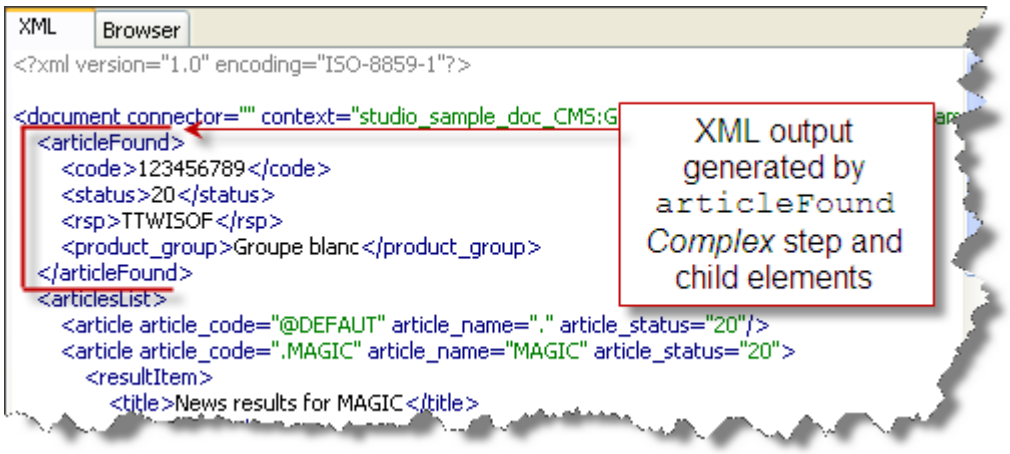


Figure 2 - 248: Portion of XML generated by first Complex and Element steps

An `<articleFound>` element has been generated here because an article was found in the XML output returned by the `GetArticleData` transaction. If not, an `<articleNotFound>` would have been generated instead.

- the second *Complex* step generating a complex `<article>` XML element together with three XML attributes (`article_code`, `article_name`, `article_status`) for each row of the `articles` XML table returned by the `GetArticleData` transaction:

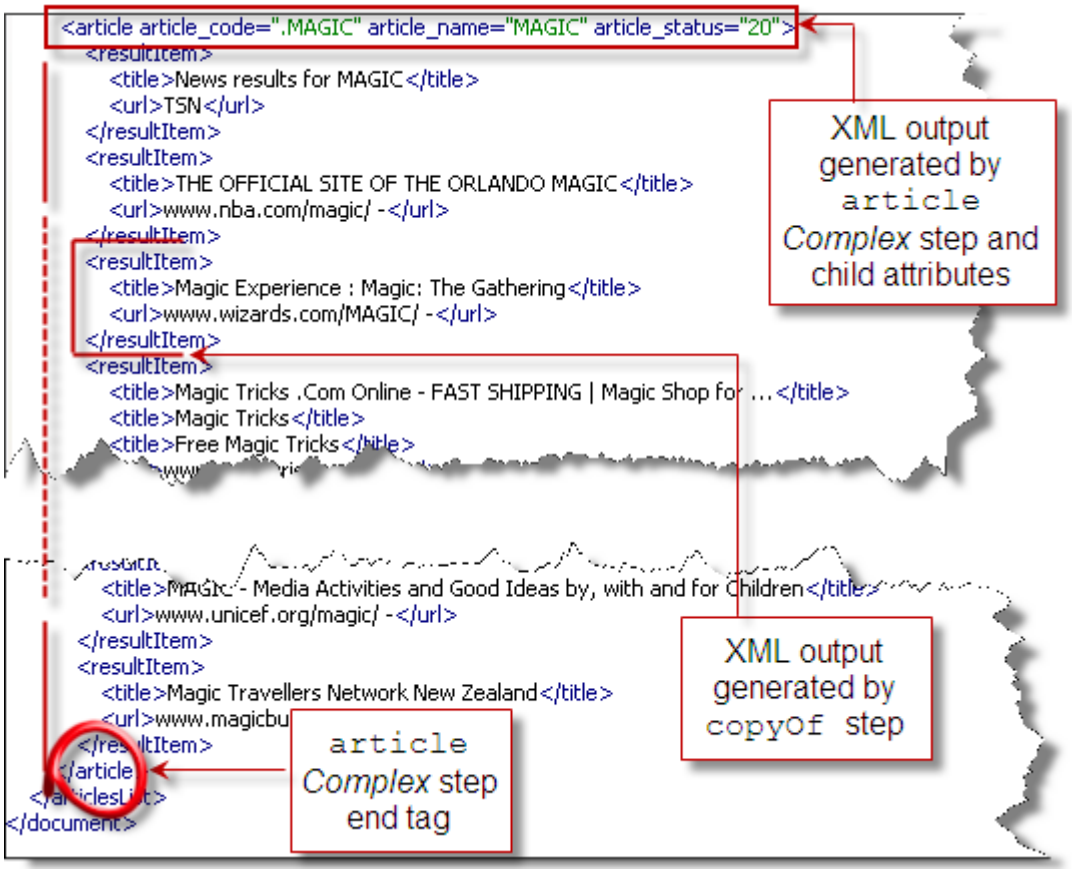


Figure 2 - 249: Portion of XML generated by second Complex and Element steps

In addition, `resultItem` nodes and their child nodes are copied from the `searchGoogleWithLimit` transaction XML output (when found) into the structure of `<article>` elements (see Figure 2 - 249) through the execution of the `copyOf` step.

The sequence XML output contains as many `<article>` complex elements as rows found in the `articles` XML table.

2.8 Analyzing the Second Sequence XML Output

This section describes the XML output generated by the second sequence and proposes methods for generating extra or more useful information in the sequence output:

- Second Sequence XML Output
- Adding Steps to Check Database Transactions

2.8.1 Second Sequence XML Output

INTRODUCTION

After the sequence has been executed, the sequence XML output is displayed either:

- in the **XML** tab of the **Sequence View** (sequence executed from the Studio):

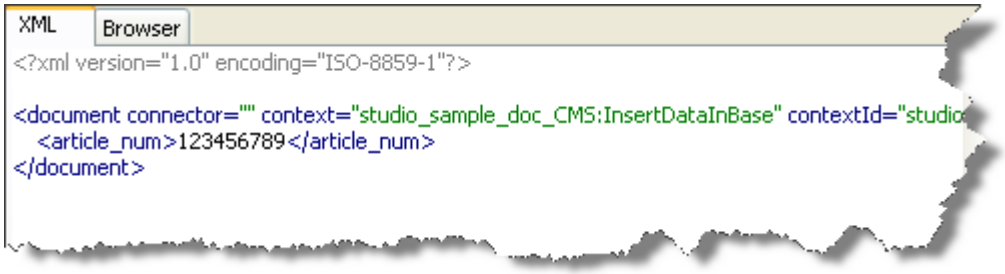


Figure 2 - 250: Second sequence XML output in XML tab of Sequence View

- in the **Execution result** of the Web Browser (sequence executed from the test platform):

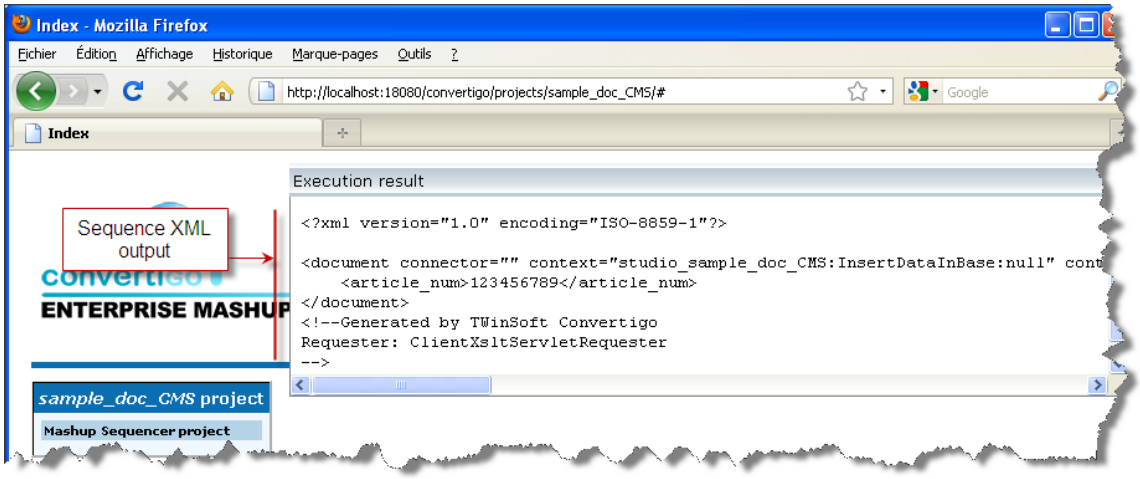


Figure 2 - 251: Second sequence XML output in Execution result window of Web Browser

Only the XML generated by the first step appears in the sequence XML output.

Indeed, as mentioned when describing the **Step Source** wizard, some steps generate XML, others do not. The only step defined in the second sequence as generating XML is the first *jElement* step (see *"To create and set a jElement step serving as source for a step variable"* on page 2-24).

Whether a step generates or not XML depends on the value of its **Output** property. In the case of the *jElement* step, it is set to `true` by default:

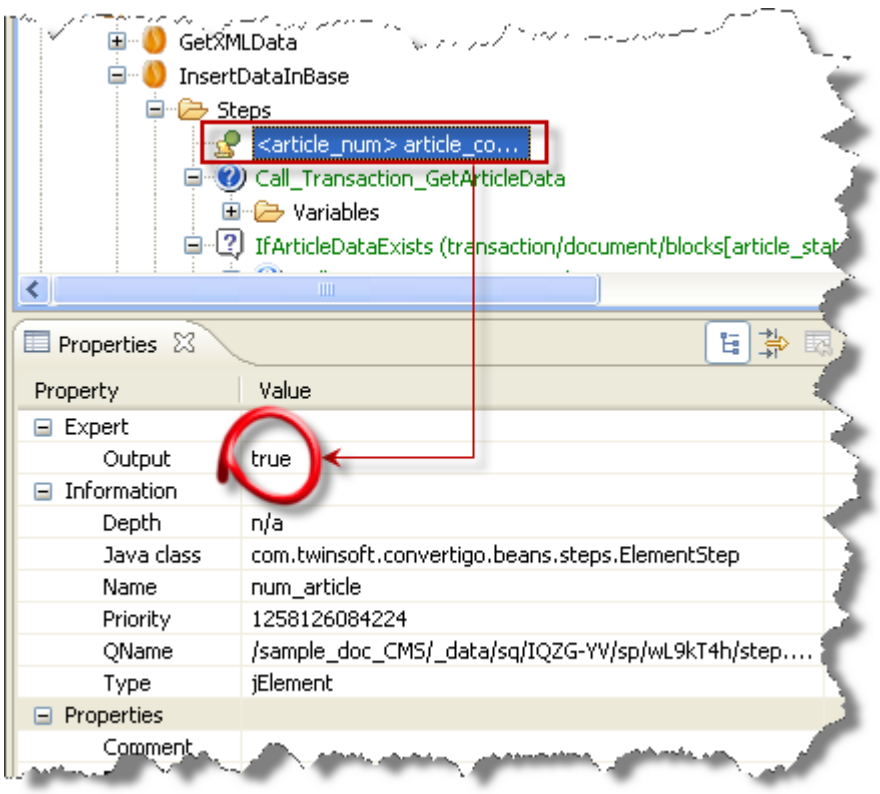


Figure 2 - 252: Output property of selected step set to true

In the second sequence, steps other than the *jElement* step do not generate XML since their **Output** property is set to `false` by default. As a conclusion, the only information we can rely on so far about the sequence execution is the information generated by the first step.

The question arising now is: *"What can possibly be done in terms of project settings for the sequence XML output to be more meaningful or useful?"*

For example, is it possible to get more information about transaction XML outputs, or to know how many database insertions ended successfully?

To answer these questions, several actions can be undertaken:

- to get a more complete sequence XML output, *set* all *Call Transaction* steps' **Output** property to `true`.
- to know more about database insertions, *create* and *set* check steps (of *IfExistThenElse* type) based on the XML schema of SQL transactions.

These are two distinct solutions for generating extra XML information about the sequence execution. You can choose either method, but do not apply both at the same time.

We will first set the **Output** property of all *Call Transaction* steps to `true`.

The creation and setting of additional check steps is done in a further section (see *"Adding Steps to Check Database Transactions"* on page 2-162).

To set the Call Transaction steps' `Output` property to `true`

For each *Call Transaction* step of the sequence:

1 In the **Projects View**, select the required *Call Transaction* step with a left-click.

The **Properties View** is automatically updated:

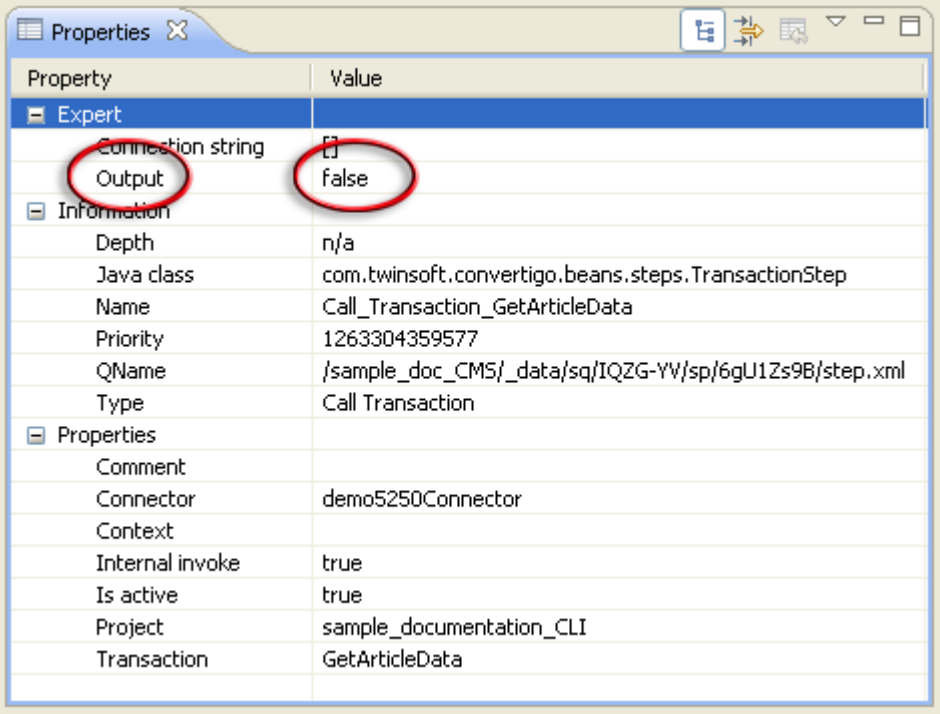


Figure 2 - 253: Call Transaction step properties

2 Click in the **Value** column of the **Output** property.

The current value is highlighted and a symbol appears.

3 Click on .

A drop-down menu displays available **Output** values.

4 Select `true`.

5 Press `Enter`.

The property is updated and the step appears bolded in the **Projects View**.

6 Save your project by clicking on or by pressing `Ctrl + S`.

7 After the **Output** property of all *Call Transaction* steps has been set to `true`, execute the sequence again (for example from the Studio, see *"To execute a sequence from the Studio"* on page 2-143).

*Prior to executing the sequence again, remember to reset the CLI connector by clicking on the **Disconnect the connector icon** , then on the **Connect the connector** icon*

At the end of the sequence execution, the **Sequence View** displays the complete sequence

XML output (if executed from the Studio):

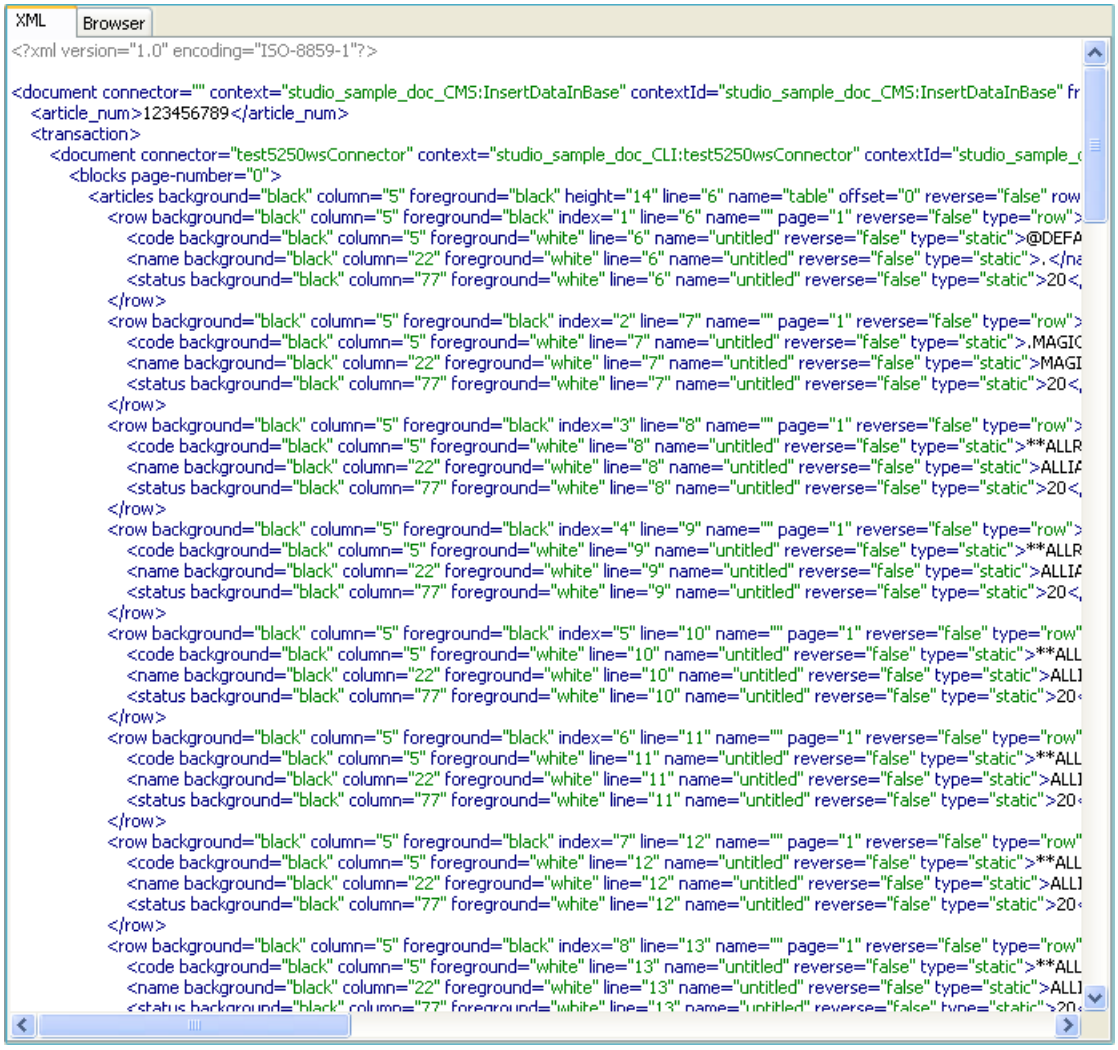


Figure 2 - 254: Second Sequence XML output in Sequence View

We will now analyze the second sequence XML output.

XML Output Analyzis

This sub-section analyzes in detail the XML output generated by the second sequence when all *Call Transaction* steps are defined as generating XML code (**Output** property set to `true`).

The XML output includes as many `<transaction>` tags as *Call Transaction* steps generating XML through the setting of their **Output** property to `true`. A `<transaction>` tag (ended by a `</transaction>` end tag) contains the whole XML output generated by a *Call Transaction* step, i.e. the whole XML output of the called transaction.

*Steps defined as generating XML by definition (**Steps which generate XML**, see Figure 2 - 26) do not generate `<transaction>` tags. Only Call Transaction steps, not generating XML by definition, do when their **Output** property is set to `true`.*

In the context of the second sequence of this *Quick Guide* project, the sequence XML output includes the XML output generated by:

- the *jElement* step:

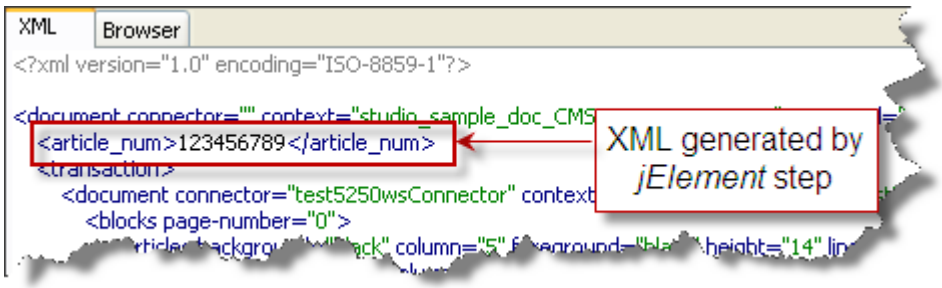


Figure 2 - 255: Portion of XML generated by jElement step

By definition, a *jElement* step generates an XML element, so the XML output generated by this step is not nested in a `<transaction>` tag, as mentioned in the previous note.

- the `Call_Transaction_GetArticleData` step:

Figure 2 - 256: Beginning of XML portion generated by Call_Transaction_GetArticleData step

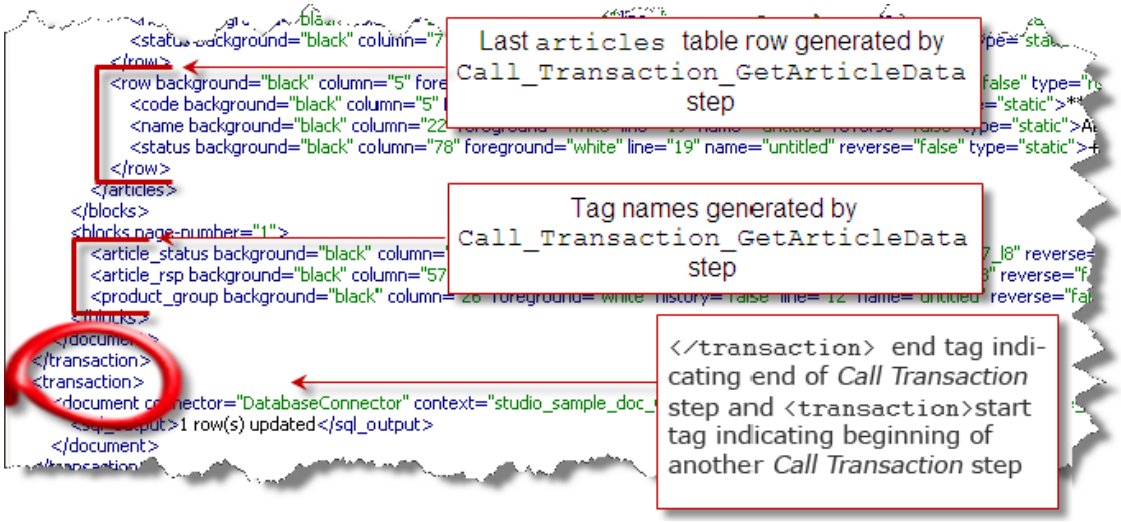


Figure 2 - 257: End of XML portion generated by Call_Transaction_GetArticleData step

The `Call_Transaction_GetArticleData <transaction>` tag (ended by a `</transaction>` end tag, see Figure 2 - 257) contains the whole XML output generated by the `GetArticleData` transaction.

For more information on the XML output generated by the `GetArticleData` transaction, refer to the "Starting with Convertigo Legacy Integrator" Quick Guide.

- the two `Call_Transaction_InsertArticle` steps:

Figure 2 - 258: XML portion generated by Call_Transaction_InsertArticle step

The first `Call_Transaction_InsertArticle` step is executed following the `IfArticleDataExist` check and the second `Call_Transaction_InsertArticle` step following the `IfArticlesTableExists` check, when iterating on the first row of the `articles` XML table.

Two SQL transactions were defined in the sequence (`InsertArticle` and `InsertWebSite`). To check which SQL transaction returned the `<sql_output>` tag, scroll to the right of the required `<document>` tag and find the `transaction` name:

Figure 2 - 259: "transaction" attribute showing SQL transaction name

- the first `Call_Transaction_SearchGoogle` step, on the first iterated row of the `articles` XML table:

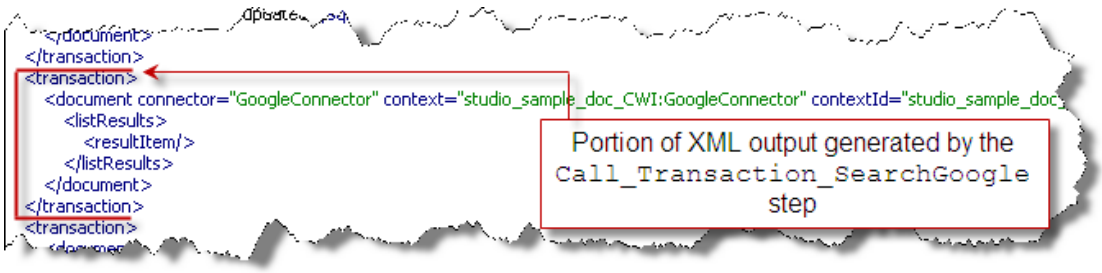


Figure 2 - 260: XML portion generated by Call_Transaction_SearchGoogle step

This portion of XML was generated by the `searchGoogleWithLimit` transaction called by the `Call_Transaction_SearchGoogle` step. The `listResults` is empty because the transaction has not returned any Google result for the first iterated row.

- the first `Call_Transaction_InsertWebSite` step and third `Call_Transaction_InsertArticle` step:

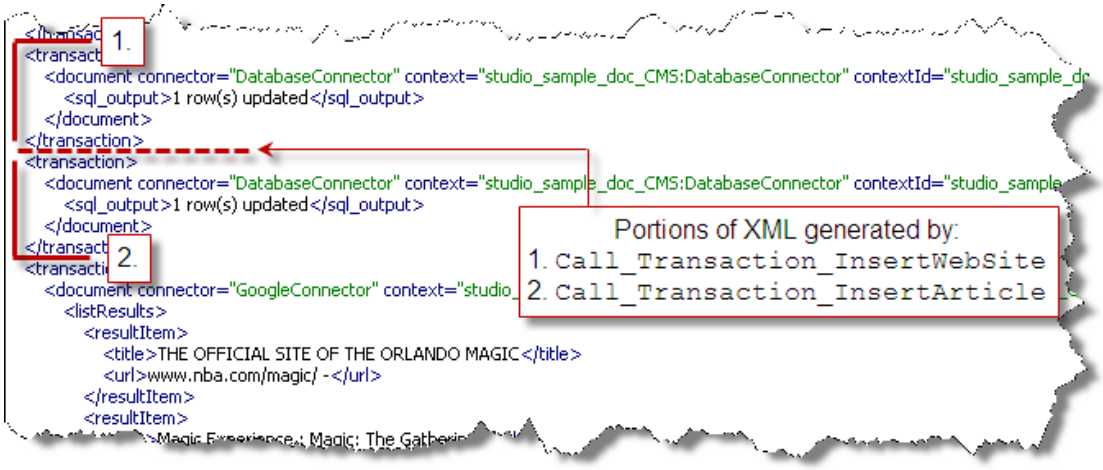


Figure 2 - 261: XML portion generated by second and third Call SQL transaction steps

The XML portion generated by the `Call_Transaction_InsertWebSite` step corresponds to the first iteration on the `listResults` XML table `resultItem` rows, and the XML portion generated by the `Call_Transaction_InsertArticle` step corresponds to the second iteration on the `articles` XML table `row` rows.

Then, the `Call_Transaction_SearchGoogle` is executed a second time:

Figure 2 - 262: Second execution of Call_Transaction_SearchGoogle step (beginning)

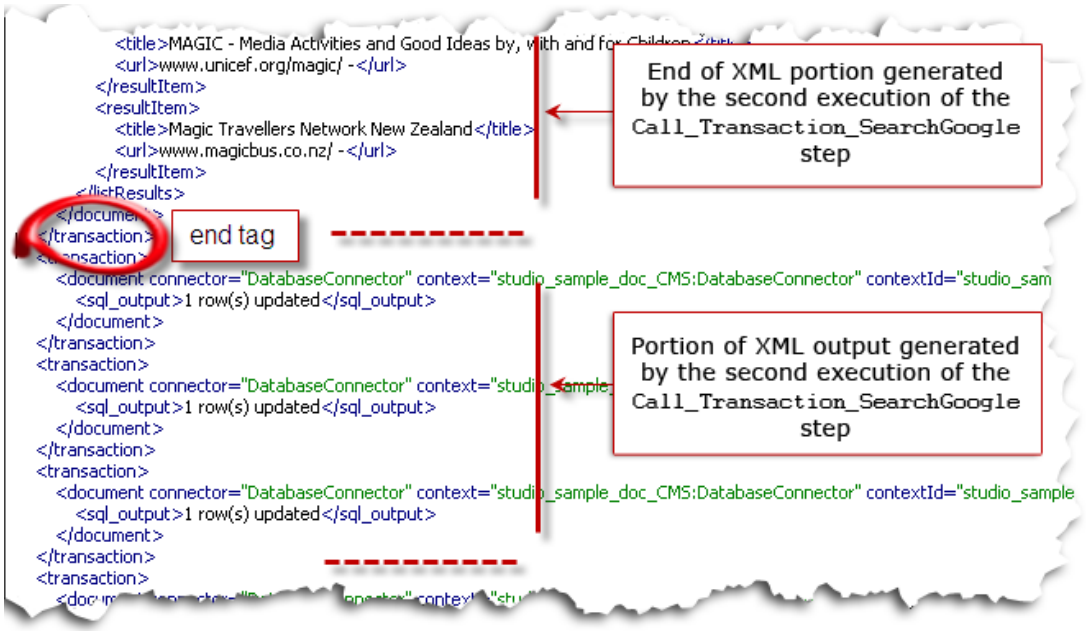


Figure 2 - 263: Second execution of Call_Transaction_SearchGoogle step (...)

After the `Call_Transaction_SearchGoogle` has been executed a second time and returned Google results, the `Call_Transaction_InsertWebSite` is executed three times. This number corresponds to the value of the **Max. iterations** property set for the `IteratorOnEachResultItem` step (see *"To limit the number of iterations (Iterator steps)"* on page 2-139).

WHAT COMES NEXT?

As a conclusion, the XML output is much more detailed now than before, when the only information we had about the sequence was generated by the first *jElement* step.

But for this knind of sequence, the ideal output would be one including only relevant information about SQL transactions and database insertions. To this end, we will reset all **Output** properties to `false` and create check steps after SQL transactions to structure the XML output.

2.8.2 Adding Steps to Check Database Transactions

Adding check steps after SQL transactions will help us resolve the following issue: *"Following an SQL transaction call, is it possible to know if the SQL query associated with the transaction ended normally or not."*

To this end, we will create and set a check step (of *IfExistThenElse* type), the source of which points towards nodes of the XML output generated by the *Call Transaction* step calling the SQL transaction to be checked.

Prior to creating and setting check steps, make sure that you have: tested the SQL transactions to be checked, set them as public methods and updated their schema (see "Testing an SQL Transaction and Updating its XML Schema" on page 2-89).

To create and set an SQL transaction check step

1. Set the **Output** property of all *Call transaction* steps to `false` (to keep in the final sequence XML output only information generated by check steps).
2. In the **Projects View**, right-click on the parent step (for example `IfArticleDataExist`).

   A contextual menu appears.

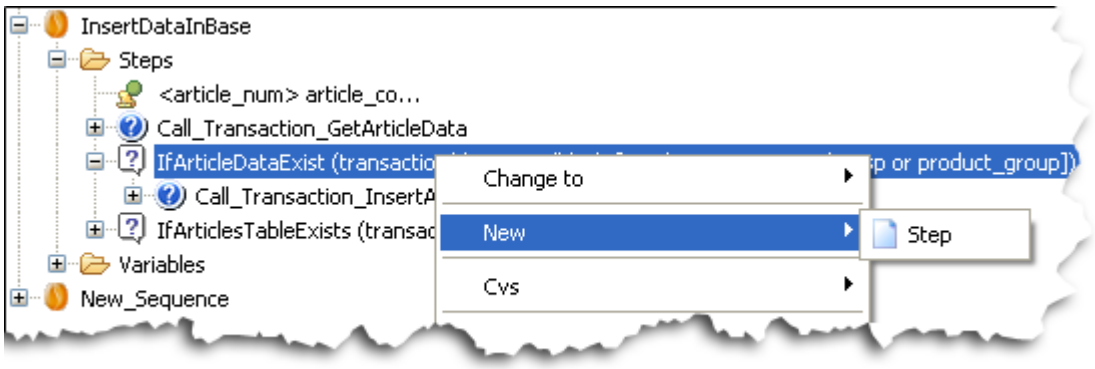


Figure 2 - 264: Parent step contextual menu

3. Select ***New > Step***.

   The **New Step** wizard is automatically launched.
4. In the **Other steps** area, select **IfExistThenElse**:

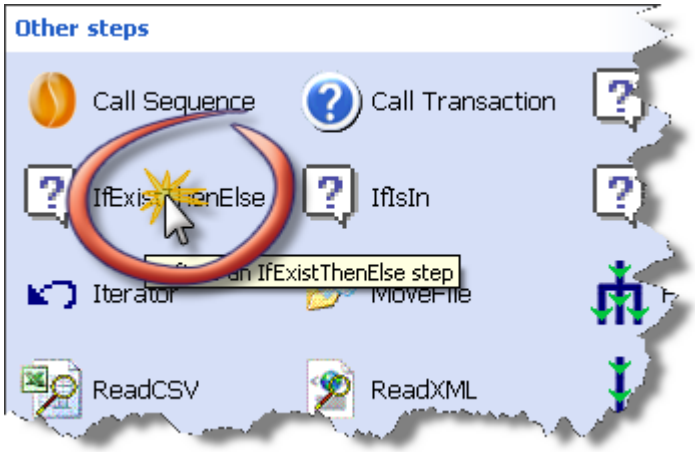


Figure 2 - 265: Selecting an IfExistThenElse step

5 Click on **Next**.

6 In the **Name** field that appears, type in the *IfExist* step name (for example `IfSql_outputExists`):

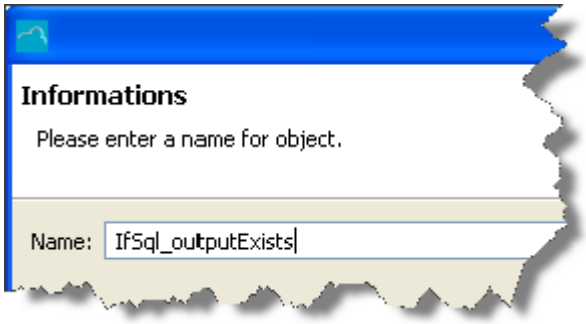


Figure 2 - 266: Entering the IfExistThenElse step name

7 Click on **Finish**.

The new step appears in the **Steps** folder of the project:

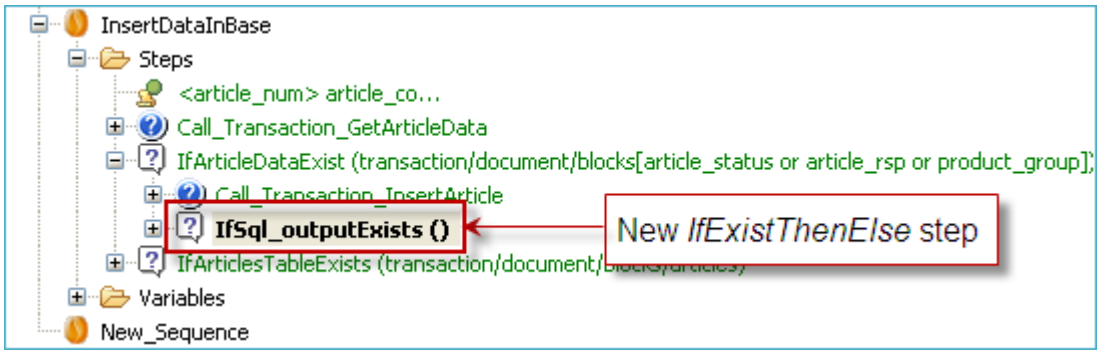


Figure 2 - 267: New IfExistThenElse step in Steps folder

8 Save your project by clicking on or by pressing `Ctrl + S`.

The *IfExistThenElse* step has been created. It is parent to two sub-steps, *Then* and *Else*:

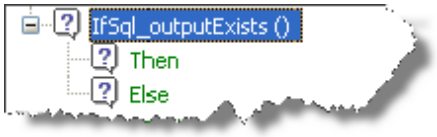


Figure 2 - 268: IfExistThenElse sub-steps

The *IfExistThenElse* step source must be set so as to point towards the `sql_output` XML node returned by the `InsertArticle` SQL transaction (called by the preceding step).

9 Select the *IfExistThenElse* step (for example `IfSql_outputExists`) with a left-click.

The **Properties View** is automatically updated.

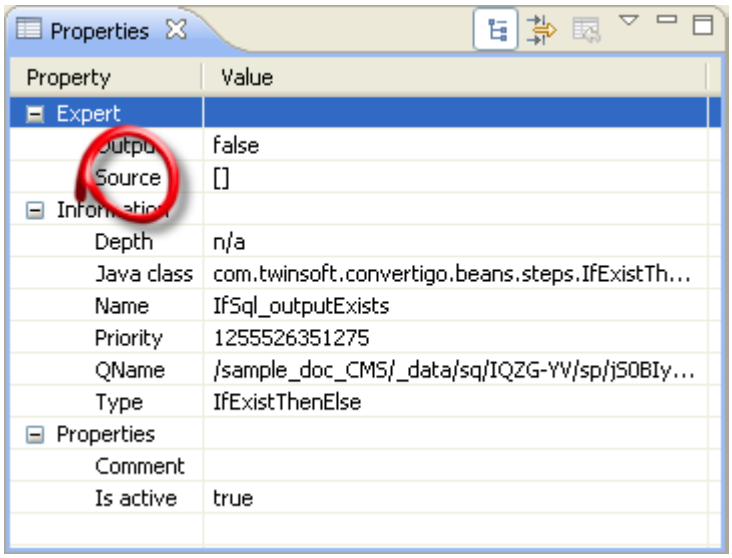


Figure 2 - 269: IfExistThenElse step properties

10 Click in the **Value** column of the **Source** property.

A button appears.

11 Click on the button to launch the **Step Source** wizard.

The **Step Source** wizard is automatically launched.

12 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

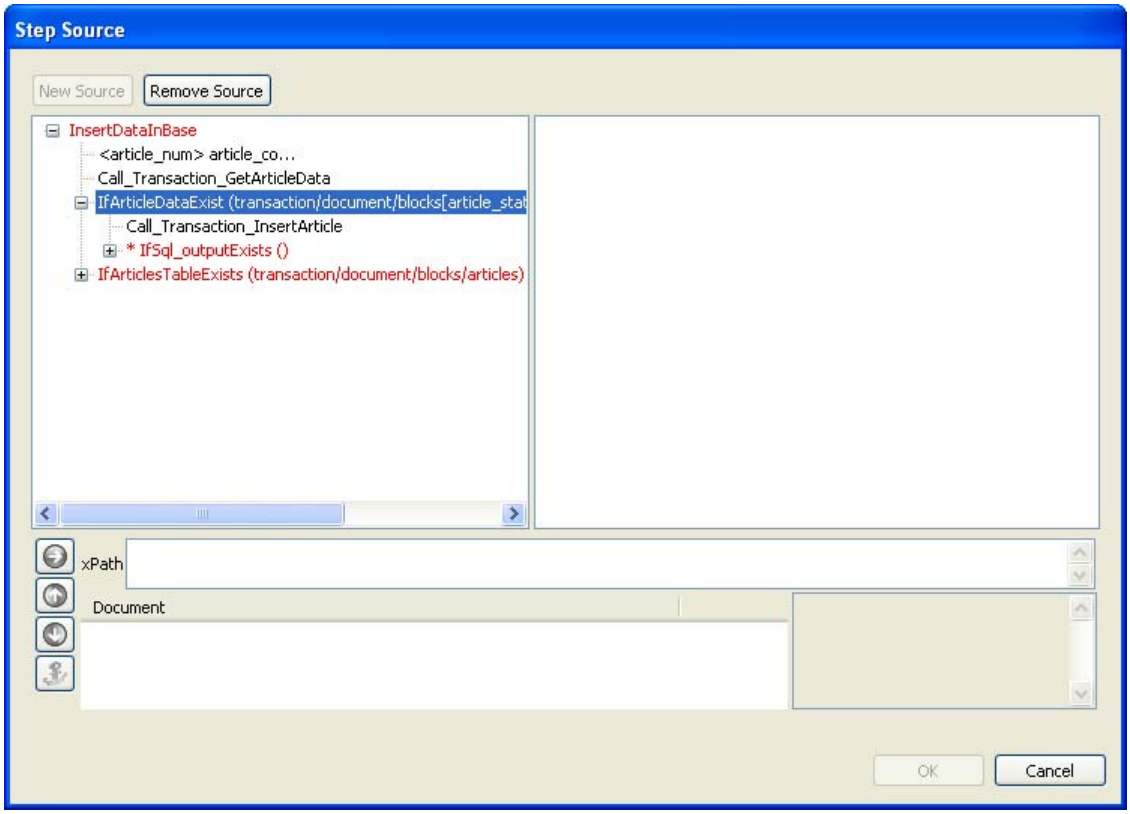


Figure 2 - 270: Setting of IfSql_outputExists step source

The *IfExistThenElse* check must be performed on the XML schema of the output generated when calling the `InsertArticle` transaction. The step to be selected is therefore `Call_Transaction_InsertArticle`.

13 In the **Steps Tree Structure**, select `Call_Transaction_InsertArticle` with a left-click.

The selected step's **XML Output Schema** is displayed:

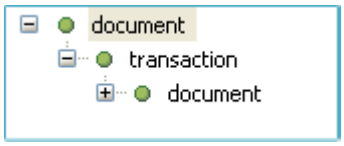


Figure 2 - 271: XML Schema of Call_transaction_InsertArticle step

14 Expand the `document` node:

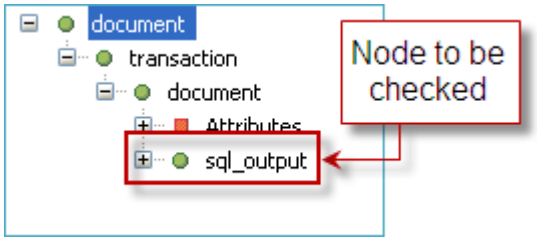


Figure 2 - 272: Node to be checked in the Call_transaction_InsertArticle XML output schema

15 Select the node to be checked (`sql_output`).

The XPath expression to this node is automatically generated in the **XPath Evaluator**:

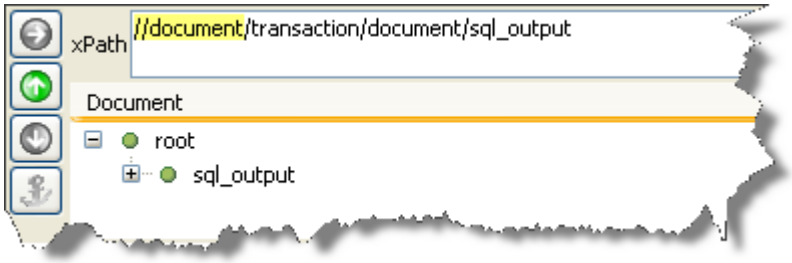


Figure 2 - 273: Generation of XPath to sql_output node

Checking whether the `sql_output` node is present is enough. We do not need to check also whether its content ("`1 row(s) updated`") is correct.

16 Click on **OK**.

The step source property is automatically updated.

17 Save your project by clicking on  or by pressing `Ctrl + S`.

The presence of the `sql_output` node in the SQL transaction XML response indicates that the SQL transaction ended normally.

We will now add:

- under the *Then* sub-step, an extra *Element* step - this step will generate a simple `<insertOk>` element including as message the `<sql_output>` node content ("`1 row(s) updated`") and indicating that the SQL insertion was correct.
- under the *Else* sub-step, an extra *Concat* step - this step will generate an `<insertError>` element, containing a concatenated error message in the following format when the SQL insertion ends in error.

For more information on the concatenated error message, see "To create and set a Concat step" on page 2-171.

We will first create the *Element* step generated when a database insertion ends normally, i.e. when an `sql_ouput` XML tag ***has been generated*** by the previously called SQL transaction.

To create and set an Element step

1 Right-click on the `IfSql_outputExists` *Then* sub-step.

A contextual menu appears.

2 Select ***New > Step***.

A **New Step** wizard is automatically launched.

3 In the **Generating XML step** area, select **Element**:

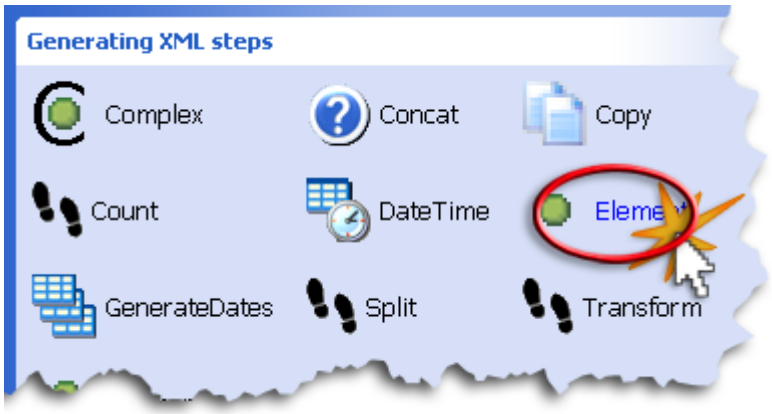


Figure 2 - 274: Selecting an Element step

An *Element* step generates a simple XML element.

4 Click **Next**.

5 In the **Name** field that appears, type in the *Element* name (for example `insertOk`):

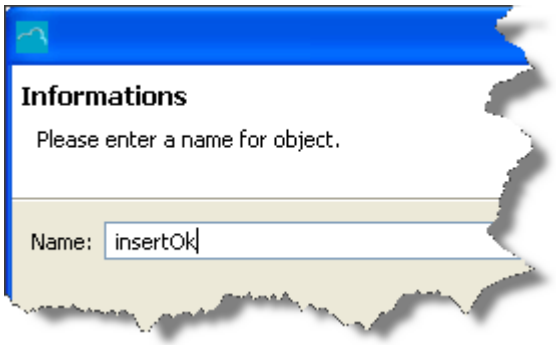


Figure 2 - 275: Entering the Element step name

6 Click on **Finish**.

The new step appears in the **Steps** folder of the project:

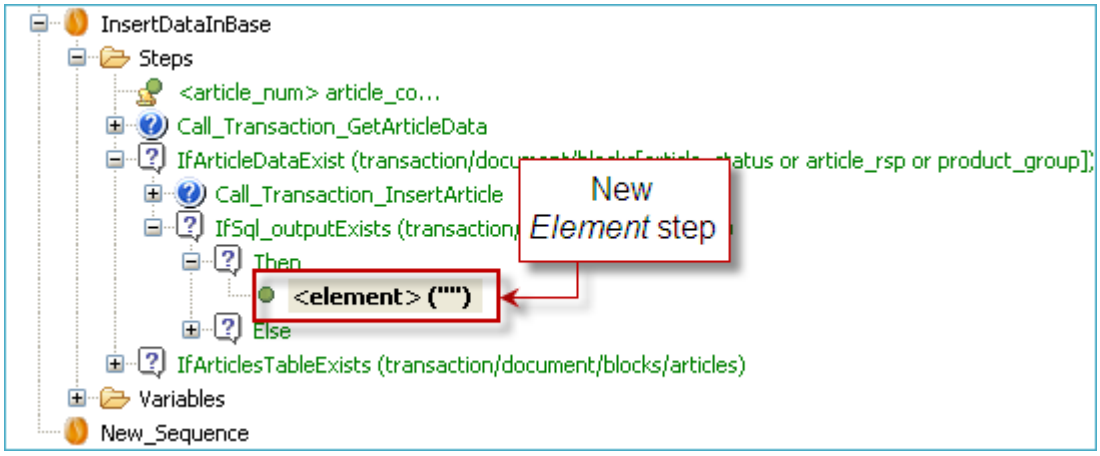


Figure 2 - 276: New Element step in Steps folder

7 Save your project by clicking on or by pressing `Ctrl + S`.

Now that the *Element* step is created, we must set its two main properties: **Node Name** and **Source**.

The value of the **Node Name** property sets the name of the XML tag generated by the *Element* step (for example `insertOk`).

As for the **Source** property, we would like the step to display the message generated by the SQL transaction when updating a database table row:`1 row(s) updated`.

To this end, we will set the *Element* step source so that it points to the `sql_output` node in the XML schema of the `Call_Transaction_InsertArticle` step.

The `Call_Transaction_InsertArticle` XML schema is known only if the schema of the `InsertArticle` SQL transaction has been extracted (see "Testing an SQL Transaction and Updating its XML Schema" on page 2-89).

8 Select the previously created step with a left-click.

The **Properties View** is automatically updated:

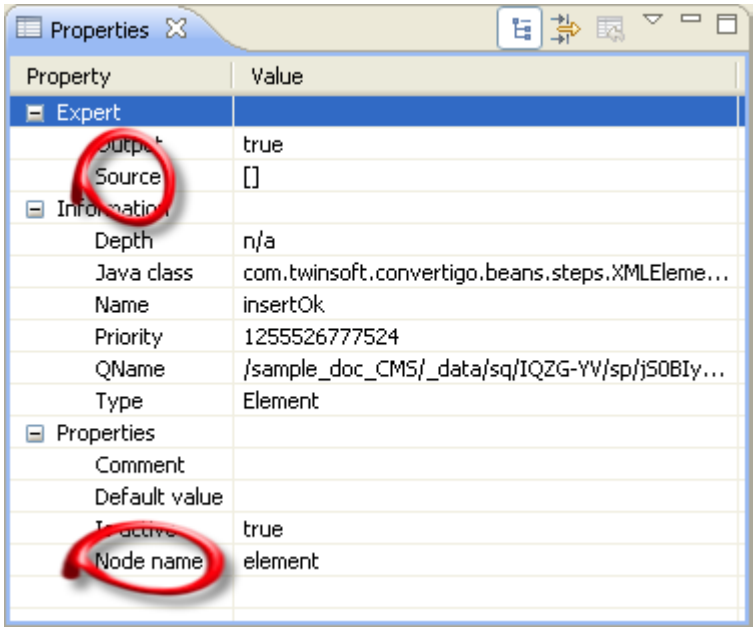


Figure 2 - 277: Element step properties

9 Click in the **Value** column of the **Node Name** property.

The value is highlighted.

10 Enter the required node name (for example `insertOk`).

11 Press `Enter`.

The value is updated: 

12 Click in the **Value** column of the **Source** property.

A  button appears.

13 Click on the  button to launch the **Step Source** wizard.

The **Step Source** wizard is automatically launched.

14 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

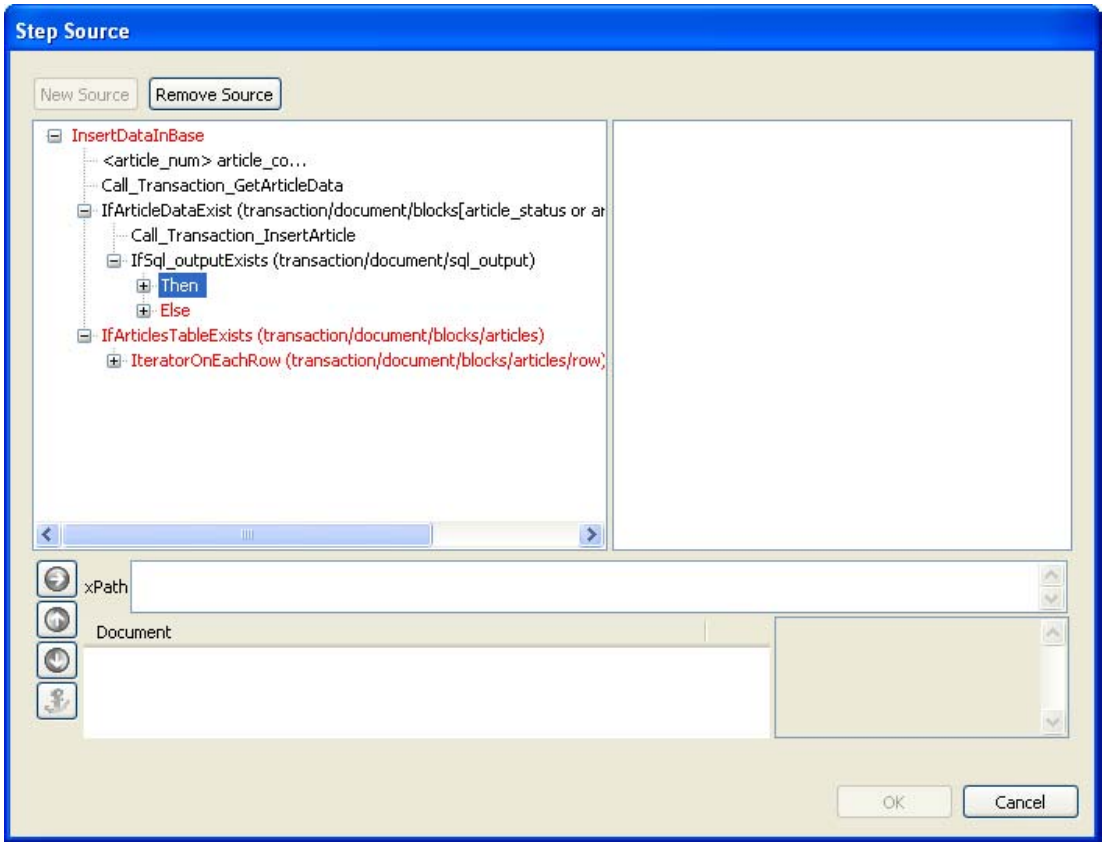


Figure 2 - 278: Setting of Element step source

The steps defined so far appear in the **Steps Tree Structure**.

15 In the **Steps Tree Structure**, select the `Call_Transaction_InsertArticle` step to display its XML schema (and the required node, `sql_output`).

The selected step's **XML Output Schema** is automatically displayed:

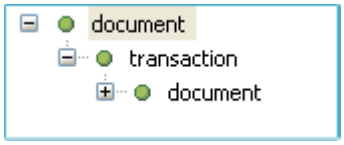


Figure 2 - 279: XML Schema of Call_transaction_InsertArticle step

16 Expand the `document` node:

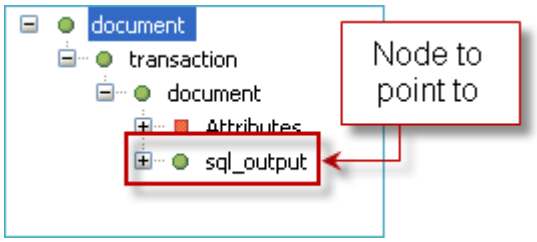


Figure 2 - 280: Node to point to in the Call_transaction_InsertArticle XML output schema

17 Select the node to point to (`sql_output`).

The XPath expression to this node is automatically generated in the **XPath Evaluator**:

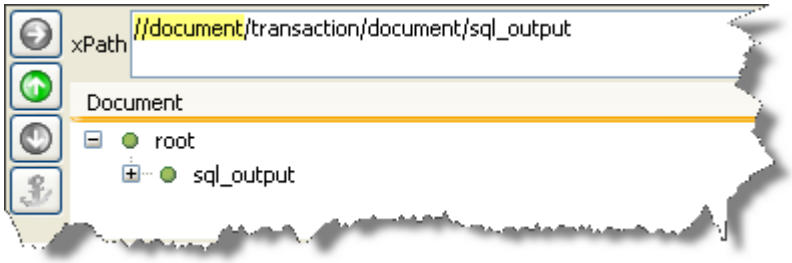


Figure 2 - 281: Generation of XPath to sql_output node

Pointing to the `sql_output` node is sufficient for the *Element* step to display its content when present.

18 Click on **OK**.

The step source property is automatically updated.

19 Save your project by clicking on  or by pressing `Ctrl + S`.

The *Then* child step is now set to insert the required message in the sequence XML output when the database insertion performed by the `InsertArticle` SQL transaction ends normally.

WHAT COMES NEXT?

We will now set the *Else* child step, which concatenates an error message when the SQL transaction returns an error (or in other words, when it does not generate the XML node indicating that the transaction ended normally, i.e. when an `sql_ouput` XML tag ***has not been generated*** by the previously called SQL transaction).

The following procedure describes how to create and set this step.

To create and set a Concat step

1 Right-click on the `IfSql_outputExists` *Else* sub-step.

A contextual menu appears.

2 Select ***New > Step***.

A **New Step** wizard is automatically launched.

3 In the **Generating XML step** area, select **Concat**:

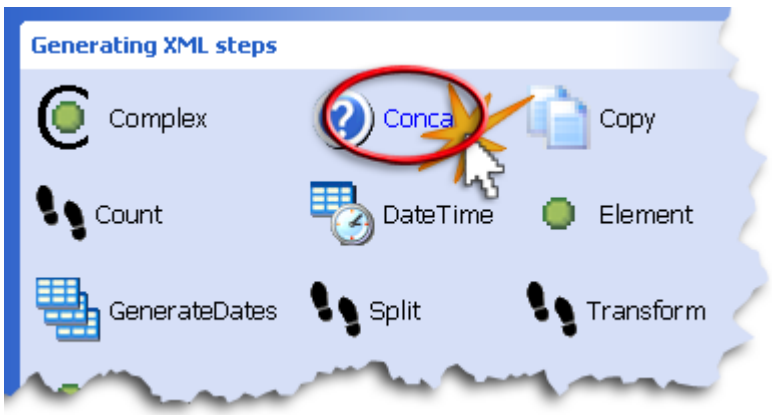


Figure 2 - 282: Selecting a Concat step

A *Concat* step concatenates either fixed strings or the content of sources.

4 Click **Next**.

5 In the **Name** field that appears, type in the *Concat* step name (for example `insertError`):

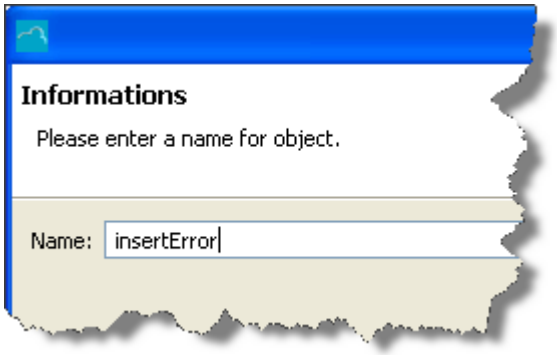


Figure 2 - 283: Entering the Concat name

6 Click on **Finish**.

The new step appears in the **Steps** folder of the project:

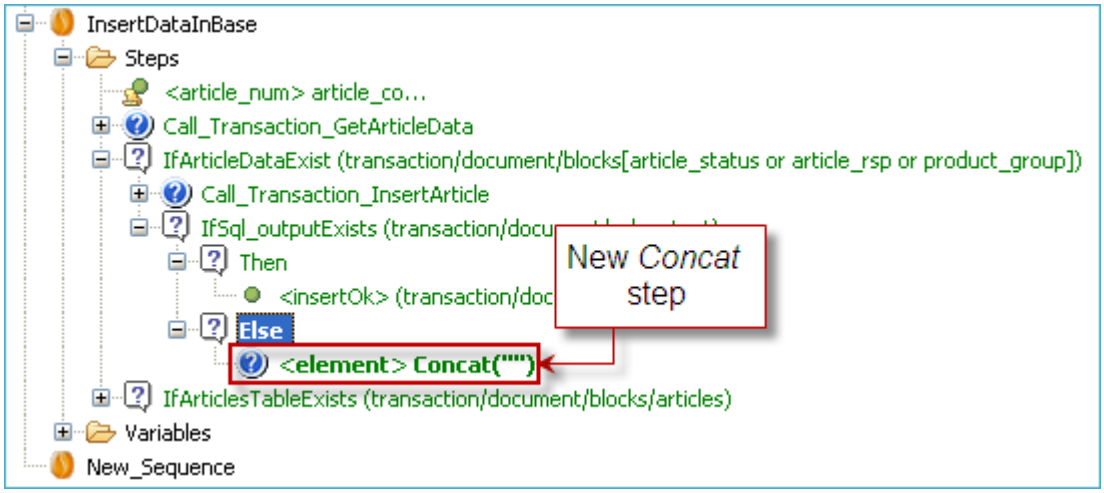


Figure 2 - 284: New Concat step in Steps folder

7 Save your project by clicking on  or by pressing `Ctrl + S`.

Now that the *Concat* step is created, we must set its two main properties: **Node Name** and **Sources**.

The value of the **Node Name** property sets the name of the XML tag generated by the *Concat* step (for example `insertError`).

As for the **Sources** property, it contains, in the case of a concatenated step, a table including the different parts of the message to be concatenated (see Table 2 - 16 on page 2-174).

We will now see how to set the *Concat* step properties.

8 Select the previously created step with a left-click.

The **Properties View** is automatically updated:

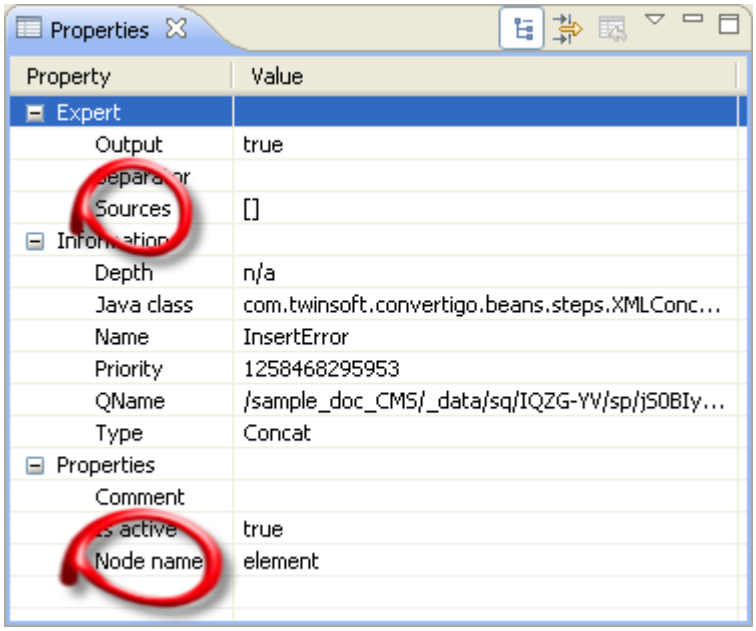


Figure 2 - 285: Concat step properties

9 Click in the **Value** column of the **Node Name** property.

The value is highlighted.

10 Enter the required node name (for example `insertError`).

11 Press `Enter`.

The value is updated: 

12 Click in the **Value** column of the **Sources** property.

A  button appears.

13 Click on the  button.

An **Action sources** window opens:

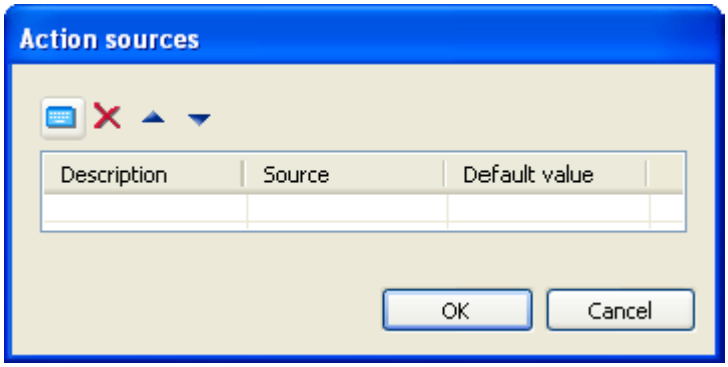


Figure 2 - 286: Action sources window

This window contains all elements to be concatenated. These elements can be set either as default (fixed) values or as sources.

The message to be displayed in case of a database insertion error is:

"Error when inserting article: `article_code`, `article_status`, `article_rsp`,

product_group"

This message can be divided into eight elements where:

Table 2 - 16: Error message elements - First Concat step

The message element...	...is...	...and points to node...
"Error when inserting article:"	a fixed string	NA
article_code	sourced from <*article_num*> *article_code* step	article_num
", "	a fixed string	NA
article_status	sourced from Call_Transaction_GetArticleData step	blocks/article_status
", "	a fixed string	NA
article_rsp	sourced from Call_Transaction_GetArticleData step	blocks/article_rsp
", "	a fixed string	NA
product_group	sourced from Call_Transaction_GetArticleData step	blocks/product_group

We will now create eight lines in the table and set their elements.

14 In the **Action sources** window, click eight times on  .

This creates eight elements in the table:

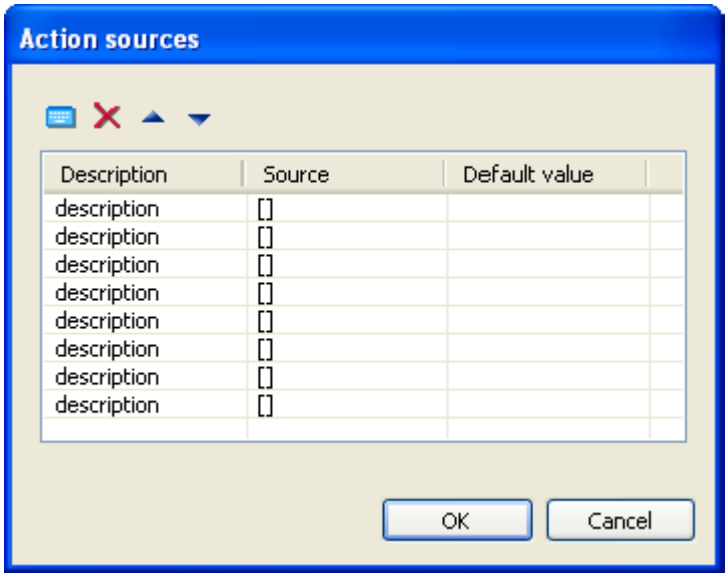


Figure 2 - 287: Action sources window with unset elements

We will now set the message elements one after another.

15 Click in the **Description** column of the first row.

The default value (description) is highlighted.

16 Enter a name for this part of the message (for example `message`).

17 Press `Enter`.

18 Click in the **Default value** column of the row.

19 Enter the required message part (for example `"Error when inserting article:"`).

20 Press `Enter`.

The first element of the error message is now set:

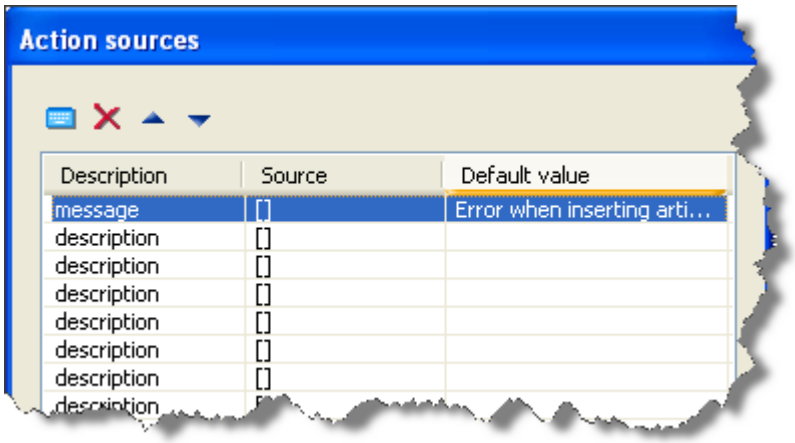


Figure 2 - 288: Setting of first element to be concatenated

21 Click in the **Description** column of the second row.

The default value (`description`) is highlighted.

22 Enter a name for this part of the message (for example `code_article`).

23 Press `Enter`.

24 Click in the **Source** column of the row.

A  button appears.

25 Click on the  button to launch the **Step Source** wizard.

The **Step Source** wizard is automatically launched.

26 Click on **New Source**.

The three panes become active (see Table *"Step Source wizard description"* on page 1-16):

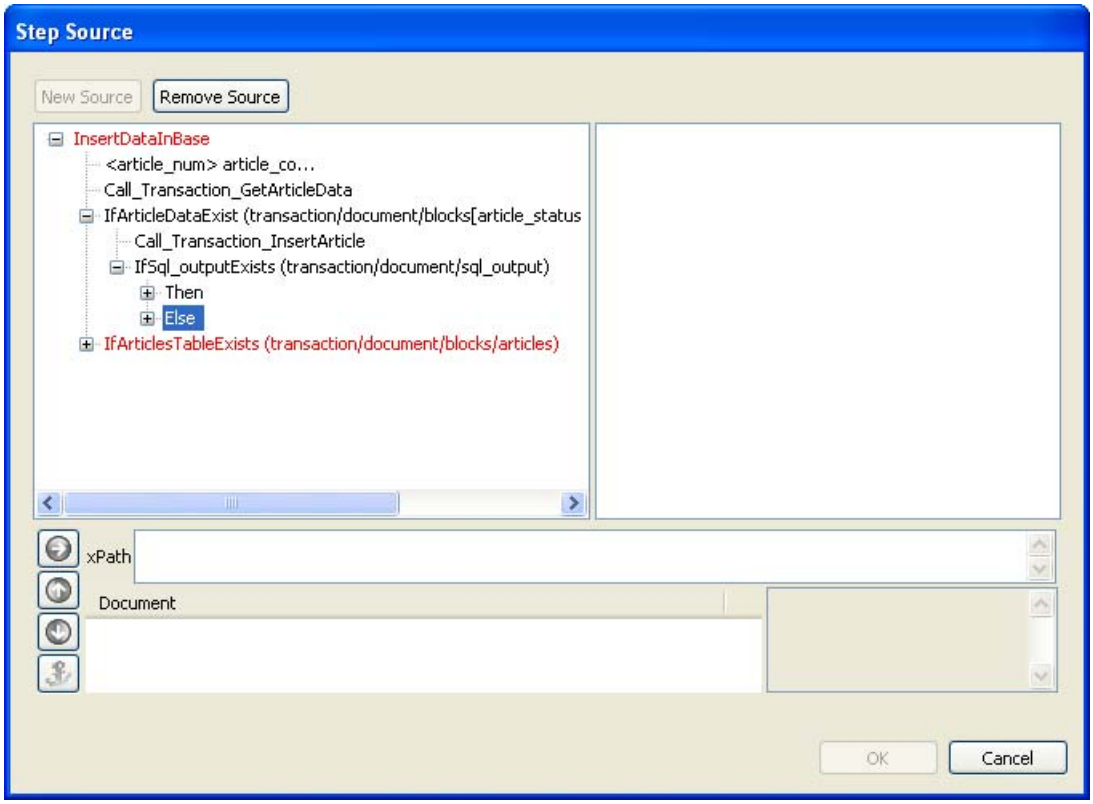


Figure 2 - 289: Setting of a Concat element source

The steps defined so far appear in the **Steps Tree Structure**.

As mentioned in Table *"Error message elements - First Concat step"* on page 2-174, the source of the `code_article` element must point towards the `article_num` node generated by the first *jElement* step.

27 In the **Steps Tree Structure**, select the first *jElement* of the sequence (`<article_num> article_code`).

The selected step's **XML Output Schema** is displayed:

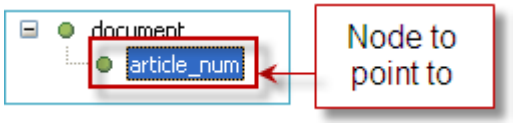


Figure 2 - 290: XML Schema of first jElement step

28 Select the node to point to (`article_num`).

The XPath expression to this node is automatically generated in the **XPath Evaluator**:

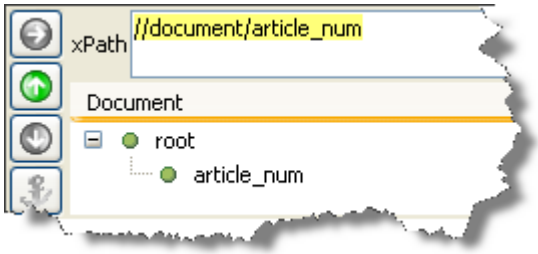


Figure 2 - 291: Generation of XPath to code_article node

29 Click on **OK**.

The message element source property is automatically updated in the **Action sources** window.

30 Repeat point 15 to 20 to set all remaining fixed elements as follows:

Table 2 - 17: Fixed elements settings

Description	Default Value
`message2`	,
`message3`	,
`message4`	,

31 Repeat points 21 to 29 to set all remaining sourced elements as follows:

Table 2 - 18: Sourced elements settings

Description	Source	
	Step	Node
`article_status`	`Call_Transaction_GetArticleData`	`blocks/article_status`
`article_rsp`		`blocks/article_rsp`
`product_group`		`blocks/product_group`

The **Action sources** window now contains all set elements:

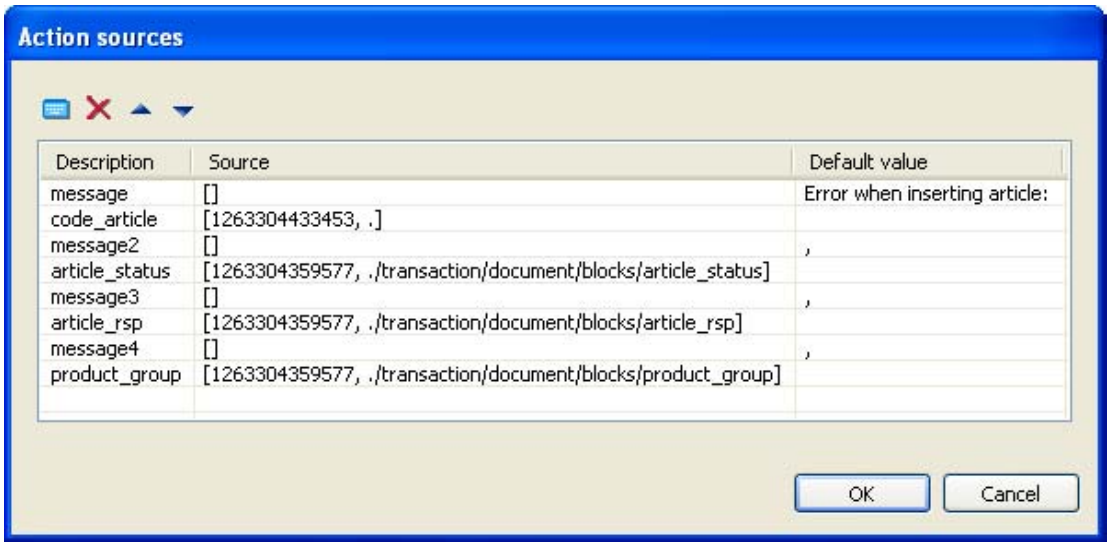


Figure 2 - 292: Message elements in Action sources window

32 Click on **OK**.

The **Action sources** window closes and the *Concat* step **Sources** property is automatically updated.

33 Save your project by clicking on or by pressing `Ctrl + S`.

The *Concat* step is now set to generate the required error message when inserting

article data in the `articles` table returns an error.

WHAT COMES NEXT?

Given that SQL transaction checks must be performed each time an SQL transaction occurs, we will now, for each of the remaining *Call Transaction* steps (`Call_Transaction_InsertArticle` and `Call_Transaction_InsertWebSite`):

1 *Copy* the `IfSql_outputExists` step (including its *Then* and *Else* sub-steps) and paste it as child step of the selected *Call Transaction* step.

2 *Set* the pasted `IfSql_outputExists` step **Source** property so that it points to the `sql_output` node generated by the required *Call Transaction* step.

3 *Set* the pasted *Then* and *Else* sub-steps **Sources** property so that the sources of:

- the `insertOk` element (*Then* sub-step),
- message elements to be concatenated (*Else* sub-step),

point towards required nodes.

The following procedure describes how to copy and paste the `IfSql_outputExists` step as following brother of remaining *Call Transaction* steps.

To copy and paste the second `IfSql_outputExists` step

1 Right-click on the previously created `IfSql_outputExists` step (see *"To create and set an SQL transaction check step"* on page 2-162).

A contextual menu appears.

2 Select **Copy**:

You can also copy the step by using the `Ctrl + C` shortcut, or drag and drop the step on the appropriate destination step.

3 Right-click on the destination step (for example `IteratorOnEachRow` parent to `Call_Transaction_InsertArticle`).

A contextual menu appears.

4 Select **Paste**.

A confirmation window opens:

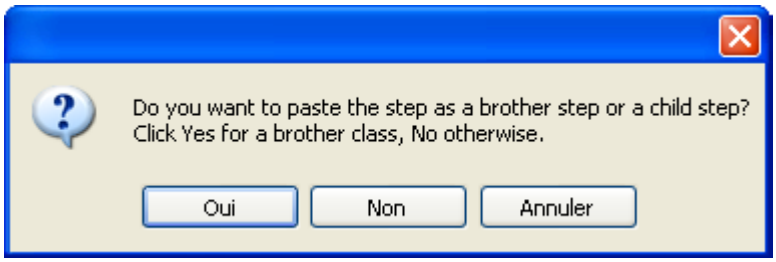


Figure 2 - 293: Paste confirmation window

5 To paste the new step as a child of the selected step, click **No**.

The step is automatically pasted as a child of the selected parent step, at the end of the parent steps tree structure:

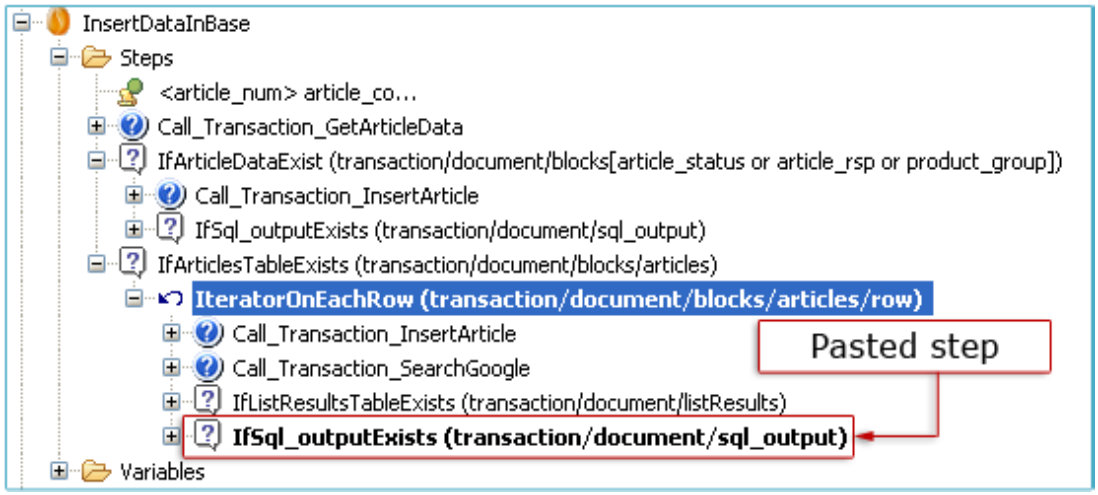


Figure 2 - 294: Pasted step (second IfSql_outputExists step)

We want this step to be executed straight after the call on the `InsertArticle` SQL transaction, so we need to increase the `IfSql_outputExists` step priority.

6 Select the pasted `IfSql_outputExists` step with a left-click.

7 In the **Projects View** toolbar, click on the **Increase selected object(s) priority** icon twice so that the step appears just below the `Call_Transaction_InsertArticle` step.

Fort more information about how to increase an object's priority, see "To increase a step priority" on page 2-27.

The step is now properly ranked in the **Steps** folder tree structure:

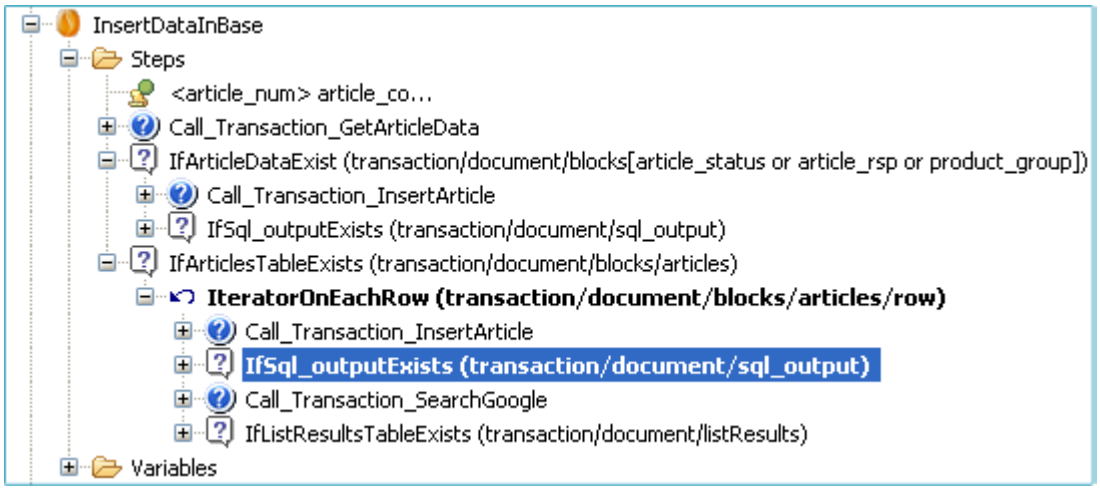


Figure 2 - 295: Step priority increased

8 Save your project by clicking on or by pressing `Ctrl + S`.

We now need to set the `Ifsql_outputExists` source so that it points towards the

`sql_output` node generated by the ***second*** `Call_Transaction_InsertArticle` SQL transaction.

9 To set the pasted `Ifsql_outputExists` source, repeat points 9 to 16 of the procedure for creating and setting an *IfExistThenElse* step (see *"To create and set an SQL transaction check step"* on page 2-162) - in the **Step Source** wizard, select the second `Call_Transaction_InsertArticle` step then the corresponding `sql_output` node (see Figure 2 - 296):

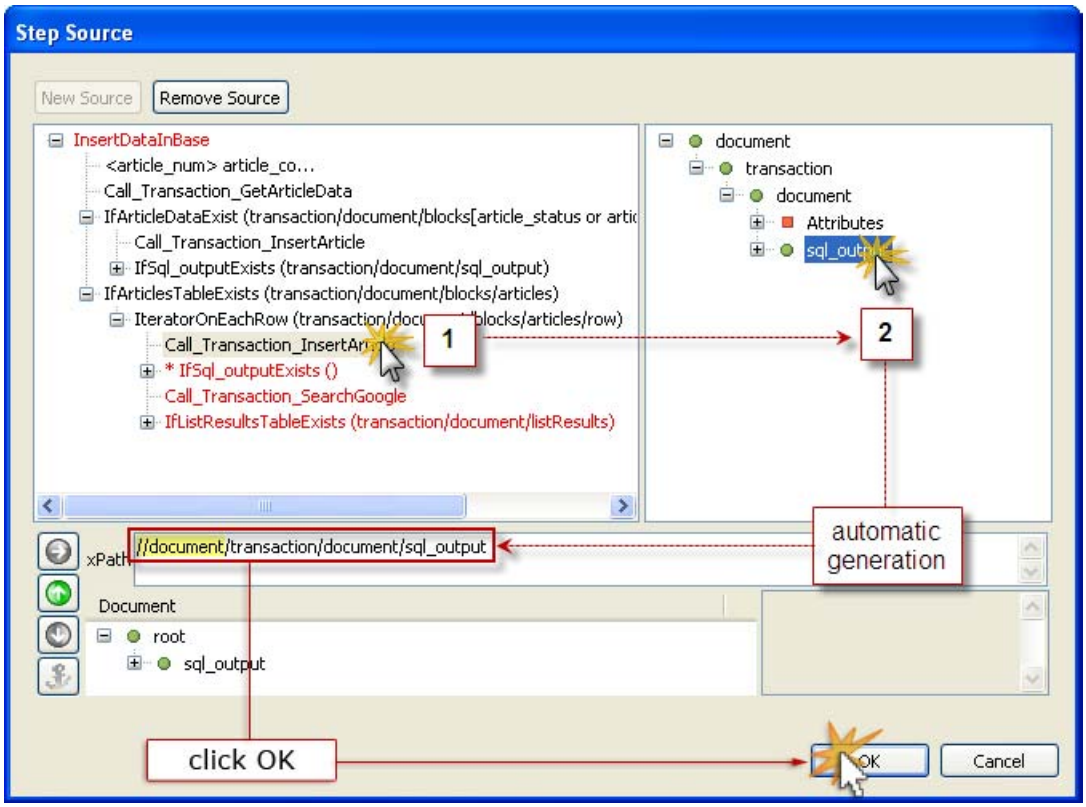


Figure 2 - 296: Setting the second IfSql_outputExists step source

Now that the `IfSql_outputExists` step source is set, we will set the messages generated when the SQL transaction ends normally (*Then* sub-step) or in error (*Else* sub-step):

10 Save your project by clicking on  or by pressing `Ctrl + S`.

11 In the **Projects View**, expand the `IfSql_outputExists` step by clicking on the ⊞ symbol to the left of the step.

12 Expand the `Then` sub-step by clicking on the ⊞ symbol to the left of the step.

13 Select the `<insertOk> (sql_output)` step with a left-click.

The **Properties View** is automatically updated.

14 To set this step source, repeat points 9 to 18 of the procedure for creating and setting an *Element* step (see *"To create and set an Element step"* on page 2-167) - in the **Step Source** wizard, select the second `Call_Transaction_InsertArticle` step then the corresponding `sql_output` node (see Figure 2 - 296).

15 Save your project by clicking on or by pressing `Ctrl + S`.

16 In the **Projects View**, expand the `Else` sub-step by clicking on the symbol to the left of the step.

17 Select the `<insertError> Concat(,...)` step with a left-click.

The **Properties View** is automatically updated.

18 Click in the **Value** column of the **Sources** property.

A button appears.

19 Click on the button.

The **Action sources** window opens with message elements defined when setting the first *Concat* step (see *"To create and set a Concat step"* on page 2-171):

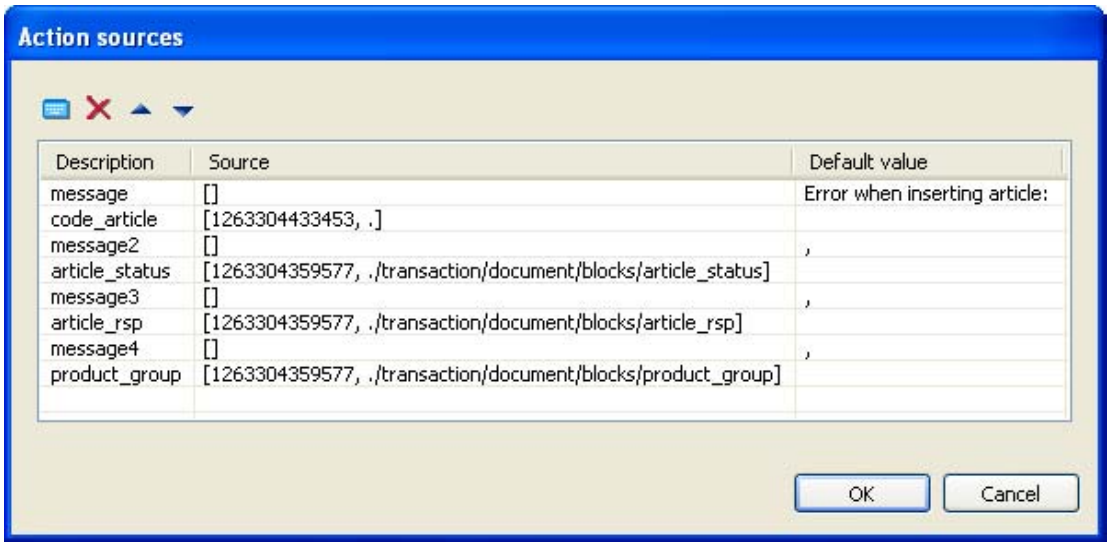


Figure 2 - 297: Message elements set for the first Concat step

The message elements to be set for this second *Concat* step correspond to the data inserted in the `articles` table when executing the second `Call_Transaction_InsertArticle`: article code, article rsp and article name (see Table *"`Call_Transaction_InsertArticle` step - Imported variable sources"* on page 2-110).

The `rsp_produit` and `product_group` message elements defined for the first *Concat* step therefore disappear and are replaced by a `article_name`, so that the message displayed in case of a database insertion error while executing the second `Call_Transaction_InsertArticle` is:

"Error when inserting article: `article_code`, `article_status`, `article_name`".

This message can be divided into six elements where:

Table 2 - 19: Error message elements - Second Concat step

The message element...	...is...	...and points to node...
"Error when inserting article:"	a fixed string	NA
`article_code`	sourced from *`IteratorOnEachRow`* step	`blocks/articles/row/code`
`", "`	a fixed string	NA
`article_status`	sourced from *`IteratorOnEachRow`* step	`blocks/articles/row/status`
`", "`	a fixed string	NA
`article_name`	sourced from *`IteratorOnEachRow`* step	`blocks/articles/row/name`

We will now:

- delete or add required rows,
- change the setting of the remaining element sources so that they point towards the nodes described in Table *"Error message elements - Second Concat step"* on page 2-182.

20 Click in the **Description** column of the sixth element (`article_rsp`).

The element is highlighted.

21 Click three times on the **Delete row** icon to delete the last three rows of the table.

22 Click on the **Add row** icon .

A new row is added.

23 Click in the **Description** column of the newly created row.

24 The default value (`description`) is highlighted.

25 Enter the name of the message element (`article_name`).

The six message elements are now created. Fixed elements with a default value can remain unchanged.

We will now set the sources of the remaining three elements.

26 Click in the **Source** column of the second element (`article_code`).

A button appears.

27 Click on the button to launch the **Step Source** wizard.

The **Step Source** wizard is automatically launched with the three panes already active because the source is already set:

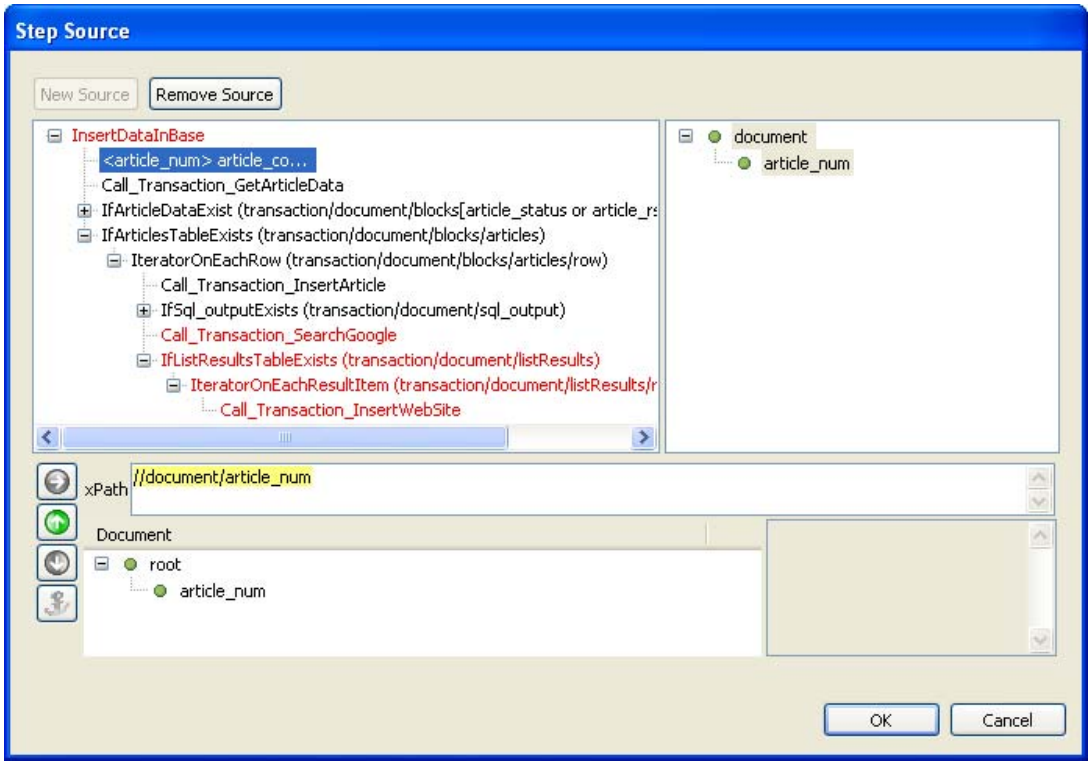


Figure 2 - 298: Setting of a Concat element source (code_article) before changes

As mentioned in Table *"Error message elements - Second Concat step"* on page 2-182, the source of the `code_article` element must point towards the `code_article` node of the `articles` XML table `row` generated by the `Call_Transaction_GetArticleData` step, and ***currently iterated on*** by the `IteratorOnEachRow` step.

We must therefore select the `IteratorOnEachRow` step in the **Steps Tree Structure** of the **Step Source** wizard.

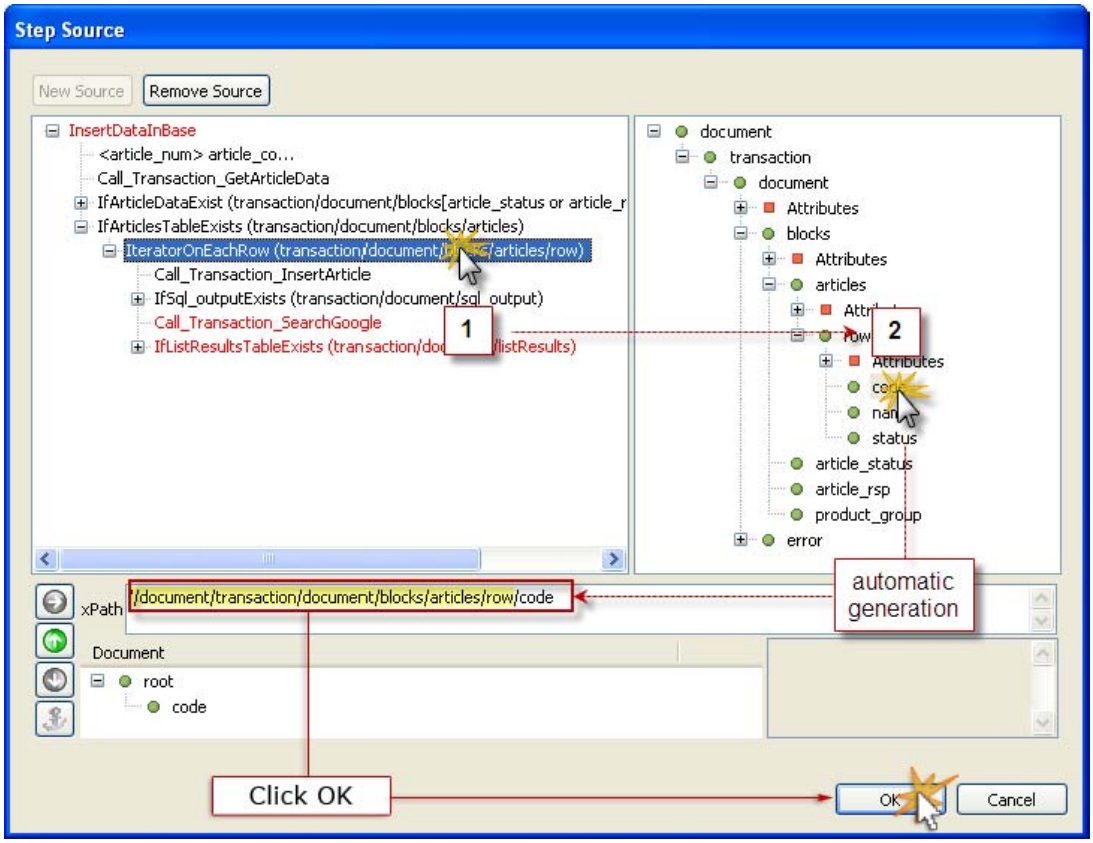


Figure 2 - 299: Setting of a Concat element source (code_article)

28 In the **Steps Tree Structure**, select the `IteratorOnEachRow` step.

The selected step's **XML Output Schema** is displayed.

29 In the **XML Output Schema**, select the node to point to (`code`).

The XPath expression to this node is automatically generated in the **XPath Evaluator**.

30 Click on **OK**.

The whole process is illustrated by the figure 2 - 300.

The message element source property is automatically updated in the **Action sources** window.

31 Repeat points 26 to 30 to set the fourth (`article_status`) and sixth (`article_name`) message element sources as pointing to the nodes described in Table *"Error message elements - Second Concat step"* on page 2-182.

The message elements of the second *Concat* step as well as the whole second `IfSql_ouputExists` step are now properly set:

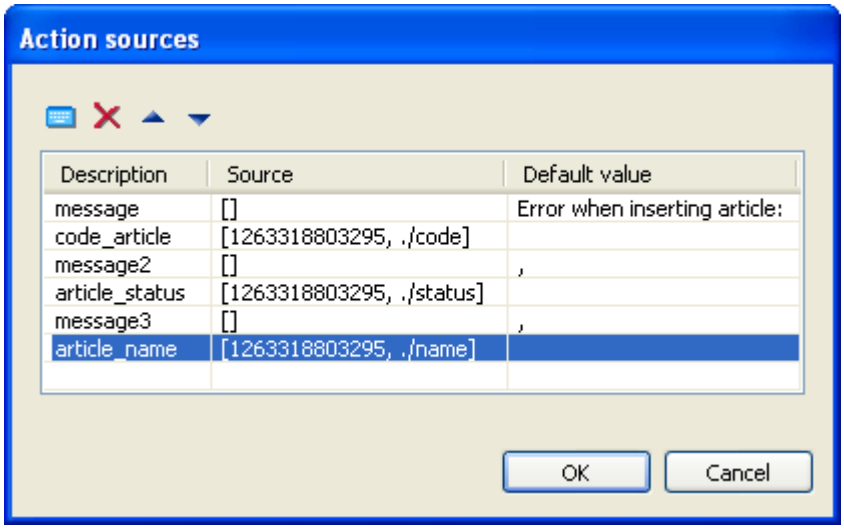


Figure 2 - 300: Message elements of the second Concat step

32 Click on **OK**.

The **Action sources** window closes and the *Concat* step **Sources** property is automatically updated.

33 Save your project by clicking on  or by pressing `Ctrl + S`.

WHAT COMES NEXT?

We must still copy and paste a third `IfSql_ouputExists` step, straight after the third Call Transaction step, `Call_Transaction_InsertWebSite`.

To copy and paste the third `IfSql_outputExists` step

1 Right-click on the previously created `IfSql_outputExists` step (see *"To copy and paste the second `IfSql_outputExists` step"* on page 2-178).

A contextual menu appears.

2 Select **Copy**:

You can also copy the step by using the `Ctrl + C` shortcut, or drag and drop the step on the appropriate destination step.

3 Right-click on the parent step (for example `IteratorOnEachResultItem` parent to `Call_Transaction_InsertWebSite`).

A contextual menu appears.

4 Select **Paste**.

A confirmation window opens:

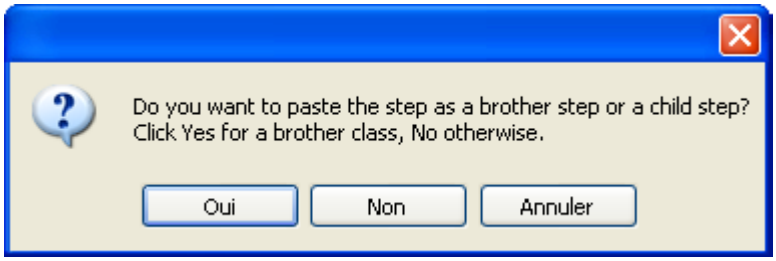


Figure 2 - 301: Paste confirmation window

5 To paste the new step as a child of the selected step, click **No**.

The step is automatically pasted as a child of the selected parent step, at the end of the parent Steps Tree Structure. Unlike previous `IfSql_outputExists` steps, we do not need to reposition it:

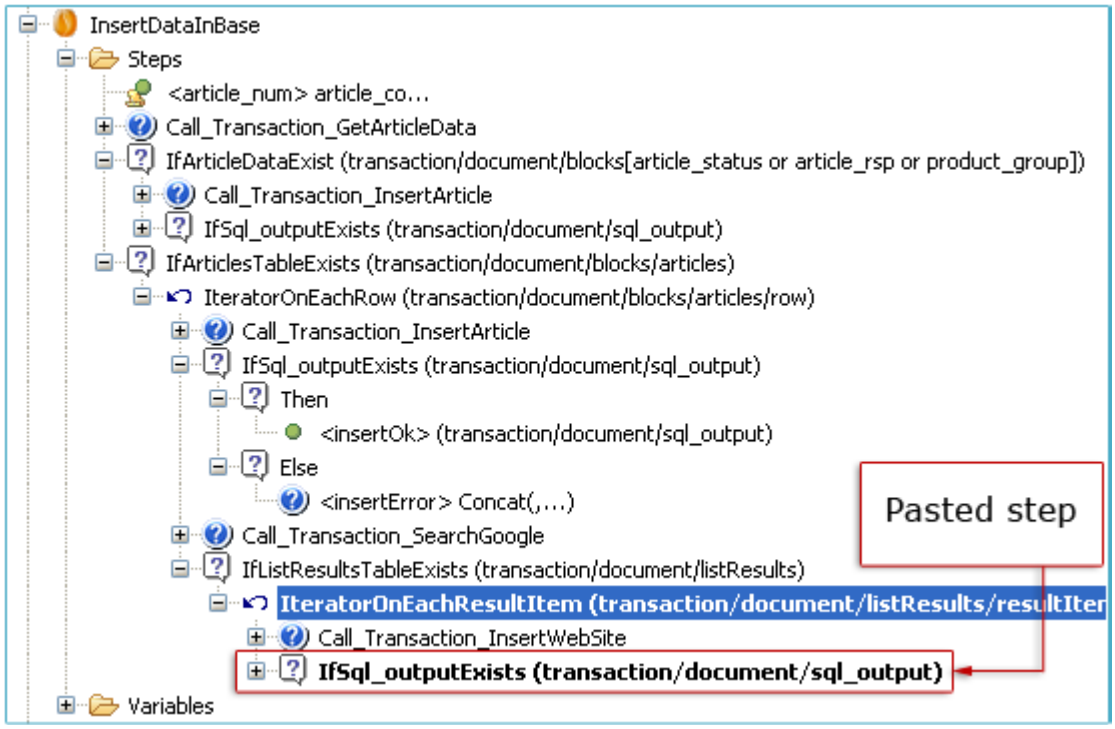


Figure 2 - 302: Pasted step (third IfSql_outputExists step)

6 Save your project by clicking on  or by pressing `Ctrl + S`.

We will now set the `Ifsql_outputExists` source, so that it points towards the `sql_output` node generated by the `Call_Transaction_WebSite` SQL transaction.

7 To set the newly pasted `Ifsql_outputExists` source, repeat points 9 to 16 of the procedure for creating and setting an *IfExistThenElse* step (see *"To create and set an SQL transaction check step"* on page 2-162) - in the **Step Source** wizard, select the `Call_Transaction_InsertWebSite` step then the corresponding `sql_output` node (see Figure 2 - 303):

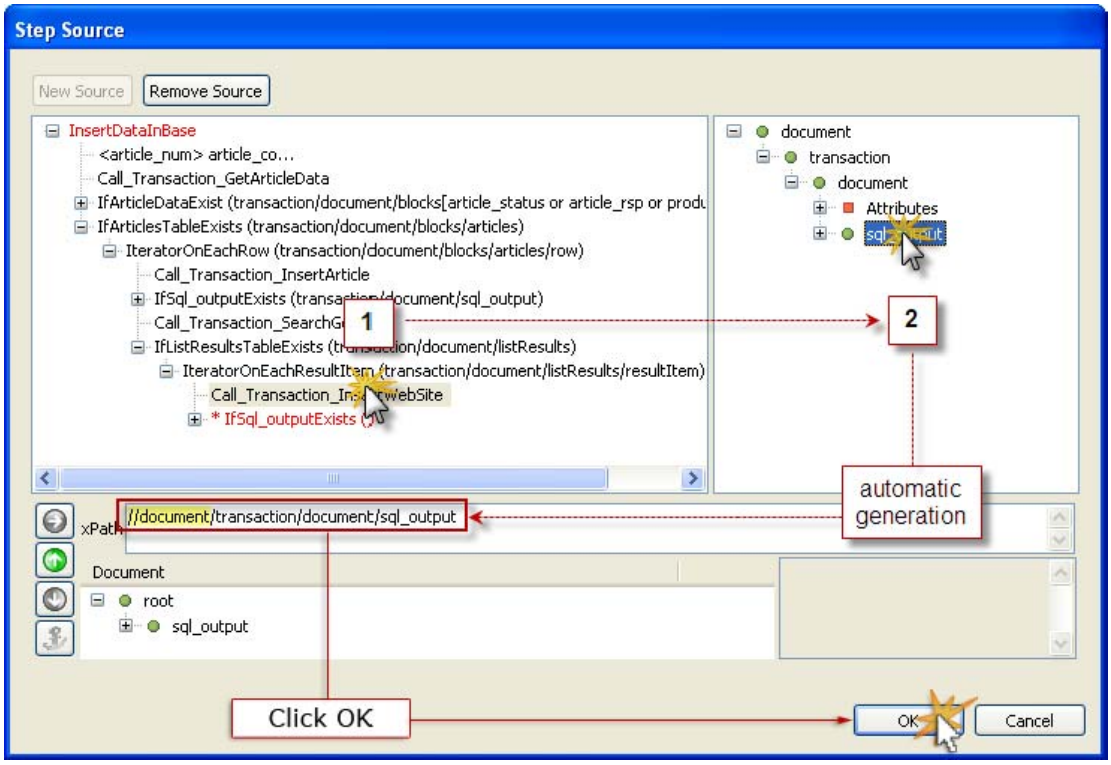


Figure 2 - 303: Setting the third IfSql_outputExists step source

Now that the `IfSql_outputExists` step source is set, we will set the messages generated when the SQL transaction ends normally (*Then* sub-step) or in error (*Else* sub-step):

8 Save your project by clicking on or by pressing `Ctrl + S`.

9 In the **Projects View**, expand the `IfSql_outputExists` step by clicking on the ⊞ symbol to the left of the step.

10 Expand the `Then` sub-step by clicking on the ⊞ symbol to the left of the step.

11 Select the `<insertOk> (sql_output)` step with a left-click.

The **Properties View** is automatically updated.

12 To set this step source, repeat points 9 to 18 of the procedure for creating and setting an *Element* step (see *"To create and set an Element step"* on page 2-167) - in the **Step Source** wizard, select the second `Call_Transaction_InsertWebSite` step then the corresponding `sql_output` node (see Figure 2 - 303).

13 Save your project by clicking on or by pressing `Ctrl + S`.

14 In the **Projects View**, expand the `Else` sub-step by clicking on the ⊞ symbol to the left of the step.

15 Select the `<insertError> Concat(,...)` step with a left-click.

The **Properties View** is automatically updated.

16 Click in the **Value** column of the **Sources** property.

The **Action sources** window opens with message elements defined when setting the

first *Concat* step (see *"To create and set a Concat step"* on page 2-171):

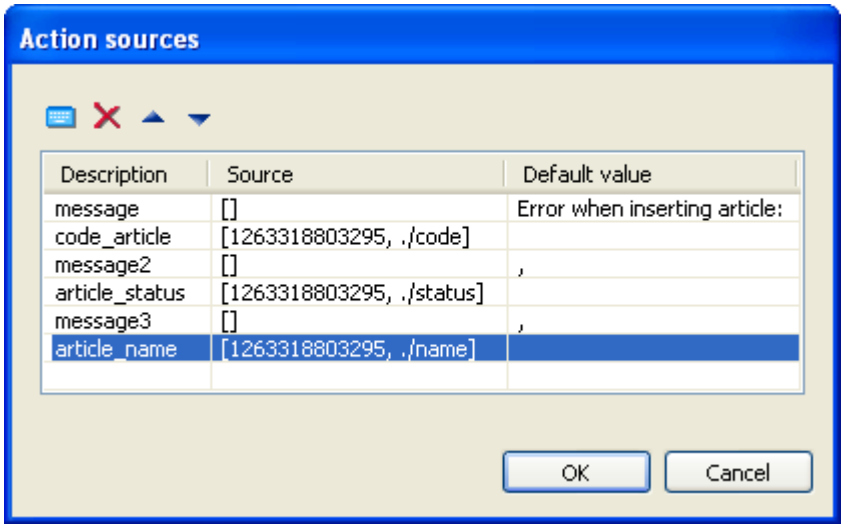


Figure 2 - 304: Message elements set for the second Concat step

The message elements to be set for this third *Concat* step correspond to the data inserted in the `web_site` table when executing the `Call_Transaction_InsertWebSite`: article code, url and title (see Table *"`Call_Transaction_InsertWebSite` step - Imported variable sources"* on page 2-134).

The `article_code` message element defined for the second *Concat* step stays valid for this third *Concat* element.

The first message element, with default text value and remaining two message elements (`article_status` and `article_name`) must be changed (to *"Error when inserting web site:"* , `title` and `url`) so that the message displayed in case of a database insertion error while executing the `Call_Transaction_InsertWebSite` is:

"Error when inserting web site: `article_code`, `url`, `title`"

This message can be divided into six elements where:

Table 2 - 20: Error message elements - Third Concat step

The message element...	...is...	...and points to node...
"Error when inserting web site:"	a fixed string	NA
`article_code`	sourced from *`IteratorOnEachRow`* step	`blocks/articles/row/code`
`" , "`	a fixed string	NA
`title`	sourced from *`IteratorOnEachResultItem`* step	`listResults/resultItem/title`
`" , "`	a fixed string	NA
`url`	sourced from *`IteratorOnEachResultItem`* step	`listResults/resultItem/url`

We will now:

- change the setting of the first message element so that it contains the correct default value ,
- change the setting of the remaining element sources so that they point towards the nodes described in Table *"Error message elements - Third Concat step"* on page 2-188.

17 Click in the **Default value** column of the first element (`message`).

The element is highlighted.

18 Enter the required message part (for example `"Error when inserting web site:"`).

19 Press `Enter`.

20 Click in the **Description** column of the fourth element (`article_status`).

The element is highlighted.

21 Change the name of the message element (for example to `title`).

22 Click in the **Source** column of this element (`title`).

A  button appears.

23 Click on the  button to launch the **Step Source** wizard.

The **Step Source** wizard is automatically launched with the three panes already active because the source is already set:

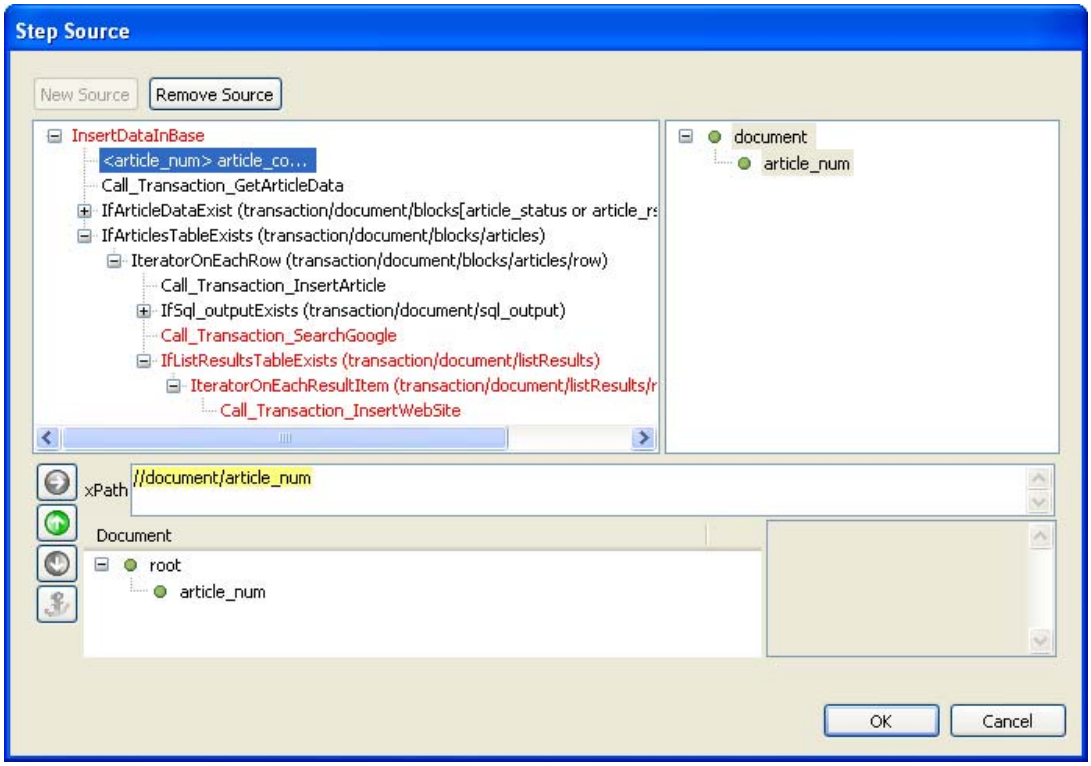


Figure 2 - 305: Setting of a Concat element source (title) before changes

As mentioned in Table *"Error message elements - Third Concat step"* on page 2-188, the source of the `title` element must point towards the `title` node of the

`listResults` XML table `resultItem` generated by the `Call_Transaction_SearchGoogle` step, and ***currently iterated on*** by the `IteratorOnEachResultItem` step.

We must therefore select the `IteratorOnEachResultItem` step in the **Steps Tree Structure** of the **Step Source** wizard.

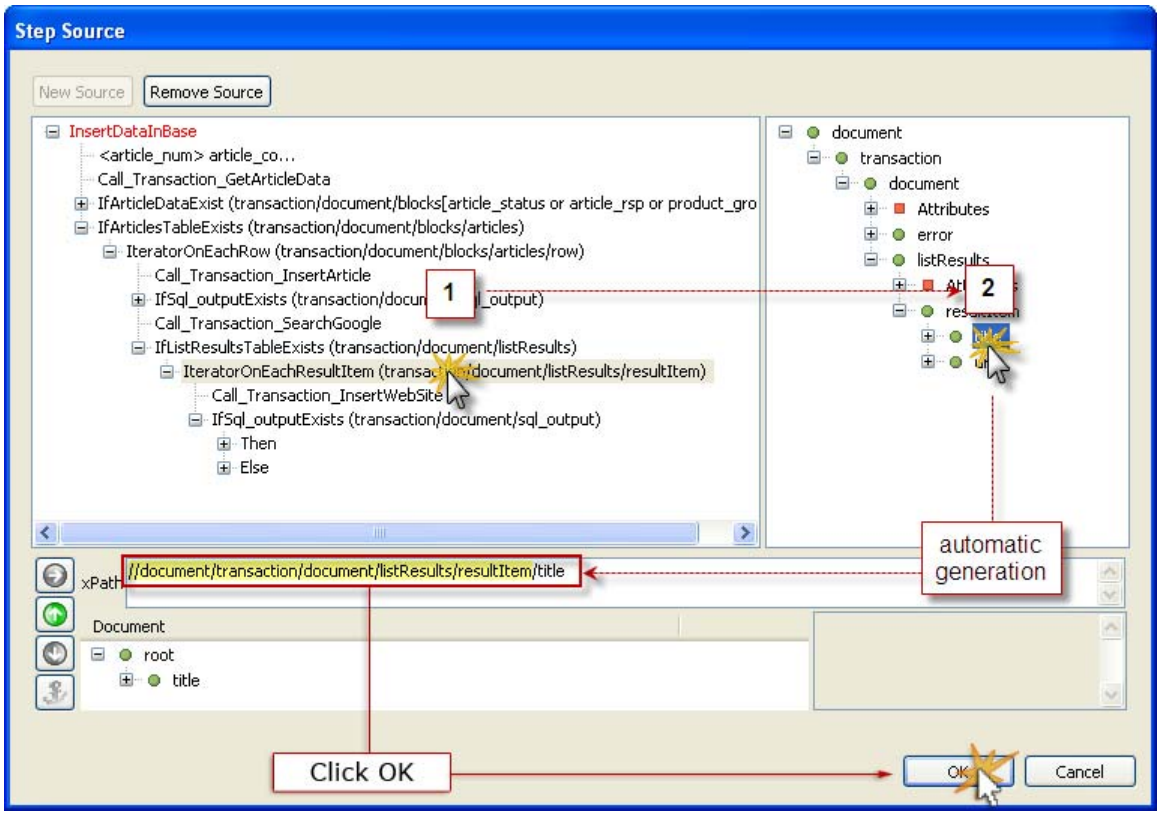


Figure 2 - 306: Setting of a Concat element source (title)

24 In the **Steps Tree Structure**, select the `IteratorOnEachResultItem` step.

The selected step's **XML Output Schema** is displayed.

25 Select the node to point to (`title`).

The XPath expression to this node isc automatically generated in the **XPath Evaluator**. The whole process is illustrated by the figure 2 - 306.

26 Click on **OK**.

The message element source property is automatically updated in the **Action sources** window.

The fourth message element (`title`) is now properly set.

27 Repeat points 20 to 26 of this procedure to:

- Change the sixth message element description from `nom_article` to `url`,
- Set its source so that it points towards the node described in Table *"Error message elements - Third Concat step"* on page 2-188.

The message elements of the third *Concat* step as well as the whole third

`IfSql_ouputExists` step are now properly set:

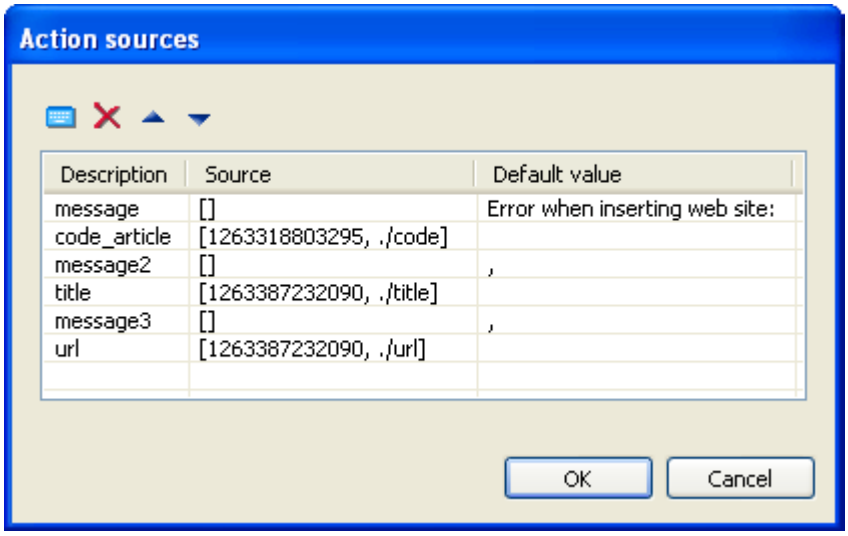


Figure 2 - 307: Message elements of the third Concat step

28 Click on **OK**.

The **Action sources** window closes and the *Concat* step **Sources** property is automatically updated.

29 Save your project by clicking on or by pressing `Ctrl + S`.

WHAT COMES NEXT?

We will now execute the sequence to see if the XML output generated by the sequence meets our need of a clear output informing us of possible database insertion errors.

To execute the sequence again, follow one of the two methods described in the sequence execution procedure (see *"Executing a Sequence"* on page 2-138).

*Remember to reset the **Output** property of all Call Transaction steps to `false`. Also, prior to executing your sequence live, remember to reset the **Max. iterations** property of Iterator steps.*

After the sequence has been executed, the sequence XML output is displayed in the **XML** tab of the **Sequence View**:

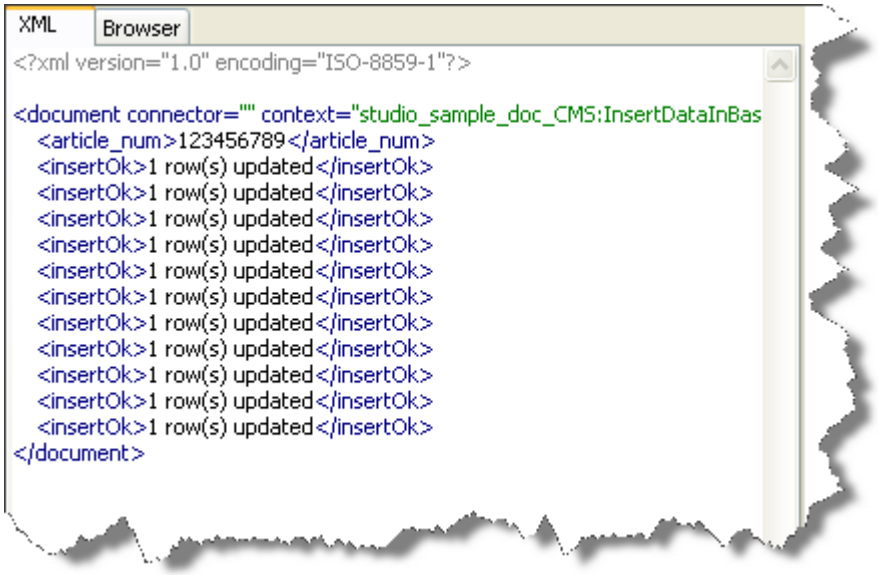


Figure 2 - 308: Sequence XML output - SQL transactions end normally

If a problem occurs, when inserting in the database (for example if a wrong user password has been set), error messages are displayed in the sequence XML output:

Figure 2 - 309: Sequence XML output - SQL transactions end in error

The different elements of concatenated error messages are clearly visible (in following figure, an error message generated by the *Concat* step of the first `IfSql_outputExists` step, see *"To create and set a Concat step"* on page 2-171):

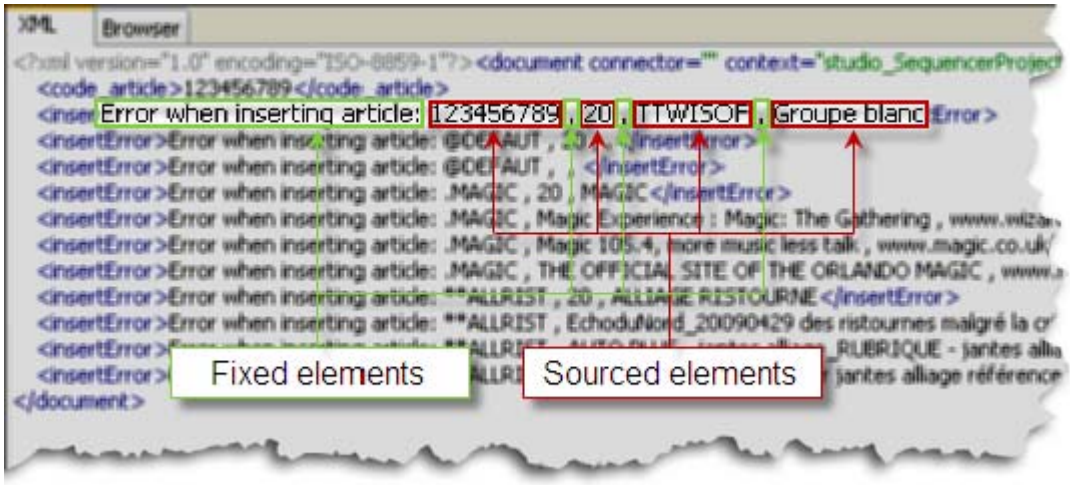


Figure 2 - 310: Detail of an error message