Chapter 10: Integration and Transformation Activities

Welcome to this specialized chapter on Integration, Transformation and Mapping Activities in BIZUIT. In previous chapters we discovered how BIZUIT becomes the integration center of our systems, allowing us to automate critical tasks and improve connectivity between the different applications we use. We will learn how to connect and extract data from external services, how to transform complex information to suit the needs of various systems, and how to design efficient workflows, all through a service-oriented structure.

BIZUIT and SOA Overview

BIZUIT is built on Service-Oriented Architecture (SOA), which means that we organize applications into independent modules or services that communicate using standards. This approach offers us three main advantages:

- **Process automation:** Integration activities eliminate manual tasks, allowing data to flow automatically.
- **Connectivity between systems:** We connect databases, web services (SOAP and REST), files and other resources, ensuring a continuous flow of information.
- **Scalability and flexibility:** The platform adapts to the growing needs of our company, facilitating the integration of new systems without major changes in infrastructure.

Ideal Audience

This chapter is designed to:

- Software developers and integration specialists.
- Solution architects interested in integrating BIZUIT with complex external systems.

Objectives

The purpose of this chapter is to equip us with the necessary tools and knowledge to configure and manage integration, transformation and mapping activities in BIZUIT.

We will learn how to:

- Set up connections to external systems (e.g., databases and web services) using integration activities.

- Transform and map data to fit the specific requirements of each system, either by moving from JSON to XML or by adapting data structures.
- Design workflows that automate processes and optimize communication between systems, reducing manual intervention.

Unit 1: Introduction to Integration, Transformation and Mapping Activities

BIZUIT Integration Activities

Integration activities in BIZUIT are fundamental tools that allow us to automatically connect our platform with other external systems and applications. These activities function as linkages, facilitating the exchange of information between BIZUIT and resources such as databases, web services, and third-party applications. By integrating in this way, we are able to maintain dynamic and adaptive workflows, allowing us to make decisions based on up-to-date and accurate data.

For example, if we have a customer management system in a SQL database and a web service that queries sales data, our integration activities will be responsible for connecting these sources to automate the flow of information, thus optimizing our processes.

How Workflows Work

Within our workflows, integration activities act as connectors that enable data input and output to or from external systems. Depending on the objective we pursue, these connectors can:

- Query data in external databases.
- Execute commands on remote systems.
- Obtain or send information using web services (both REST and SOAP).

In this way, BIZUIT becomes the orchestrator of our processes, ensuring that each step receives the most up-to-date information and allowing the synchronization of data between critical systems, such as updating inventory or managing customer requests.

Examples of Integration Activities in BIZUIT

To illustrate the potential of these activities, let's consider some practical examples:

- **Integration with REST and SOAP services:** With a REST integration activity, we can query real-time data from an external API, such as the exchange rate of a currency or the availability of a product. Similarly, through SOAP services, we interact with legacy systems and obtain automatic responses.

- **Connection to external databases:** BIZUIT allows us to connect with databases such as SQL Server, Oracle or MySQL, which makes it easier for us to perform real-time queries and operations, essential for financial or customer management systems.
- **Remote file access:** We can set up activities that read or write files to remote locations, such as FTP servers or cloud storage. For example, it is possible to automatically download an order CSV file and upload it to our system for processing.

Thanks to these examples, we see how BIZUIT optimizes connectivity and automation, allowing different data sources to be integrated without compatibility or synchronization issues, ensuring an agile and coordinated workflow.

Now that we've understood what integration activities are and how they work in BIZUIT, we've seen how these tools allow us to connect our workflows in a transparent and automated way with external systems. This is the crucial first step towards complete and effective automation.

To take our integration to the next level, we need to adapt data between different formats and structures, which leads us to transformation activities. In the next section, we will explore how these activities allow us to convert and map data, expanding our integration possibilities and ensuring that each system receives the information in the right format.

Introduction to Format Transformation Activities

Format transformation activities in BIZUIT allow us to convert data from one format to another, which is crucial in integrations where each system requires data in a specific format. Let's imagine that we have a workflow in which we need to transform data from one system into JSON so that another system receives it in XML or CSV. These activities are critical, as in most business environments data comes from diverse sources in unique formats. Without the possibility of transforming them, it would be very difficult for us to achieve fluid communication between the systems.

BIZUIT allows us to adjust the data to the required format automatically, streamlining the flow of information between different systems and ensuring that the transformation is carried out accurately and in real time.

Concept of Mappings and Transformations

At BIZUIT, data mappings and transformations are core elements that ensure information is interpreted correctly between different systems. Mapping allows us to associate specific fields in one system with those in another, so that each piece of data is aligned and positioned in the right place when integrated with other systems.

On the other hand, transformations allow us to modify or adjust the structure and format of the data to suit the needs of each system. This includes operations such as changing the format of a date, converting a string to a number, or adjusting the hierarchical structure of data from JSON to XML. Both concepts work together to make sure that information flows correctly and that each system can understand and use the data received.

Conclusion

With this, we have completed Unit 1, in which we explore the fundamental concepts of integration, transformation and mapping activities in BIZUIT. We now understand how integration activities connect external systems, how format transformations allow us to tailor data to each system's requirements, and how data mappings and transformations ensure that information is placed in the right place and format.

In the next unit, we will take it a step further and focus on the practical setup of integration activities in BIZUIT, where we will learn how to apply these concepts in real-world environments.

Unit 2: Setting Up Integration Activities

Configuring the SQL Statement Activity

We will explore in depth the "SQL Statement" activity in BIZUIT, a powerful tool that allows us to connect the platform with SQL Server databases or any database for which we have OLEDB or ODBC drivers, and execute SQL commands directly from our workflows. This integration is key to automating data management, optimizing our processes, and accessing up-to-date information from other systems without the need to duplicate it in BIZUIT.

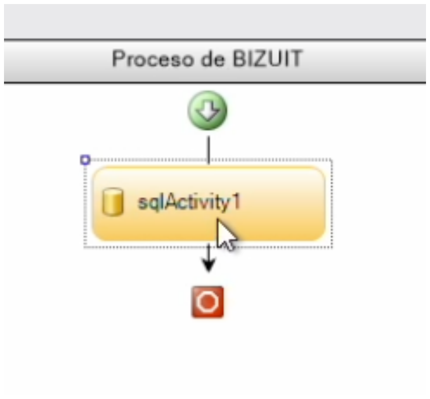
Imagine, for example, a company that manages its inventory in an external SQL database. Thanks to the "SQL Statement" activity, we can configure queries that extract real-time data on the stock of products, which is essential to coordinate orders, update stocks or validate the availability of materials in different processes.

In this unit, we break down each step of the configuration: from database selection and connection, to defining parameters and validating the activity in a test environment. Our goal is to make sure we take full advantage of this functionality in BIZUIT, integrating external data efficiently and reliably.

Now, without further ado, let's start with the configuration of the "SQL Statement" activity in BIZUIT and see how we can transform this tool into an essential ally for our workflow.

Initial Activity Access and Setup

To set up the activity, we first drag it into BIZUIT's process designer. Once placed, we double-click on it to access the configuration wizard. This wizard guides us step by step to set all the necessary properties that will allow us to execute the desired SQL statement.



General Properties of the "SQL Statement" Activity

Within the wizard, in the first step, we configure the General Properties of the activity. These properties include:

- **Activity Name:** To identify it within our workflow.
- **Description:** A text that clarifies the objective of the activity in the flow.

We also have the following options:

- **Option to Preserve Previous Results:**

  When activated, if in the last step of the wizard we map the results of the activity to an XML parameter or variable, the values of those nodes that are not remapped will be kept in the XML. This means that previous data from unmapped nodes is preserved, allowing the XML to maintain a complete structure and preventing overwriting of out-of-date values in that execution.
  Example: If we have an output XML that contains data from several previous runs, this option ensures that nodes not mapped in the last run keep their previous value instead of being deleted or overwritten.

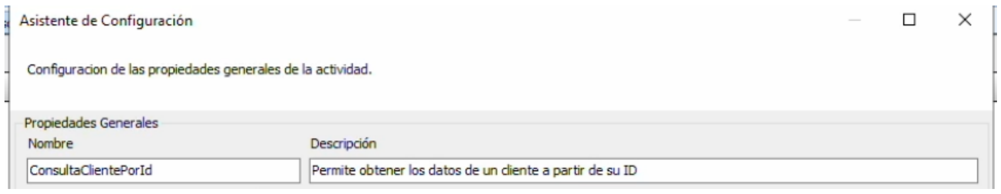
- **Option to Clean Empty Exit Nodes:**

  When you enable this option, the output structure of the activity will not include those XML nodes that are empty, that is, those fields without data. This simplifies the output XML, eliminating unnecessary nodes and reducing both the size and complexity of the data generated.
  Example: If the activity produces a dataset and some fields are empty, enabling this option will cause the nodes corresponding to those fields to not appear in the final XML, keeping only the relevant information and avoiding redundant data.

- **Tracking Entry and Exit Traces:**

  In the Tracking field, we select the level of registration we need for the activity. Options include:
  - **Disabled:** Does not log any traces.
  - **Track Activity Entries:** Saves only the entries received when the activity was executed.
  - **Record trace of activity outputs:** Save only the generated outputs.
  - **Record trace of activity inputs and outputs:** Complete option to have a detailed record of both inputs and outputs, which is useful for reviewing the behavior of the activity.

These general options are common to all the integration activities that we will look at during the chapter


Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales

Nombre	Descripción
ConsultaClientePorId	Permite obtener los datos de un cliente a partir de su ID

Selecting the Connection String Source

In the "String Source" section we specify where BIZUIT will take the connection string to the database from. We have several options:

- **Activity:** We use a connection string specified directly in the configuration of this activity.
- **Configuration File:** Select a previously saved connection string in the general system settings. In this case, a drop-down combo appears with the defined chains, facilitating the reuse of connections and ensuring consistency and security.
- **Pooled Connections:** We use a shared connection pool configured in the process, ideal for scenarios where multiple activities need to access the same database using a single connection that is kept open during flow execution.

Connection String Configuration

In the "Connection String" field, we enter the necessary details to connect to the database, such as the server, the name of the database and the access credentials. BIZUIT offers us an assistant that guides us in each field to ensure that the connection chain is complete and correct.

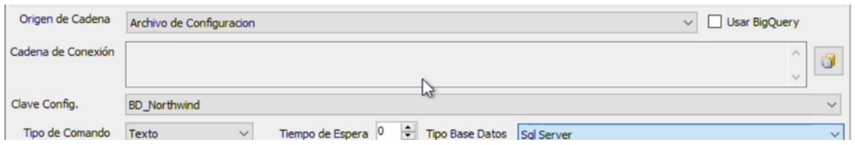
In case of using the "Configuration File" option, select the appropriate key of the combo.

Note on Database Type:

The connection string varies depending on the type of database selected. We can choose between:

- **SQL Server:** For direct connections using the native SQL Server driver, optimized for .NET.
- **OLEDB:** For databases that support connections through OLEDB drivers, allowing us flexibility in integration with different providers.
- **ODBC:** For ODBC-compatible databases, where the connection string must conform to the ODBC driver specifications.

Each type of database requires a specific string; for example, the format for SQL Server differs from that used for OLEDB or ODBC connections, adjusting to the requirements of each driver.



Origen de Cadena: Archivo de Configuracion ☐ Usar BigQuery
Cadena de Conexión:
Clave Config.: BD_Northwind
Tipo de Comando: Texto Tiempo de Espera: 0 Tipo Base Datos: Sql Server

Enabling BigQuery

In the configuration of the "SQL Statement" activity, we observe the option "Use BigQuery" in the String Source section. Checking this box allows us to connect the activity directly to a Google BigQuery database, which is useful for companies that store and query large volumes of data on Google Cloud.

☑ Usar BigQuery

BigQuery-Specific Parameters:

Once the option is enabled, two additional fields are required to authenticate and configure the connection:

- **Project ID:** Enter the Google Cloud project ID that contains the BigQuery database we want to connect to. This field is required to route the activity to the correct project.
- **Credential:** Here we enter the credential required to authenticate access to BigQuery, which can be an API key or a service account configured with the appropriate permissions. The credential must have read or execute permissions in BigQuery, depending on the query we make.

We can also define a Waiting Time (in seconds) for the execution of the sentence. This parameter helps us manage the maximum time that the activity waits for a response before considering that the query has failed due to exceeding the time limit, which is especially useful when working with very large datasets.

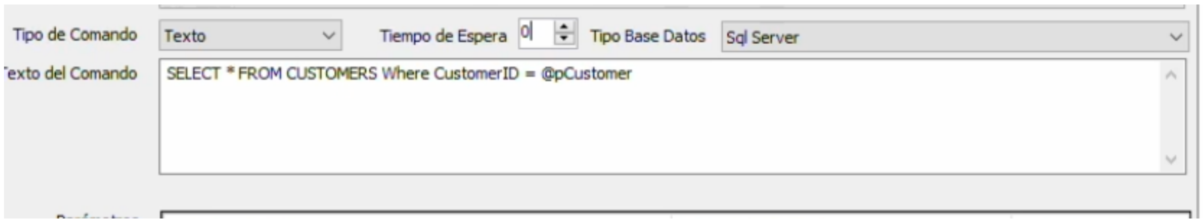
Selecting the Command Type

In the "Command Type" section, select the SQL operation we want to execute:

- **Text:** Allows us to enter SQL statements directly (e.g., SELECT, INSERT, UPDATE, DELETE, EXEC).
- **Stored Procedure:** Runs stored procedures in the database (option available only for SQL Server).

In the "Command Text" field, we type the SQL statement we want to execute. For example, we can enter:

SELECT * FROM Customers WHERE CustomerID = @pCustomer to perform a filtered query based on a specific parameter.



Tipo de Comando	Texto
Tiempo de Espera	0
Tipo Base Datos	Sql Server
'exto del Comando	SELECT * FROM CUSTOMERS Where CustomerID = @pCustomer

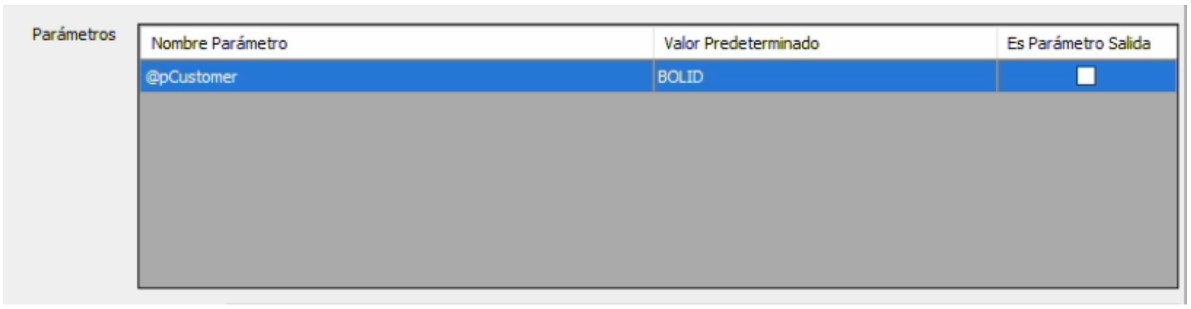
Parameter Definition

The "SQL Statement" activity allows us to define parameters in the query using the "@" symbol to make it dynamic and adaptable to different contexts. These parameters are automatically detected if we include them in the query. In our example, the @pCustomer parameter was automatically added to the parameter grid.

In the Parameters section we configure the following:

- **Parameter Name:** For example, @pCustomer.
- **Default:** This is the initial value that will be used for testing or in the absence of another value during execution.
- **Output Parameter:** By checking this option, we allow the parameter to receive values from the database as a result of execution, returning calculated or processed results. Output parameters are particularly useful when we need the database to return processed information to us that we then reuse in later stages of the workflow.

If in the wizard step, where we perform the input mappings, we do not map the @pCustomer parameter, the activity will use the default value specified in the grid, allowing that value to be used in the execution of the query even if no other value is provided at runtime.



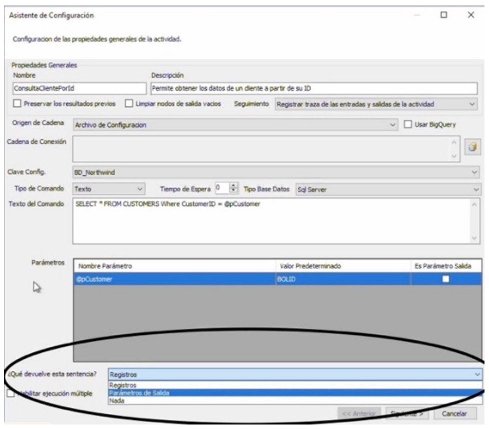
Parámetros

Nombre Parámetro	Valor Predeterminado	Es Parámetro Salida
@pCustomer	BOLID	☐

SQL Statement Return Configuration

In the "What does this statement return?" section, we specify the type of expected result of SQL statement execution. The available options are:

- **Records:** Indicates that the SQL statement will return a set of records, as in a SELECT query. This output can be used in the workflow to feed other activities or processes with data extracted from the database.
- **Output Parameters:** This means that the SQL statement will not return a set of records, but values through the parameters marked as output. This is useful for stored procedures or queries that do not generate tabular data, but specific values to use later.
- **Nothing:** Indicates that the SQL statement will not generate any returns that we need to process or map. It is commonly used for INSERT, UPDATE, DELETE or other statements in which only execution without the need for data return is concerned.


Asistente de Configuración
Configuracion de las propiedades generales de la actividad.
Propiedades Generales

Nombre: ConsultaClientePorId
Descripción: Permite obtener los datos de un cliente a partir de su ID
☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios Seguimiento: Registrar traza de las entradas y salidas de la actividad
Origen de Cadena: Archivo de Configuracion ☐ Usar BigQuery
Cadena de Conexión:
Clave Config.: BD_Northwind
Tipo de Comando: Texto Tiempo de Espera: 0 Tipo Base Datos: Sql Server
Texto del Comando:

```
SELECT * FROM CUSTOMERS Where CustomerID = @pCustomer
```

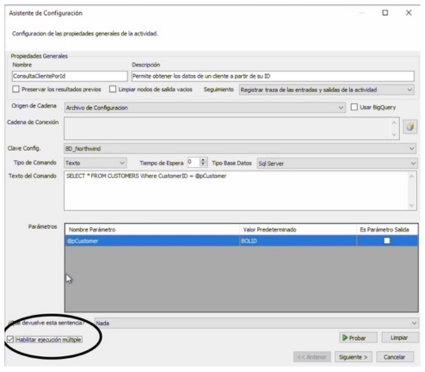
Parámetros

Nombre Parámetro	Valor Predeterminado	Es Parámetro Salida
@pCustomer	BOLID	☐

¿Qué devuelve esta sentencia? Registros
Registros
Parámetros de Salida
Nada
☐ Habilitar ejecución múltiple

Enable Multi-Run

The "Enable Multiple Execution" option allows the SQL statement or stored procedure to run multiple times in a loop. This is necessary when we need to process multiple inputs or perform the same operation iteratively on a dataset within the activity.


Asistente de Configuración
Configuracion de las propiedades generales de la actividad.
Propiedades Generales

Nombre: ConsultaClientePorId
Descripción: Permite obtener los datos de un cliente a partir de su ID
☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios Seguimiento: Registrar traza de las entradas y salidas de la actividad
Origen de Cadena: Archivo de Configuracion ☐ Usar BigQuery
Cadena de Conexión:
Clave Config.: BD_Northwind
Tipo de Comando: Texto Tiempo de Espera: 0 Tipo Base Datos: Sql Server
Texto del Comando:

```
SELECT * FROM CUSTOMERS Where CustomerID = @pCustomer
```

Parámetros

Nombre Parámetro	Valor Predeterminado	Es Parámetro Salida
@pCustomer	BOLID	☐

¿Qué devuelve esta sentencia? Nada
☑ Habilitar ejecución múltiple

Activity Validation and Testing

Before deploying, it is critical to validate the configuration of the activity. The "SQL Statement" activity in BIZUIT includes a "Test" button that allows us to run the query in a test environment. This functionality is key as it not only verifies that the database connection and query are running correctly, but also helps us infer the output structure of the activity.

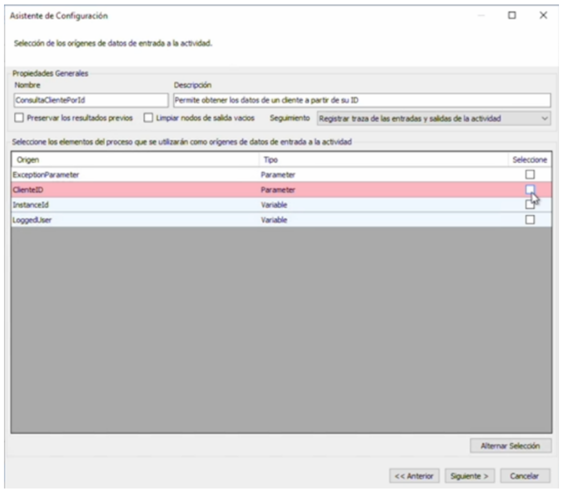
During the test, we check:

- **Connection:** That the activity successfully connects to the database specified in the connection string.
- **Execution of the Statement:** That the SQL query or the stored procedure is executed without errors.
- **Data Structure and Accuracy:** That the data obtained is accurate and meets our expectations.

Selecting Data Sources for Parameters and Next Step

Once the configuration is validated, we click "Next" to advance to the next step of the configuration wizard. In this stage, we select the Data Sources from which we will obtain the necessary values for the parameters of the SQL statement. These values can come from:

- **Process parameters:** Values defined at the process level.
- **Process variables:** Variables calculated or assigned in the workflow.
- **Result of execution of previous activities:** Values generated by other previous activities in the flow.
- **System parameters:** Options such as InstanceId, LoggedUser, Exception, which allow us to use system information in the query.


Asistente de Configuración

Selección de los orígenes de datos de entrada a la actividad.

Propiedades Generales

Nombre: ConsultaClientePorId

Descripción: Permite obtener los datos de un cliente a partir de su ID

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacíos Seguimiento: Registrar traza de las entradas y salidas de la actividad

Seleccione los elementos del proceso que se utilizarán como orígenes de datos de entrada a la actividad

Origen	Tipo	Seleccione
ExceptionParameter	Parameter	☐
ClienteID	Parameter	☐
InstanceId	Variable	☐
LoggedUser	Variable	☐

This flexibility in data sources for parameters ensures that the query dynamically adapts to the values generated or assigned in the workflow, increasing the accuracy and customization of the activity.

Setting up the "SQL Statement" activity in BIZUIT is a crucial step in achieving effective database integration. With this configuration, we can execute SQL commands in an automated and personalized way, allowing us to perform real-time queries, update data, and much more. This integration optimizes the interaction between BIZUIT and other enterprise data systems, improving efficiency in our workflows and allowing us to access up-to-date and accurate information directly from BIZUIT.

Configuring REST Activity

The REST activity in BIZUIT is a powerful and flexible tool that allows us to make calls to external APIs directly from our workflows. Its operation is similar to tools such as Postman, but fully integrated into the platform, which makes it easy to consult and update data in real time without the need to store or duplicate information.

For example, let's say we need to check if an employee is enabled to receive reimbursements through a corporate API. With REST activity, we can set up a request that sends your ID and receives an immediate response, allowing us to make automated decisions, such as approving or rejecting the refund request.

In this unit, we'll explore every aspect of your configuration, from HTTP method selection to authentication, header usage, request body, and response validation. Our goal is to effectively integrate external data into our workflows, optimizing our processes through the flexibility that this activity offers.

REST Services Concept

REST services enable communication between systems using HTTP requests, using methods such as GET, POST, PUT, and DELETE. This model has become the standard for integration due to its simplicity, efficiency, and flexibility. Thanks to this technology, our applications can exchange data in real time, which is crucial for operations such as price inquiries, order submission, or payment status verification.

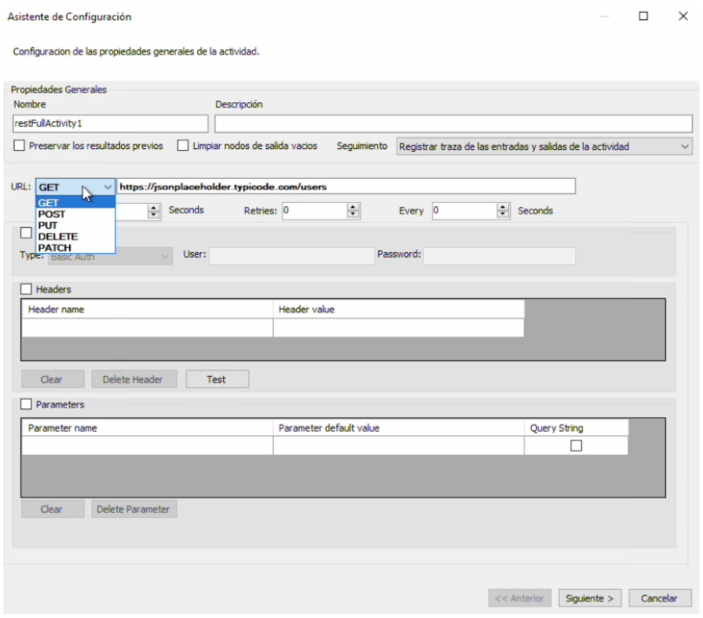
BIZUIT allows us to consume REST services in a simple and efficient way, connecting with a wide variety of external APIs through its REST activity.

URL and HTTP Method

The first step in configuration is to define the URL and HTTP method of the request:

- GET: Gets data from the server.
- POST: Sends data to create a resource.
- PUT/PATCH: Updates an existing resource.
- DELETE: Deletes a resource from the server.

In addition, we set a timeout to define the maximum timeout and set up automatic retries in case of failures.



Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales

Nombre: restFullActivity1

Descripción:

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios

Seguimiento: Registrar traza de las entradas y salidas de la actividad

URL: GET https://jsonplaceholder.typicode.com/users

GET / POST / PUT / DELETE / PATCH

Seconds Retries: 0 Every 0 Seconds

☐

Type: Basic Auth User: Password:

☐ Headers

Header name	Header value

Clear / Delete Header / Test

☐ Parameters

Parameter name	Parameter default value	Query String
		☐

Clear / Delete Parameter

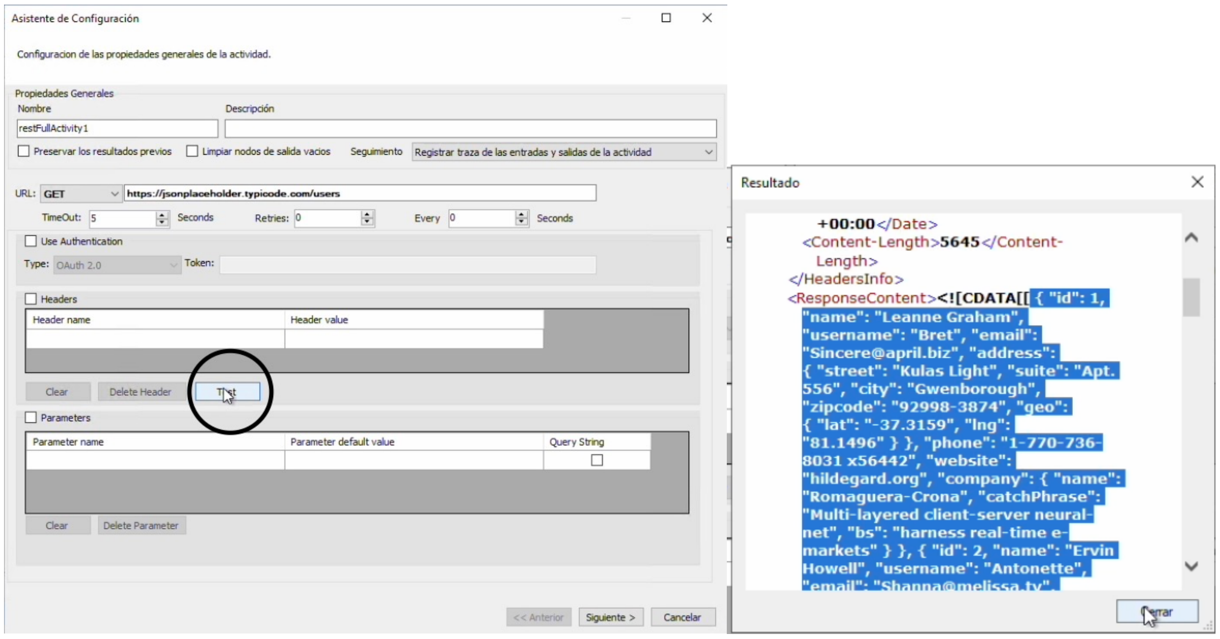
<< Anterior / Siguiente > / Cancelar

Authentication

Many APIs require authentication to access your data. BIZUIT allows us to configure:

- **Basic Auth**: Username and password hardcoded in the request header.
- **OAuth 2.0**: Using Access Tokens for Secure Authentication.

We can test the connection with the API by pressing the Test button, verifying that the response is as expected.


Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales

Nombre	restFullActivity1
Descripción	
Preservar los resultados previos	☐
Limpiar nodos de salida vacios	☐
Seguimiento	Registrar traza de las entradas y salidas de la actividad
URL	GET https://jsonplaceholder.typicode.com/users
TimeOut	5 Seconds
Retries	0
Every	0 Seconds
Use Authentication	☐
Type	OAuth 2.0
Token	
Headers	☐
Parameters	☐

Header name	Header value

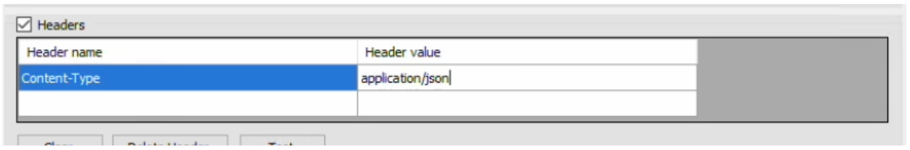
Parameter name	Parameter default value	Query String
		☐

```
    +00:00</Date>
  <Content-Length>5645</Content-
    Length>
</HeadersInfo>
<ResponseContent><![CDATA[[ { "id": 1,
  "name": "Leanne Graham",
  "username": "Bret", "email":
  "Sincere@april.biz", "address":
  { "street": "Kulas Light", "suite": "Apt.
  556", "city": "Gwenborough",
  "zipcode": "92998-3874", "geo":
  { "lat": "-37.3159", "lng":
  "81.1496" } }, "phone": "1-770-736-
  8031 x56442", "website":
  "hildegard.org", "company": { "name":
  "Romaguera-Crona", "catchPhrase":
  "Multi-layered client-server neural-
  net", "bs": "harness real-time e-
  markets" } }, { "id": 2, "name": "Ervin
  Howell", "username": "Antonette",
  "email": "Shanna@melissa.tv",
```

Headers Configuration

HTTP headers allow you to send additional information to the API. For example:

- Content-Type: application/json to indicate the format of the data sent.
- Authorization: Bearer <token> if the API requires an authentication token.


☑ Headers

Header name	Header value
Content-Type	application/json

Inclusion of Parameters

Parameters can be passed in two ways:

- **Query String**: For GET requests, they are added to the URL (example: ?id=123).
- **Request Body**: For POST and PUT, they are sent within the body in JSON or XML formats.

BIZUIT allows you to define default values for parameters, ensuring that the request works even if a value is not provided at runtime.

Body Configuration

For POST and PUT, you need to submit data in the request body. Supported formats include:

- **application/json**: Data in JSON format.
- **application/xml**: Data in XML format.
- **multipart/form-data**: Use of files and forms.

We can parameterize values within the body, allowing content to be dynamically generated based on workflow inputs.

Configuration Validation and Testing

Before finishing, we test the configuration using the Test button. During this validation, we check:

- That the authentication works correctly.
- That the headers and the body are accepted by the server.
- That the data in the response match what is expected.

Once validated, we can continue to configure the data sources so that the values used come from process variables, results of previous activities, or system parameters.

The REST activity in BIZUIT allows us to make calls to external APIs in an efficient and automated way, without the need to resort to external tools such as Postman. Its configuration gives us the possibility of connecting to a wide variety of external services, optimizing our workflows by consulting and updating data in real time.

Thanks to this integration, we can improve decision-making in our processes and ensure that each system involved receives the information at the right time. In the next unit, we'll explore how we can map and transform the insights gained through these integrations, ensuring that the data is tailored to our operational needs.

Web Services Activity Settings

Now, we'll explain how to configure an integration activity in BIZUIT that consumes a SOAP service. We'll cover key SOAP concepts, detailed setup steps, and best practices for testing and validation.

Introduction

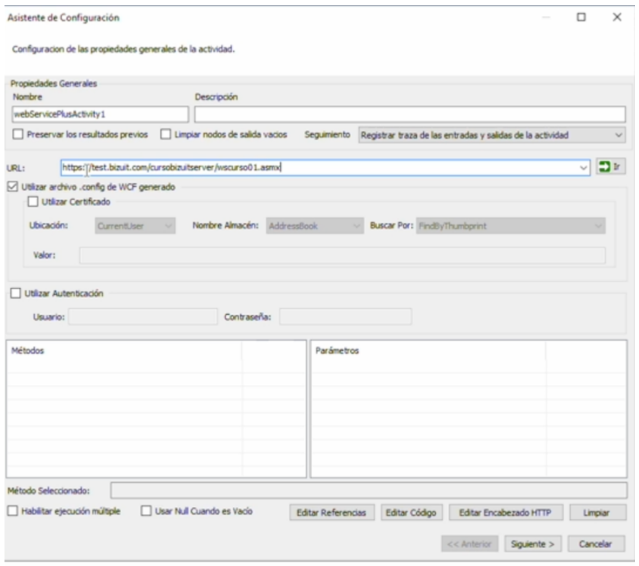
The Web Services activity in BIZUIT allows us to invoke web services via the SOAP protocol. To do this, it uses the .NET framework to generate a proxy, which facilitates the automatic identification of the methods and data types of the service to which we connect. This is essential when we need to integrate BIZUIT with other systems that expose SOAP services, allowing us to access and manipulate data in real time.

Let's say we want to query a web service that handles employee information. If we need to verify if an employee is eligible to receive reimbursements, this activity allows us to submit their ID number as a parameter and get a response that we can then use within our workflows.

Below, we'll detail each of the steps required to set up this activity in BIZUIT, from defining the service URL to selecting the authentication method and settings.

Initial activity setup

The first thing we do is add the Web Service activity to the workflow in BIZUIT and access its settings. On the initial screen, we need to enter the URL of the web service, which should point to the WSDL file. This file describes the available methods and the types of data we can use. In our case, we'll work with a test URL that lays out a set of methods for employee management.


Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales

Nombre: webServicePlusActivity1

Descripción:

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios

Seguimiento: Registrar traza de las entradas y salidas de la actividad

URL: https://test.bizuit.com/cursobizuitserver/wscurso01.asmx

☑ Utilizar archivo .config de WCF generado

☐ Utilizar Certificado

Ubicación: CurrentUser Nombre Almacén: AddressBook Buscar Por: FindByThumbprint

Valor:

☐ Utilizar Autenticación

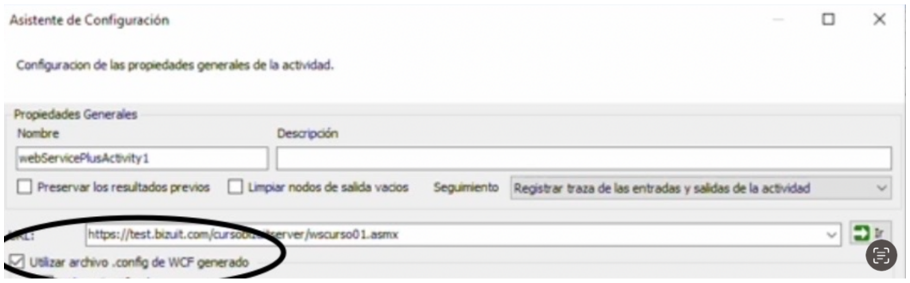
Usuario: Contraseña:

Método Seleccionado:

☐ Habilitar ejecución múltiple ☐ Usar Null Cuando es Vacío

Using the WCF Configuration File

The activity allows us to use a previously generated WCF configuration file. Activating this option is useful if you already have a prepared file, as it prevents you from manually configuring parameters such as waiting times or security. If we do not enable this option, BIZUIT will try to use its default settings, which will need to be manually edited if specific settings are required.


Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales

Nombre: webServicePlusActivity1

Descripción:

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios Seguimiento: Registrar traza de las entradas y salidas de la actividad

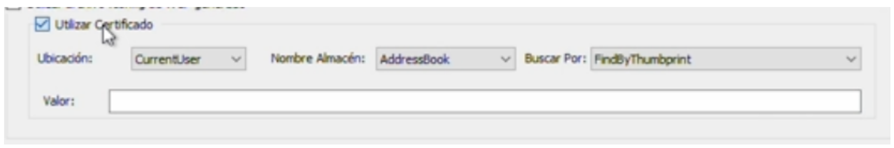
URL: https://test.bizuit.com/curso[illegible]server/wscurso01.asmx

☑ Utilizar archivo .config de WCF generado

Configuring Certificates for Secure Authentication

If the service requires authentication by means of a digital certificate, we can enable the corresponding option and configure the following parameters:

- **Location:** We define whether the certificate is in *CurrentUser* or *LocalMachine*.
- **Store Name:** We choose the certificate container, such as *AddressBook* or *My*.
- **Search by:** We specify the search criteria, such as *FindByThumbprint* to locate it by fingerprint.
- **Value:** We enter the fingerprint or certificate identifier.


☑ Utilizar Certificado

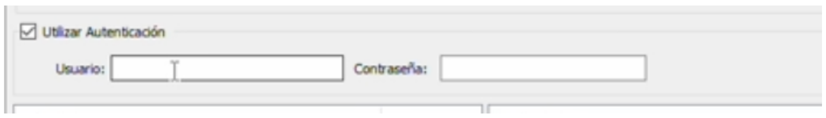
Ubicación: CurrentUser Nombre Almacén: AddressBook Buscar Por: FindByThumbprint

Valor:

This configuration is essential when working with services that require secure connections or certificate-based authentication.

User authentication

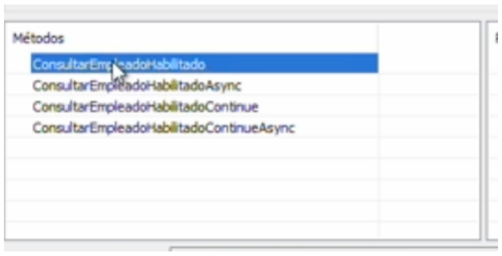
If the service requires authentication with username and password, we activate the Use Authentication option and enter the necessary credentials. This type of authentication is common in restricted services, where only authorized users can access data.


☑ Utilizar Autenticación

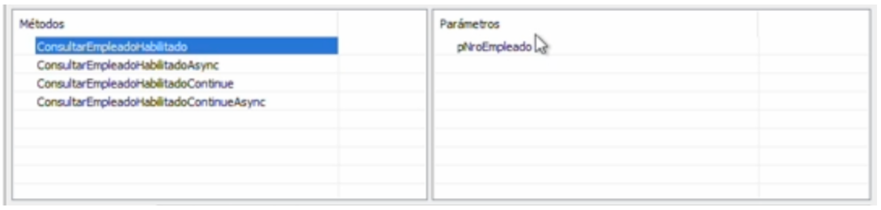
Usuario: Contraseña:

Selecting Methods and Configuring Parameters

When we connect to the service, BIZUIT generates a proxy and shows us the list of available methods. In this case, we want to use the QueryEmployeeEnabled method, which allows us to check if an employee can receive certain benefits.


Métodos
ConsultarEmpleadoHabilitado
ConsultarEmpleadoHabilitadoAsync
ConsultarEmpleadoHabilitadoContinue
ConsultarEmpleadoHabilitadoContinueAsync

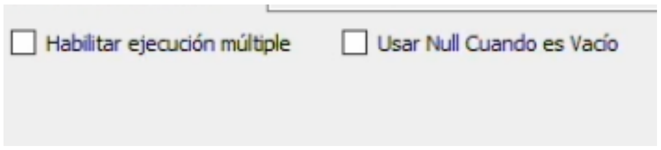
When you select a method, the parameter section is automatically populated with the required values. In this example, the method needs the pNroEmployee parameter, which represents the number of employees to be queried. To set it up, we simply map it to a workflow value in a later step.


Métodos
ConsultarEmpleadoHabilitado
ConsultarEmpleadoHabilitadoAsync
ConsultarEmpleadoHabilitadoContinue
ConsultarEmpleadoHabilitadoContinueAsync
Parámetros
pNroEmpleado

Advanced Configuration Options

BIZUIT offers several advanced options to adapt the activity to our needs:

- **Enable multiple execution:** Allows the activity to be executed multiple times in sequence, useful when we need to process multiple requests.
- **Use Null when empty:** In case a parameter doesn't have a value, it will be sent as *null* instead of an empty string, which is important if the web service requires explicit null values.



Configuring HTTP headers and custom code

In the advanced settings section, we can also modify the HTTP headers and proxy code:

- **Edit HTTP Headers:** Allows us to add specific values in the request, such as *Content-Type* or *authentication tokens*.
- **Edit references and code:** If necessary, we can customize the proxy code or add additional references to improve communication with the web service.

These options are useful in more complex integrations where we need specific configurations to ensure the correct functioning of the activity.

Practical example

Let's say we need to verify if an employee is eligible to receive reimbursements. To do this, we select the QueryEmployeeEnabled method and move forward in the configuration wizard.

At this stage, we need to define the source of the data that will be sent as parameters to the web service. BIZUIT allows us to obtain these values from:

- **Process parameters:** Values previously defined in the workflow.
- **Process variables:** Variables calculated or assigned in the flow.
- **Results of execution of previous activities:** Data generated by other activities in the flow.
- **System parameters:** Information such as InstanceId, LoggedUser, or Exception.

In our case, we map the NoEmployeeParameter of the process to the pNoEmployeeParameter of the selected method. Once the query is made, the response will be stored in the response node, and we can map it to another variable or parameter, such as vRespuestaWS, for later use in the workflow.

The Web Service activity in BIZUIT is a key tool for integration with SOAP web services. It allows us to leverage methods and data from other systems within our workflows, making it easier to automate complex processes.

From configuring the URL and certificates to selecting methods and parameters, this activity simplifies the consumption of external services and gives us the flexibility to tailor each integration to our needs.

If we want to dig deeper into integration with other protocols, the next step is to explore how to set up an integration activity using the TCP protocol to establish real-time connections with external systems.

TCP Activity Settings

In this section, we're going to explore together TCP activity in BIZUIT, a tool that allows us to connect to TCP servers to send and receive messages directly. This activity is essential when we need to integrate BIZUIT with systems or devices that communicate using the TCP protocol, such as industrial management systems, medical devices or real-time data servers.

By setting up this activity, we are able to send specific messages or retrieve information from external systems, ensuring a constant and reliable flow of data.

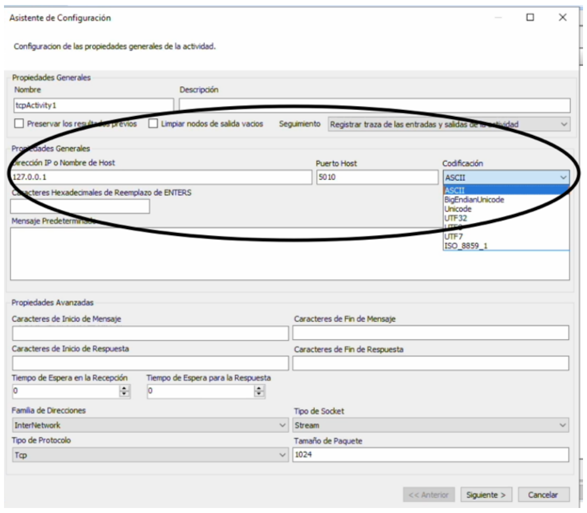
Today we're going to learn how to configure each parameter of TCP activity: from connection details and the default message, to advanced message and response control options. Thanks to this configuration, we will be able to optimize the integration of BIZUIT with TCP servers, achieving efficient communication adapted to our needs.

To get started, we add TCP activity to our workflow in BIZUIT and access its settings. On the home screen, you'll see all the available options.

General Properties

We start with the General Properties, where we set up the basic connection data:

- **IP Address or Hostname**: Here we enter the address of the TCP server we want to connect to. It is important that you point to the exact server that offers the desired service.
- **Host Port**: We specify the server port. We need to make sure that it is enabled and the correct one to establish the connection.
- **Encoding**: We select the type of encoding that will be used for the messages, such as ASCII or UTF-8, depending on the format that our server handles.


Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales

Nombre: tcpActivity1

Descripción:

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios

Seguimiento: Registrar traza de las entradas y salidas de la actividad

Propiedades Generales

Dirección IP o Nombre de Host: 127.0.0.1

Puerto Host: 5010

Codificación: ASCII

Caracteres Hexadecimales de Reemplazo de ENTERS:

Mensaje Predeterminado:

Propiedades Avanzadas

Caracteres de Inicio de Mensaje:

Caracteres de Fin de Mensaje:

Caracteres de Inicio de Respuesta:

Caracteres de Fin de Respuesta:

Tiempo de Espera en la Recepción: 0

Tiempo de Espera para la Respuesta: 0

Familia de Direcciones: InterNetwork

Tipo de Socket: Stream

Tipo de Protocolo: Tcp

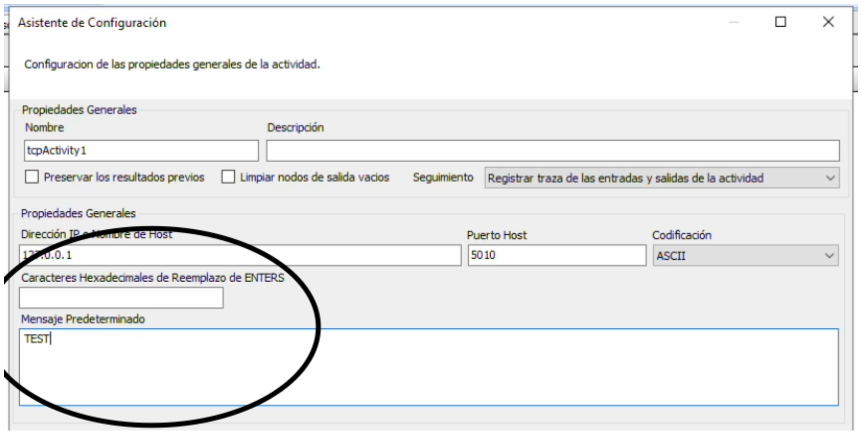
Tamaño de Paquete: 1024

These fields are critical to ensure a successful connection and consistent communication with the format expected by the server.

Message Settings

In this section, we define how sending messages will be handled:

- **Hexadecimal ENTERS Replacement Characters**: We indicate the characters that will replace the ENTER (carriage return), when the server requires it.
- **Default Message**: Here we can enter a message that will be sent to the server in case there is no data mapped in the activity entry.


Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales
Nombre: tcpActivity1
Descripción:
Preservar los resultados previos
Limpiar nodos de salida vacios
Seguimiento: Registrar traza de las entradas y salidas de la actividad

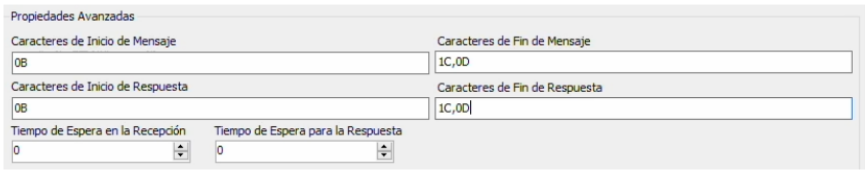
Propiedades Generales
Dirección IP o Nombre de Host: 127.0.0.1
Puerto Host: 5010
Codificación: ASCII
Caracteres Hexadecimales de Reemplazo de ENTERS:
Mensaje Predeterminado: TEST

Thanks to these settings, we adapt the messages to the expected format, ensuring correct interpretation by the server.

Advanced Message Properties

In Advanced Properties, we set up additional details for messages and their replies:

- **Message Start and End Characters**: We specify the hexadecimal characters that delimit each sent message, making it easier for the server to interpret it.
- **Response Start and End Characters**: We define the characters that mark the beginning and end of the expected response. This is especially useful if the server responds in multiple parts or with long messages.


Propiedades Avanzadas
Caracteres de Inicio de Mensaje: 0B
Caracteres de Fin de Mensaje: 1C,0D
Caracteres de Inicio de Respuesta: 0B
Caracteres de Fin de Respuesta: 1C,0D
Tiempo de Espera en la Recepción: 0
Tiempo de Espera para la Respuesta: 0

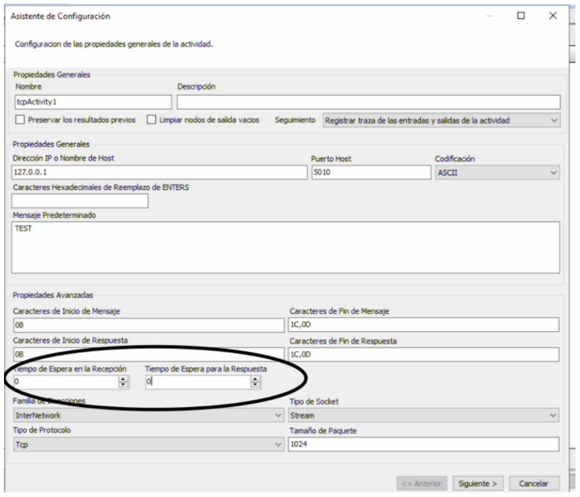
These settings allow us to customize the behavior of end-to-end communication.

Configuring Timeouts

Now we move on to the parameters related to waiting times:

- **Wait Time at Reception**: Defines how long we wait for a message before canceling the connection.

- **Response Timeout**: Determines how long we wait for a response after sending a message. If the server takes too long, the activity will throw an error.


Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales
Nombre: tcpActivity1
Descripción:
☐ Preservar los resultados previos
☐ Limpiar nodos de salida vacios
Seguimiento: Registrar traza de las entradas y salidas de la actividad

Propiedades Generales
Dirección IP o Nombre de Host: 127.0.0.1
Puerto Host: 5010
Codificación: ASCII
Caracteres Hexadecimales de Reemplazo de ENTERS:
Mensaje Predeterminado: TEST

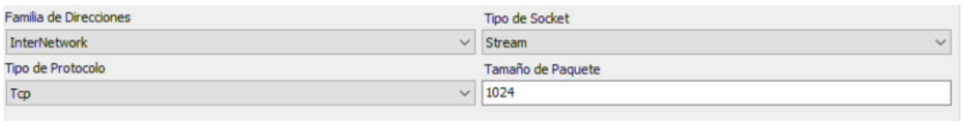
Propiedades Avanzadas
Caracteres de Inicio de Mensaje: 0B
Caracteres de Fin de Mensaje: 1C,0D
Caracteres de Inicio de Respuesta: 0B
Caracteres de Fin de Respuesta: 1C,0D
Tiempo de Espera en la Recepción: 0
Tiempo de Espera para la Respuesta: 0
Familia de Direcciones: InterNetwork
Tipo de Socket: Stream
Tipo de Protocolo: Tcp
Tamaño de Paquete: 1024

Both times allow us to correctly handle possible delays in communication and maintain the stability of the flow.

Connection and Protocol Settings

In this section, we define technical aspects of the TCP protocol:

- **Address Family**: Select the IP version to be used (e.g., InterNetwork for IPv4).
- **Socket Type**: We choose between different types, with Stream being the one indicated for a flow communication, typical of the TCP protocol.
- **Protocol Type**: We confirm that it is TCP, guaranteeing a reliable connection.
- **Packet Size**: We indicate the maximum size of the data per packet, which helps optimize the transfer.


Familia de Direcciones: InterNetwork
Tipo de Socket: Stream
Tipo de Protocolo: Tcp
Tamaño de Paquete: 1024

With these options, we adapt the TCP connection to the specific requirements of the environment in which we are working.

Practical Example

Let's look at a specific case. Suppose we need to send an HL7 message to a RIS system that receives data over TCP.

- We configure the IP address and the Host Port of the RIS system.
- Enter the Default Message.
- We define the Start and End Characters of the message.

Then, we move on to the next step of the configuration wizard, where we select the Data Sources to get the necessary values. We can use:

- Process Parameters
- Process variables
- Results of previous activities
- System parameters (such as InstanceId or LoggedUser)

In our case, we select the Message process parameter and map it to the activity's Message parameter. As a result, we will get the response from the TCP server on the outgoing Message node, which we can then map to another variable or parameter, such as Response.

BIZUIT's TCP activity is a powerful tool to connect directly with servers that use this protocol. Thanks to its detailed configuration, we can integrate real-time data from external systems or devices, increasing the efficiency and automation of our processes.

We hope this demo has been helpful in understanding how to set up and use TCP activity in BIZUIT.

FTP Activity Settings

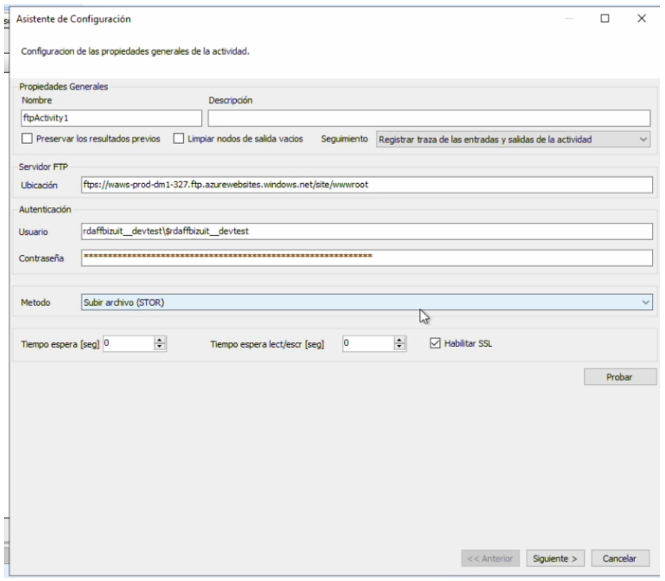
Now we'll look at how to set up an integration activity in BIZUIT that allows us to connect to FTP servers for file transfer. This functionality is essential when we need to automate tasks related to file management, such as uploading and downloading documents, creating or deleting directories, among other key operations.

Imagine, for example, a company that must store inventory reports daily on an FTP server. Thanks to the FTP activity in BIZUIT, we can automate this task in a simple and efficient way. Throughout this chapter, we will walk through step by step how to configure this activity to perform various operations on FTP servers and thus optimize our workflows.

Connecting to the FTP Server

We start by adding FTP activity to the workflow in BIZUIT. Once incorporated, we access its settings. The first thing we must do is establish the connection with the server. We enter the FTP server address in the corresponding field and then fill in the authentication data: username and password.

If the server requires a secure connection, we enable the SSL option to ensure that the data is transmitted in an encrypted and protected manner.



Configuracion de las propiedades generales de la actividad.

Propiedades Generales

Nombre: ftpActivity1

Descripción:

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios

Seguimiento: Registrar traza de las entradas y salidas de la actividad

Servidor FTP

Ubicación: ftps://waws-prod-dm1-327.ftp.azurewebsites.windows.net/site/wwwroot

Autenticación

Usuario: rdaffbizuit__devtest\$rdaffbizuit__devtest

Contraseña: **

Metodo: Subir archivo (STOR)

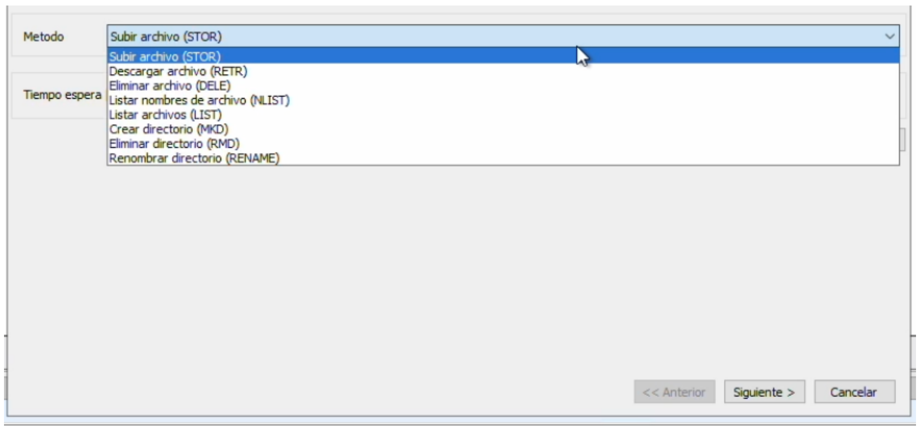
Tiempo espera [seg]: 0 Tiempo espera lect/escr [seg]: 0 ☑ Habilitar SSL

Selection of the Method of Operation

Once connected, we define what operation we want to perform on the server. In the "Method" field, BIZUIT offers us the following options:

- **File Upload (STOR):** Allows you to upload a file from BIZUIT to the FTP server.
- **Download File (RETR):** Download a file from the server to our workflow.
- **Delete File (DELE):** Deletes a specific file on the server.

- **List File Names (NLIST):** Gets a list of file names in a directory.
- **List Files:** Similar to the previous one, but returns more detailed information.
- **Create Directory (MKD):** Create a new directory in the location we indicate.
- **Delete Directory (RMD):** Deletes an existing directory.
- **RENAME:** Renames a file or directory.



Metodo: Subir archivo (STOR)

Tiempo espera

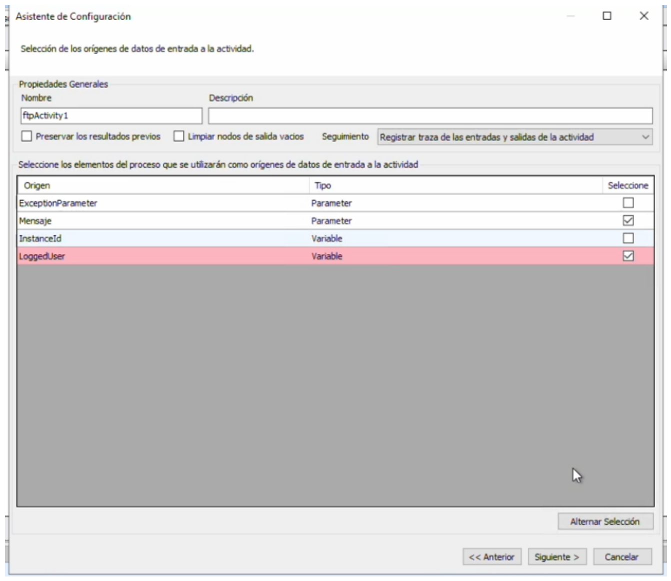
- Subir archivo (STOR)
- Descargar archivo (RETR)
- Eliminar archivo (DELE)
- Listar nombres de archivo (NLIST)
- Listar archivos (LIST)
- Crear directorio (MKD)
- Eliminar directorio (RMD)
- Renombrar directorio (RENAME)

Each method is intended for a specific operation and has different configuration and mapping options that we will explore in detail later.

Practical Examples

If we select the Rename Directory method, when we continue, BIZUIT will ask us for two essential parameters: the current name and the new name of the directory. These values can be defined based on flow variables, which allows us to streamline the operation according to the needs of the process.

For example, we might want the "Daily Reports" directory to be renamed "MonthlyReports" at the end of each month. To achieve this, we simply assign those values in the parameter mapping. In this way, we automate the operation without the need for manual intervention.



Asistente de Configuración

Selección de los orígenes de datos de entrada a la actividad.

Propiedades Generales

Nombre: ftpActivity1

Descripción:

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios Seguimiento: Registrar traza de las entradas y salidas de la actividad

Seleccione los elementos del proceso que se utilizarán como orígenes de datos de entrada a la actividad

Origen	Tipo	Seleccione
ExceptionParameter	Parameter	☐
Mensaje	Parameter	☑
InstanceId	Variable	☐
LoggedUser	Variable	☑

On the other hand, if we choose List Files, the configuration is much simpler. We don't need to specify a particular file; Just define the directory and the activity will return a list of all the files it contains. We can map the result to an XML-type variable—in our case, we'll use the Response output parameter—and then process it into the rest of the flow.

Waiting Time and Security

BIZUIT also allows us to configure advanced options, such as timeouts for the connection and for read or write operations. These configurations are useful in scenarios where the server may be slow to respond, thus preventing activity from failing due to a simple delay.

In addition, as we mentioned before, we can enable the use of SSL to ensure secure connections with servers that require it.

Validation and Testing

Before you finalize the setup, it's important to test that everything is working properly. To do this, we have the Test button, which allows us to verify both the connection to the server and the parameters defined for the selected operation.

When running the test, we observe the input and output of the activity. We confirm that the Response output parameter contains the information returned by the server in XML format. In the case of listing files, we can see the complete list of the contents of the specified directory.

The FTP activity in BIZUIT provides us with a versatile and powerful solution to automate tasks that require interaction with FTP servers. Whether it's transferring files, organizing directories, or simply getting listings, this tool integrates seamlessly into our workflows, eliminating the need to perform these tasks manually.

Configuration of Activity Component Library.

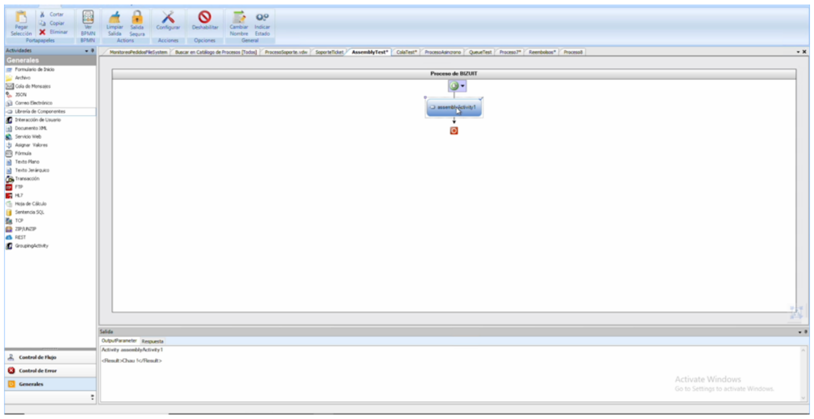
In this section, we will learn how to configure integration activities in BIZUIT using .NET assemblies and COM components. Thanks to this functionality, we will be able to achieve advanced integration with systems and libraries that operate on the Windows platform, reusing existing logic and extending our capabilities within the workflow.

The Component Library activity allows us to invoke *stateless* methods of defined classes in external .NET or COM libraries. This capability gives us a very flexible and powerful way to interact with external components, expanding the functional scope of our processes.

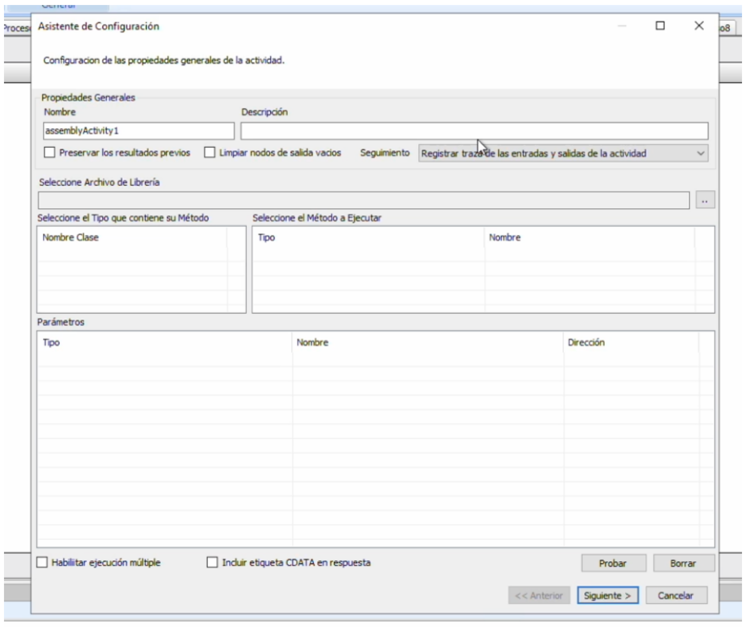
Thanks to this activity, we will be able to incorporate logics already developed in other applications, avoiding code duplication and optimizing the overall design of the workflow. In this chapter, we'll walk through step-by-step how to set up this activity, from selecting the library to running and testing the included methods.

Adding the Activity to the Workflow

To get started, we incorporated the Component Library activity into the flow. Then we access your settings. On the initial screen we find several sections that we will explore in detail.



```
Activity assemblyActivity1
<Result>Chau !</Result>
```



Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales

Nombre	Descripción
assemblyActivity1	

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios Seguimiento: Registrar traza de las entradas y salidas de la actividad

Seleccione Archivo de Librería

Seleccione el Tipo que contiene su Método

Nombre Clase

Seleccione el Método a Ejecutar

Tipo	Nombre

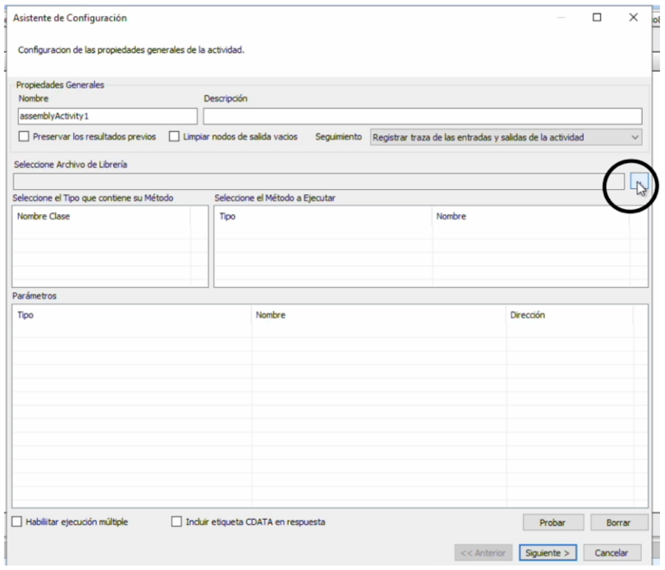
Parámetros

Tipo	Nombre	Dirección

☐ Habilitar ejecución múltiple ☐ Incluir etiqueta CDATA en respuesta

Selection of the Library Archive

The first step is to select the library file that contains the classes and methods we want to use. At the top of the settings screen, we find the field to upload the .dll file.



Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales

Nombre: assemblyActivity1

Descripción:

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacíos Seguimiento: Registrar traza de las entradas y salidas de la actividad

Seleccione Archivo de Librería

Seleccione el Tipo que contiene su Método

Nombre Clase

Seleccione el Método a Ejecutar

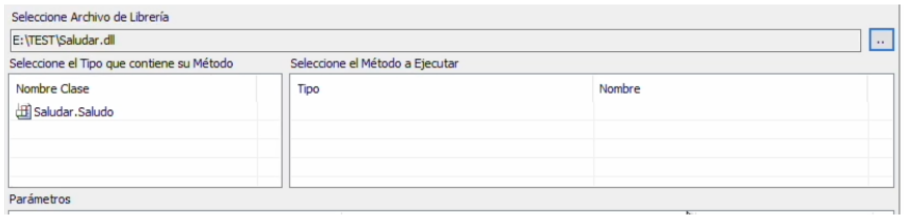
Tipo	Nombre

Parámetros

Tipo	Nombre	Dirección

☐ Habilitar ejecución múltiple ☐ Incluir etiqueta CDATA en respuesta

We press the browse button and select the corresponding file from the file system of the BIZUIT server. For example, we can use the Saludar.dll library, which contains simple methods designed for this demonstration.



Seleccione Archivo de Librería

E:\TEST\Saludar.dll

Seleccione el Tipo que contiene su Método

Nombre Clase
Saludar.Saludo

Seleccione el Método a Ejecutar

Tipo	Nombre

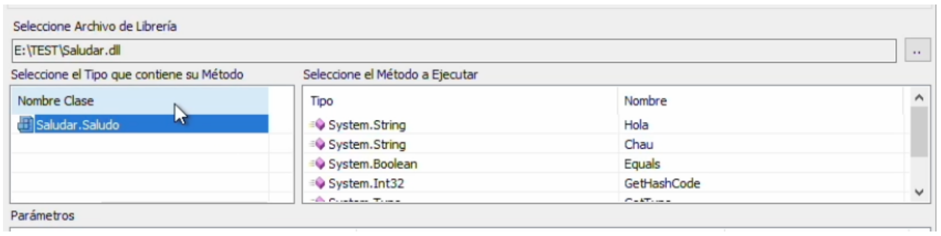
Parámetros

Important: The path and file must exist on the BIZUIT server, as runtime execution is performed in that environment.

Class and Method Selection

With the library loaded, we move on to defining which class and which method we are going to use. We find two key fields:

- **Class Name**: Select the public class that contains the desired method.



Seleccione Archivo de Librería

E:\TEST\Saludar.dll

Seleccione el Tipo que contiene su Método

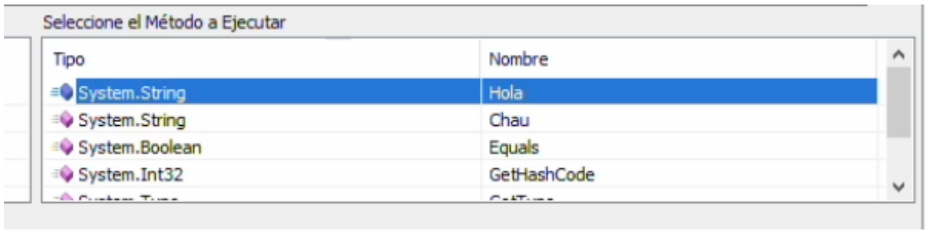
Nombre Clase
Saludar.Saludo

Seleccione el Método a Ejecutar

Tipo	Nombre
System.String	Hola
System.String	Chau
System.Boolean	Equals
System.Int32	GetHashCode

Parámetros

- **Method to Execute**: We choose the method we are going to invoke. Remember that methods must be *stateless*, that is, they must not depend on a shared or persistent state between executions.



Seleccione el Método a Ejecutar

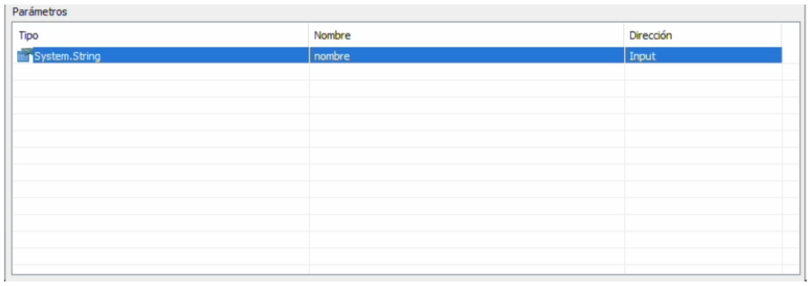
Tipo	Nombre
System.String	Hola
System.String	Chau
System.Boolean	Equals
System.Int32	GetHashCode

Parameter Configuration

The parameters section allows us to define the values that the method requires. For each parameter we visualize:

- **Type**: The expected data type (for example, System.String, System.Int32).

- **Name**: The exact name of the parameter, as defined in the method.
- **Direction**: We indicate whether the parameter is Input or Output.



Parámetros

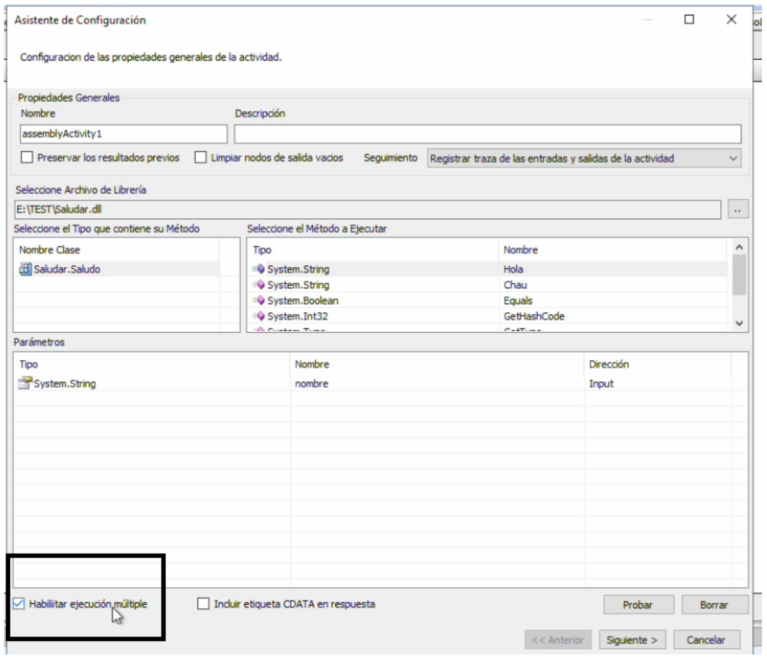
Tipo	Nombre	Dirección
System.String	nombre	Input

For example, if the Hola method receives a parameter name of type System.String, we must indicate the value that will be sent when invoking that method.

Advanced Options

At the bottom of the screen we find additional options that can be useful:

- **Enable Multiple Execution**: Allows the method to be executed repeatedly for a list of elements, generating repetitive nodes in the flow. Ideal for working with collections or records.


Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales

Nombre: assemblyActivity1

Descripción:

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios Seguimiento: Registrar traza de las entradas y salidas de la actividad

Seleccione Archivo de Librería: E:\TEST\Saludar.dll

Seleccione el Tipo que contiene su Método

Nombre Clase
Saludar.Saludo

Seleccione el Método a Ejecutar

Tipo	Nombre
System.String	Hola
System.String	Chau
System.Boolean	Equals
System.Int32	GetHashCode

Parámetros

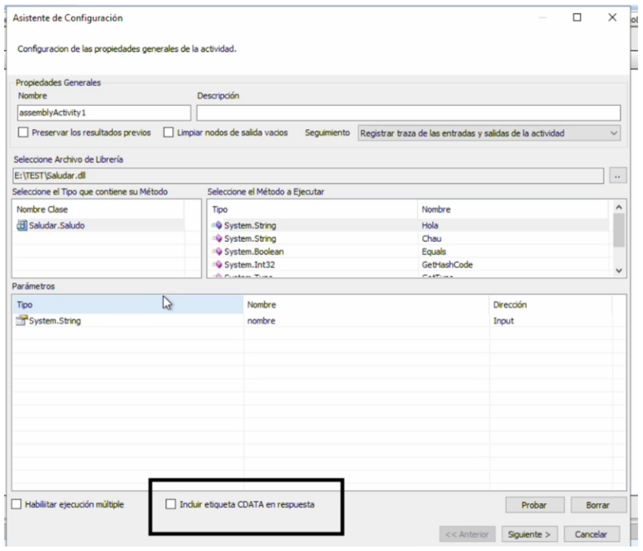
Tipo	Nombre	Dirección
System.String	nombre	Input

☑ Habilitar ejecución múltiple ☐ Incluir etiqueta CDATA en respuesta

Probar | Borrar

<< Anterior | Siguiente > | Cancelar

- **Include CDATA Tag in Response**: Useful when the method returns XML. This option encapsulates the result to avoid interpretation errors when processing the response.


Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales

Nombre: assemblyActivity1

Descripción:

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios Seguimiento: Registrar traza de las entradas y salidas de la actividad

Seleccione Archivo de Librería: E:\TEST\Saludar.dll

Seleccione el Tipo que contiene su Método

Nombre Clase
Saludar.Saludo

Seleccione el Método a Ejecutar

Tipo	Nombre
System.String	Hola
System.String	Chau
System.Boolean	Equals
System.Int32	GetHashCode

Parámetros

Tipo	Nombre	Dirección
System.String	nombre	Input

☐ Habilitar ejecución múltiple ☐ Incluir etiqueta CDATA en respuesta

Probar | Borrar

<< Anterior | Siguiente > | Cancelar

Practical Example

To apply what we have learned, we are going to make a specific configuration. Let's say we want to use the Saludar.dll library, which contains two methods: Hello and Chau. Both are given a name parameter and return a custom message.

The steps would be:

1. Select the corresponding class of Saludar.dll.
2. We chose the Hola method.
3. We define the input parameter name with a test value, for example, "John".
4. We test the activity to make sure the method returns "Hi, John."

We can repeat the same procedure with the Chau method to get a goodbye like "Chau, Juan".

Testing and Validation

Before applying the configuration in a real environment, we always validate its operation. The Test option allows us to run the method by manually entering the parameters.

During this stage we confirm:

- That the connection with the library and class is correct.
- That the parameters are being sent properly.
- That the answer is as expected.

Selecting Data Sources

Once we have verified that the activity is working correctly, we must map the parameters to process data. Values can come from:

- Process parameters.
- Process variables.
- Results of previous activities.
- System parameters (e.g., InstanceId, LoggedUser).

In our case, we can take a process parameter called Message and map it to the method name parameter. The output of the method can then be stored in another parameter such as Response, which we can reuse in later steps.

As we have seen, the Component Library activity is a key tool to extend BIZUIT's capabilities. It allows us to reuse existing business logic and easily connect to external libraries written in .NET or COM.

Now, we have the necessary knowledge to configure this activity, map its parameters and validate its execution, thus opening up new possibilities for integration and automation in our workflows.

File Activity Settings

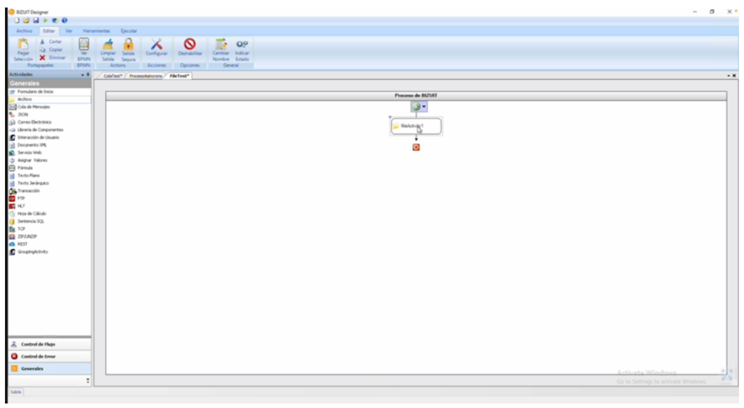
Next, we'll learn how to set up an integration activity in BIZUIT to perform operations on files and directories, both in local and shared folders. This functionality will allow us to automate day-to-day file manipulation tasks within our workflows.

The File activity in BIZUIT is a versatile tool that enables us to interact directly with the file system. Thanks to it, we can perform operations such as reading, writing, moving, copying, renaming or deleting files, thus facilitating the management of information in our automated processes.

We're going to explore each of these actions in detail, their configuration, and how to effectively integrate them within the workflow.

Initial Activity Setup

As a first step, we incorporate the File activity into the workflow and access its settings. From the initial screen, we can see the different actions that we can perform on files and folders.

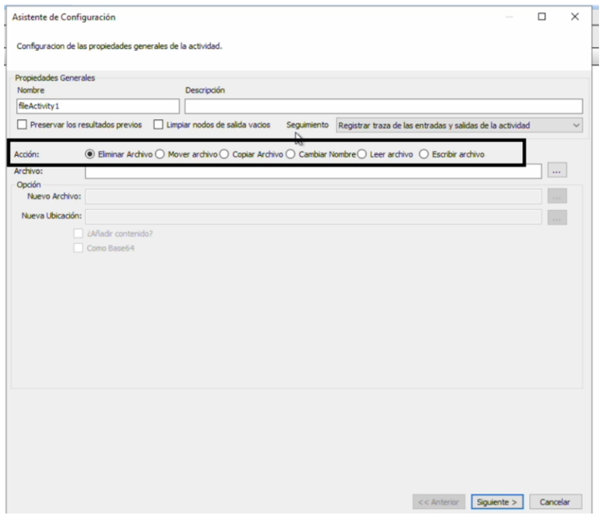


Available Stocks

In the Action field, we select the operation we want to perform. BIZUIT offers us multiple possibilities, including:

- **Delete File:** We set the activity to delete a file from the system. We just need to indicate the file path. No additional parameters are required.
- **Move File:** This option allows us to move a file from one location to another. We must specify both the original route and the new location.
- **Copy File:** By selecting this action, we create a copy of the file in another folder. We enter the source route and the new location.

- **Rename:** To rename a file, we indicate its current path and the new desired name.
- **Read File:** This option allows us to get the contents of a file and use it in our stream. We can indicate whether we want to interpret the content as plain text or Base64-encoded.
- **Write File:** With this action we write content to a file. We can create a new one or modify an existing one, selecting encoding, content type and if we want to add text to the end of the file instead of overwriting it.


Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales

Nombre: fileActivity1

Descripción:

[ ] Preservar los resultados previos [ ] Limpiar nodos de salida vacios

Seguimiento: Registrar traza de las entradas y salidas de la actividad

Acción: [✓ Eliminar Archivo] Mover archivo Copiar Archivo Cambiar Nombre Leer archivo Escribir archivo

Archivo:

Opción

Nuevo Archivo:

Nueva Ubicación:

[ ] ¿Añadir contenido?

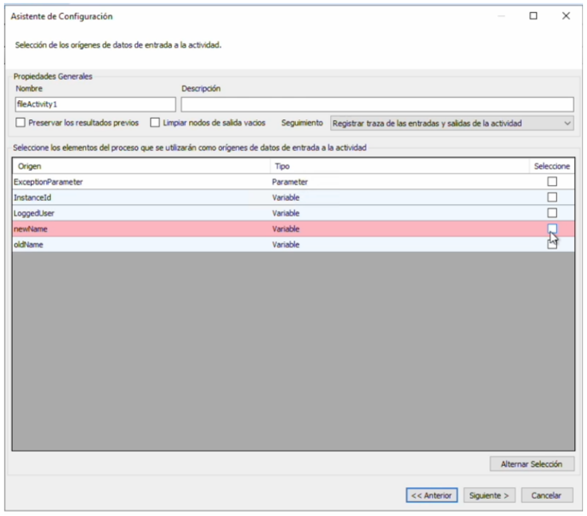
[ ] Como Base64

Practical Examples: Reading, Writing, and Deleting Files

Let's say we want to extract data from a .txt file with employee information. Select the Read File action and enter the file path.

This value can be overwritten from the input mappings screen. If we do not define a specific mapping, BIZUIT will default to the value configured in the activity.

We move forward with the configuration and select the newName variable as the data source to dynamically define the file name. When executing the process, the activity returns the contents of the file along with its name and path, values that we can assign to variables or parameters of the process.



Selección de los orígenes de datos de entrada a la actividad.

Propiedades Generales

Nombre: fileActivity1

Descripción:

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacíos Seguimiento: Registrar traza de las entradas y salidas de la actividad

Seleccione los elementos del proceso que se utilizarán como orígenes de datos de entrada a la actividad

Origen	Tipo	Seleccione
ExceptionParameter	Parameter	☐
InstanceId	Variable	☐
LoggedUser	Variable	☐
newName	Variable	☐
oldName	Variable	☐

Now let's imagine that we need to log a log. Select the Write File action, define the path and name of the destination file. We enabled the *Add Content option* so that the information is appended instead of overwritten.

We can define the character set and, if applicable, enable the encoding in Base64. In the mappings we assign:

- File path and name.
- Content to be written (dynamic or static).

This content will be recorded in the archive each time the process is executed.

Finally, in case a file is no longer needed, we use the Delete File action. We indicate the path and name of the file to be deleted.

From the input mappings, we assign the newName variable with the path of the file we want to delete. We run the process, look at the input and output data, and confirm that the file was indeed deleted from the system.

BIZUIT's Archive activity is a comprehensive solution for managing documents and file structures within our processes. With its proper configuration, we can automate essential tasks such as reading, writing, moving or deleting files, ensuring orderly and efficient management of information.

Unit 3: Setting Up Transformation Activities

General Concept

In today's integration environments, it is common for the systems involved to handle different data formats. JSON, XML, CSV or HL7, among others, are widely used standards, each with its own structure and characteristics. To get the flows in BIZUIT to work properly, we need to transform the data into the right format at each stage of the process. These transformations are key to ensuring compatibility between heterogeneous systems.

Why are they so important?

Format transformations are the bridge that allows information to travel fluidly between platforms with different requirements. Consider an inventory system that expects to receive information in XML, while an e-commerce channel requires JSON data. Or in the healthcare field, where the use of the HL7 format is essential for interoperability. Our ability to transform this data with BIZUIT is what allows us to connect diverse technologies, while always maintaining flow consistency.

Types of Data We Can Transform

Within BIZUIT, we have support to convert among the most frequent formats in business integrations:

- JSON: Ideal for modern web services and REST APIs, thanks to its lightness and speed.
- XML: Useful in more traditional enterprise environments and essential for SOAP-based integrations.
- CSV: Common in data exports, spreadsheets, or simple database systems.
- HL7: The standard par excellence in health, used to share information between clinical systems.

Main Format Converters in BIZUIT

1. From CSV/TXT to XML: When we receive information in flat files (e.g., from a spreadsheet), we can transform it into XML to integrate it with more modern systems that require hierarchical structures.

2. From XML to CSV/TXT: It is also possible to do the reverse operation: convert complex data into simple text files for analysis, reporting or exports to more basic systems.

3. From JSON to XML and vice versa: This conversion is vital for connecting REST applications with systems that still operate on top of SOAP. It allows us to maintain frictionless compatibility between modern and traditional.

4. From HL7 to XML: In healthcare, many systems work exclusively with HL7 messages. If one of our systems does not support it, we can transform them to XML, which allows us to integrate clinical data, shifts or studies without losing relevant information.

   Example:
   If we receive an HL7 message from a hospital, BIZUIT can automatically convert it into XML with readable tags such as <PatientID> or <DOB>, making it easier to analyze and store.

5. XML to HL7: Similarly, if we need to send information to a medical system, we can transform an XML generated in another environment into an HL7 message ready to be processed by that system.

   Example:
   Suppose we generate an XML with the laboratory results of a patient. With BIZUIT, we transform that file into an HL7 message valid to be received by a clinical system.

Configuration and Validation

Before applying any transformation, we need to make sure that the data is being converted correctly. For this, BIZUIT offers visual validation tools that allow us to:

- Preview the final result.
- Adjust the structure of the message.
- Verify that the mapping between elements is correct.

This not only gives us control over the transformation, but also reduces risks when implementing the flow into production.

HL7 Activity Settings

Now, we will address the configuration of HL7 activity, a fundamental tool that allows us to convert messages between HL7 and XML formats, facilitating the integration of systems in healthcare environments.

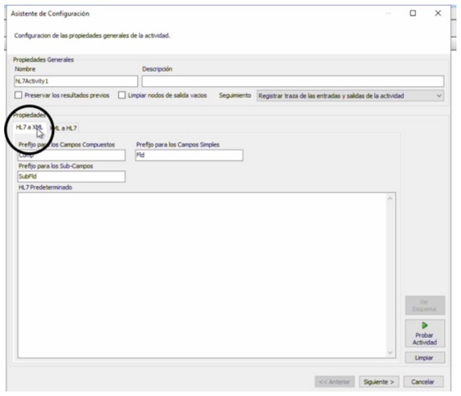
What is this activity for?

The HL7 standard is widely used in medical institutions for the exchange of clinical information. Thanks to this activity, we can integrate data from systems that use HL7 with others that operate in XML, achieving agile and reliable interoperability.

HL7 to XML Conversion

To convert HL7 messages to XML:

- Select the option "HL7 to XML".



Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales

Nombre: hL7Activity1

Descripción:

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios Seguimiento: Registrar traza de las entradas y salidas de la actividad

Propiedades

HL7 a XML | XML a HL7

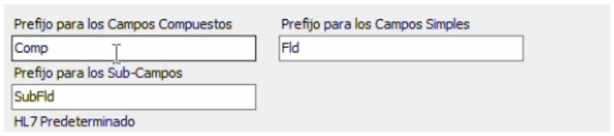
Prefijo para los Campos Compuestos: Comp

Prefijo para los Campos Simples: Fld

Prefijo para los Sub-Campos: SubFld

HL7 Predeterminado

- We define prefixes for compound, simple fields and subfields (e.g. Comp, Fld, SubFld) to organize the resulting XML.



Prefijo para los Campos Compuestos: Comp

Prefijo para los Campos Simples: Fld

Prefijo para los Sub-Campos: SubFld

HL7 Predeterminado

- We can visualize the final structure using the "View Schematic" option.

- We indicate the source of the data for the InputHL7Message parameter (for example, an input variable of the process).
- We execute the process and validate that the XML generated contains the correctly structured data (such as the patient's last name, in our example).

XML to HL7 Conversion

We can also perform reverse conversion:

- Select "XML to HL7".
- We use a pre-generated XML structure as a base.
- We create an XML variable (vXMLADTA01) that contains the structure of the message.
- We query a database with a SQL statement to populate that structure with actual data, preserving the data we don't replace.
- We pass the variable as input and get an automatically generated HL7 message.

Validation and testing

During testing, we check:

- That there are no errors in the conversion.
- The HL7 message or XML is correctly structured.
- The data has been mapped completely and accurately.

HL7 activity is essential for integrating medical systems with enterprise platforms into our workflows at BIZUIT. Its ability to convert between HL7 and XML bi-directionally allows us to automate clinical processes, improve interoperability, and ensure a secure and efficient flow of data.

JSON Activity Configuration

In this section we will work with the JSON activity in BIZUIT, an essential tool for converting data between JSON and XML formats. Thanks to this feature, we enable systems operating to different standards to communicate with each other smoothly and automatically.

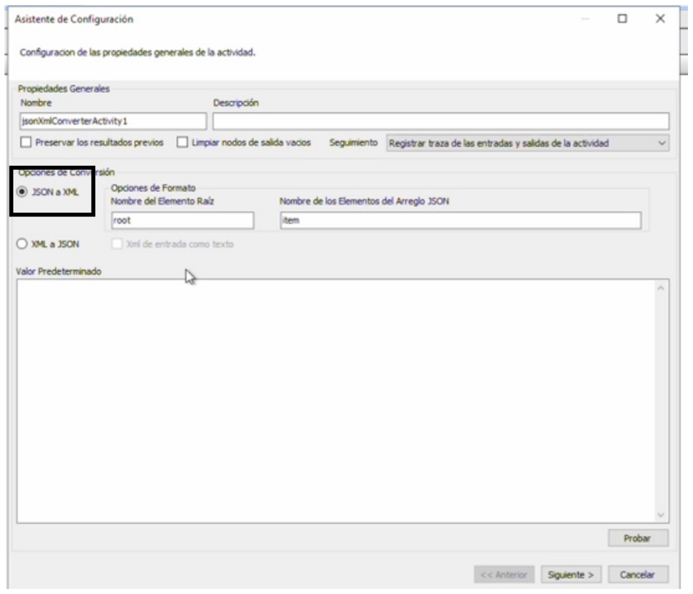
Let's think, for example, of a very common case: we receive information from an invoicing system in JSON format and we need to transform it into XML to load it into our document management system. This conversion can be set up quickly with JSON activity, avoiding manual tasks and ensuring data consistency.

In this demo, we're going to walk you through how to set up this activity, both in JSON to XML conversion and reverse conversion. We'll also look at how to define root elements, array elements, and how to map data to process variables.

The JSON activity in BIZUIT allows us to convert data both ways: from JSON to XML and from XML to JSON. This makes it a very valuable tool for adapting to the different formats required by modern systems (such as APIs that use JSON) and traditional business systems (which generally use XML).

Configuring JSON to XML Conversion

To get started, we select the "JSON to XML**"** option on the settings screen.


Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales

Nombre: jsonXmlConverterActivity1

Descripción:

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacíos Seguimiento: Registrar traza de las entradas y salidas de la actividad

Opciones de Conversión

[✓ JSON a XML]

Opciones de Formato

Nombre del Elemento Raíz: root

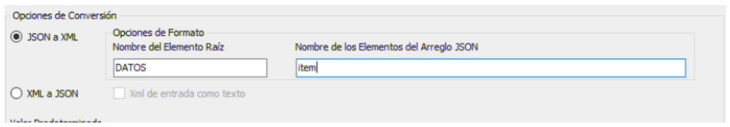
Nombre de los Elementos del Arreglo JSON: item

XML a JSON ☐ Xml de entrada como texto

Valor Predeterminado

Formatting Options

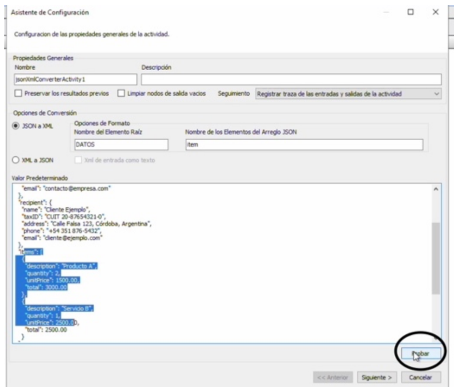
- **Root Element Name**: Here we define the name of the root node of the resulting XML. We can call it, for example, DATA.
- **Name of Array Items**: If the JSON includes arrays, we indicate how each item will be called in XML (e.g., item).



Opciones de Conversión
[✓ JSON a XML]
Opciones de Formato
Nombre del Elemento Raíz: DATOS
Nombre de los Elementos del Arreglo JSON: item
XML a JSON
Xml de entrada como texto

Practical Example

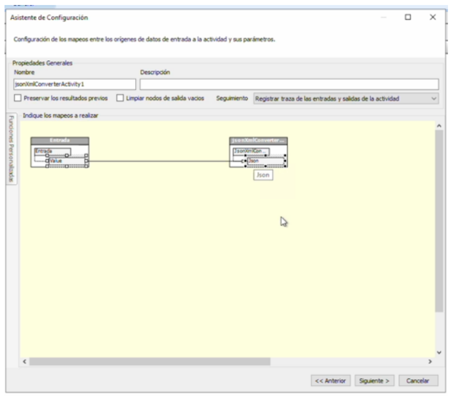
We start with a JSON object with data from an invoice (header and detail). When you click Test, the system automatically converts it to XML using the names we defined.



Asistente de Configuración
Configuracion de las propiedades generales de la actividad.
Propiedades Generales
Nombre: jsonXmlConverterActivity1
Descripción:
Preservar los resultados previos
Limpiar nodos de salida vacios
Seguimiento: Registrar traza de las entradas y salidas de la actividad
Opciones de Conversión
[✓ JSON a XML]
Opciones de Formato
Nombre del Elemento Raíz: DATOS
Nombre de los Elementos del Arreglo JSON: item
XML a JSON
Xml de entrada como texto
Valor Predeterminado

```
    "email": "contacto@empresa.com"
  },
  "recipient": {
    "name": "Cliente Ejemplo",
    "taxID": "CUIT 20-87654321-0",
    "address": "Calle Falsa 123, Córdoba, Argentina",
    "phone": "+54 351 876-5432",
    "email": "cliente@ejemplo.com"
  },
  "items": [
    {
      "description": "Producto A",
      "quantity": 2,
      "unitPrice": 1500.00,
      "total": 3000.00
    },
    {
      "description": "Servicio B",
      "quantity": 1,
      "unitPrice": 2500.00,
      "total": 2500.00
    }
```

Then, we move on to the next step of the wizard and select the data source. In our case, we take the Process Input parameter and map it to the Json node of the JsonXmlConverter parameter in the activity.



Asistente de Configuración
Configuración de los mapeos entre los orígenes de datos de entrada a la actividad y sus parámetros.
Propiedades Generales
Nombre: jsonXmlConverterActivity1
Descripción:
Preservar los resultados previos
Limpiar nodos de salida vacios
Seguimiento: Registrar traza de las entradas y salidas de la actividad
Indique los mapeos a realizar

After running the process, we verify that the output parameter (e.g., Output) correctly contains the invoice number, already transformed and mapped.

Configuring XML to JSON Conversion

If what we need is to transform an XML to JSON, we select the **"**XML to JSON" option.

We load a sample XML in the "Default" field and indicate that it is plain text. Clicking Test converts to JSON, automatically mapping the XML nodes to JSON objects.

Again, we select the data source: we take the Input parameter, link it to the Xml node of JsonXmlConverter, and set the output to contain the generated JSON.

Conversion Test

Before implementing any integration, it is critical to test:

- That there are no conversion errors.
- That the final structure (either XML or JSON) is as expected.
- That all the information is present and correctly located.

The JSON activity is a key piece in BIZUIT to achieve effective integration between systems that work with different data formats. Whether transforming JSON into XML to integrate it into legacy systems or converting XML to JSON to interact with modern APIs, this tool gives us flexibility and control over the flow of information.

Thanks to this capability, we can automate complex processes, reduce human error, and ensure that each system receives exactly the data it needs.

XML Document Activity Settings

Now we will explore the XML Document activity in BIZUIT, a tool that allows us to convert an XML text received as a scalar parameter into a structured document that can be interpreted and manipulated within our processes.

Let's imagine that we receive XML information about a customer or an order, from another system. With this activity, we can transform that text into an organized document, on which we can then apply mappings, readings or specific transformations with total precision.

During this demo, we're going to look at how to set up text replacements, remove unnecessary namespaces, and set a default XML for testing. Our goal is for you to be able to integrate these documents into your flows with clarity, order and efficiency.

The XML Document activity allows us to interpret a text in XML format and convert it into a structured document. This functionality is essential when we want to process data that comes from other systems in the form of text and we need to work on it within BIZUIT in a structured and logical way.

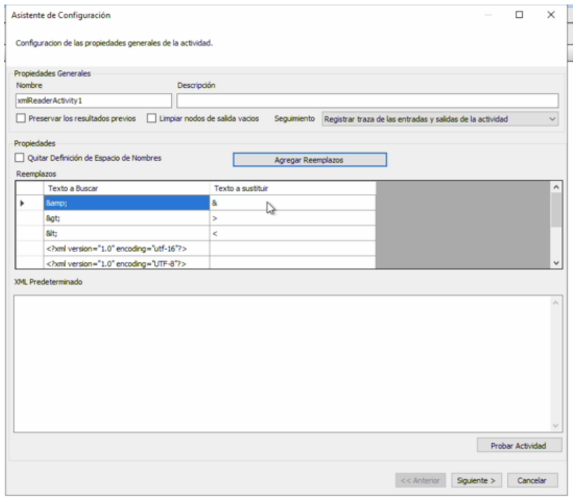
Configuring Replacements in XML Text

One of the first steps is to define specific replacements in the text. This is useful for correcting special characters or making adjustments before processing the XML.

Example of Replacements

In the "Replacements" section, we indicate pairs of text to search for and their corresponding substitution. Some typical examples:

- & by &
- > by >
- < by <



Asistente de Configuración
Configuracion de las propiedades generales de la actividad.
Propiedades Generales
Agregar Reemplazos
Probar Actividad
<< Anterior
Siguiente >
Cancelar

Nombre: xmlReaderActivity1

Descripción:

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios

Seguimiento: Registrar traza de las entradas y salidas de la actividad

Propiedades

☐ Quitar Definición de Espacio de Nombres

Reemplazos

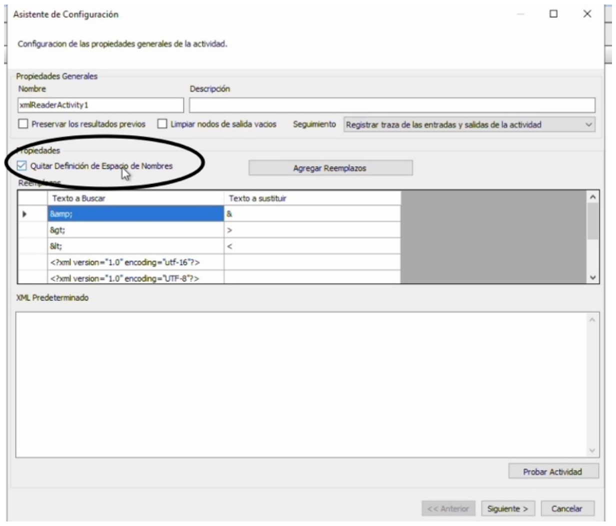
Texto a Buscar	Texto a sustituir
&amp;	&
&gt;	>
&lt;	<
<?xml version="1.0" encoding="utf-16"?>	
<?xml version="1.0" encoding="UTF-8"?>	

XML Predeterminado

These replacements ensure that the XML is well-formed and can be processed correctly, especially if the text comes from systems that apply special encoding or use escape characters.

Remove Namespace Definitions

In some cases, XML includes namespaces that are not necessary for our flow and can generate conflicts. By enabling the "Remove Namespace Definition" option, we remove these definitions to simplify the structure.



Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales

Nombre: xmlReaderActivity1

Descripción:

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios Seguimiento: Registrar traza de las entradas y salidas de la actividad

Propiedades

☑ Quitar Definición de Espacio de Nombres

Reemplazos

Texto a Buscar	Texto a sustituir
&amp;	&
&gt;	>
&lt;	<
<?xml version="1.0" encoding="utf-16"?>	
<?xml version="1.0" encoding="UTF-8"?>	

XML Predeterminado

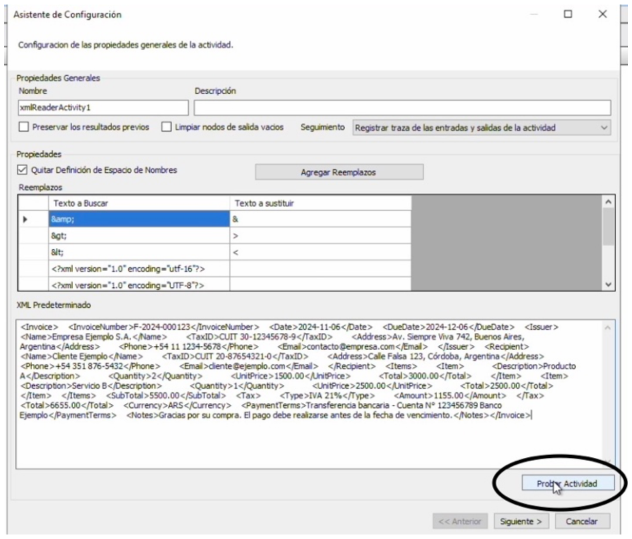
This is especially useful when we work with XML from different sources and seek uniformity in their structure, reducing possible friction when processing or mapping them.

XML Default

We can load a default XML as a basis for visualizing and testing how the document will be structured once transformed.

Default XML Example

Let's say we load an XML with information from an invoice. By pressing "Test Activity", the tool transforms the text into a structured document and shows us how it will be treated within the flow. We can confirm that all data is properly organized.



Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales

Nombre: xmlReaderActivity1

Descripción:

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios Seguimiento: Registrar traza de las entradas y salidas de la actividad

Propiedades

☑ Quitar Definición de Espacio de Nombres

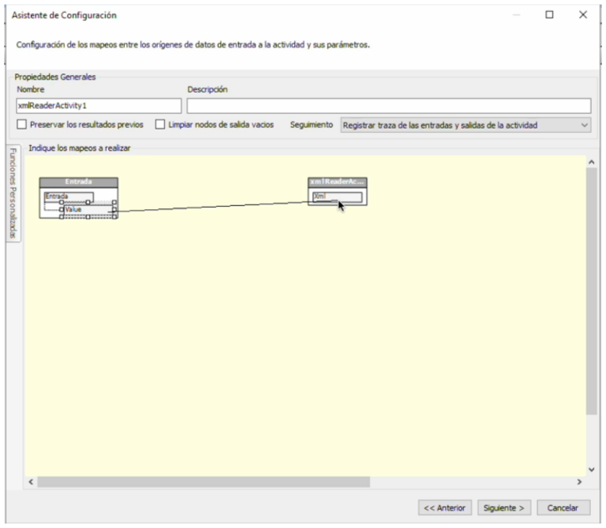
Reemplazos

Texto a Buscar	Texto a sustituir
&amp;	&
&gt;	>
&lt;	<
<?xml version="1.0" encoding="utf-16"?>	
<?xml version="1.0" encoding="UTF-8"?>	

XML Predeterminado

```
<Invoice>   <InvoiceNumber>F-2024-000123</InvoiceNumber>   <Date>2024-11-06</Date>   <DueDate>2024-12-06</DueDate>   <Issuer>
<Name>Empresa Ejemplo S.A.</Name>      <TaxID>CUIT 30-12345678-9</TaxID>      <Address>Av. Siempre Viva 742, Buenos Aires,
Argentina</Address>      <Phone>+54 11 1234-5678</Phone>      <Email>contacto@empresa.com</Email>   </Issuer>   <Recipient>
<Name>Cliente Ejemplo</Name>      <TaxID>CUIT 20-87654321-0</TaxID>      <Address>Calle Falsa 123, Córdoba, Argentina</Address>
<Phone>+54 351 876-5432</Phone>      <Email>cliente@ejemplo.com</Email>   </Recipient>   <Items>      <Item>         <Description>Producto
A</Description>         <Quantity>2</Quantity>         <UnitPrice>1500.00</UnitPrice>         <Total>3000.00</Total>      </Item>      <Item>
<Description>Servicio B</Description>         <Quantity>1</Quantity>         <UnitPrice>2500.00</UnitPrice>         <Total>2500.00</Total>
</Item>   </Items>   <SubTotal>5500.00</SubTotal>   <Tax>      <Type>IVA 21%</Type>      <Amount>1155.00</Amount>   </Tax>
<Total>6655.00</Total>   <Currency>ARS</Currency>   <PaymentTerms>Transferencia bancaria - Cuenta N° 123456789 Banco
Ejemplo</PaymentTerms>   <Notes>Gracias por su compra. El pago debe realizarse antes de la fecha de vencimiento.</Notes></Invoice>
```

Then, we move on to the next step of the wizard, where we select the data source. In our case, we take the Process Input parameter and map it to the Xml node of the corresponding activity parameter.



Asistente de Configuración

Configuración de los mapeos entre los orígenes de datos de entrada a la actividad y sus parámetros.

Propiedades Generales

Nombre: xmlReaderActivity1

Descripción:

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios Seguimiento: Registrar traza de las entradas y salidas de la actividad

Indique los mapeos a realizar

We also assign a default XML to the input parameter and run the process. When finished, we check that the output parameter (e.g., Output) correctly contains the desired information, such as the invoice number, extracted from the already structured XML document.

During this test, we verify that:

- The replacements have been successfully applied.
- The structure of the generated document is valid and complete.
- The XML format is compatible with the rest of the target stream or system.

The XML Document activity in BIZUIT is a key tool for transforming plain text into structured XML documents. It allows us to adapt and prepare the data for later use, seamlessly integrating information from external sources.

By applying overrides and removing namespaces, we make the final document ready to be interpreted and mapped, either to other activities within the flow or as an output to another system.

This capability is especially useful in contexts where XML is the exchange standard, such as integrations with legacy systems or web services.

With this activity, transforming and processing XML in BIZUIT becomes a simple, effective task that is fully integrated into our work logic.

Plain Text Activity Settings

Finally, we are going to work with the BIZUIT Plain Text activity, a tool designed to convert plain text files –such as CSV or fixed width– into XML structures that we can process within our flows.

This type of file is common in legacy systems or exports from spreadsheets, and thanks to this activity, we can transform that data into structured documents, ready to be read, mapped or integrated with other systems.

Consider, for example, an organization that keeps a list of employees in a .csv file. With this functionality, we can read that file, convert it to XML, and manipulate each record individually, integrating it into the rest of the process effortlessly.

Setting Up the Plain Text Activity in BIZUIT

The activity allows you to interpret files in two main formats: character-delimited (such as CSVs) or fixed-width. In both cases, BIZUIT transforms this data into an ordered XML structure.

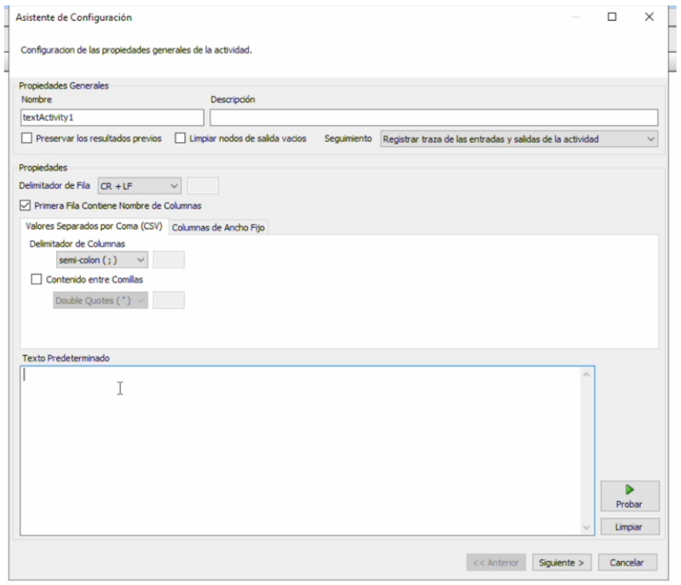
Basic Configuration: Comma Separated Values (CSV)

We select the "Comma Separated Values (CSV)" option, ideal for delimited files.

Configuration parameters:

- **Row Delimiter**: We specify how rows are separated (e.g., line break).

- **First Row Contains Column Names**: We enable this option if the first row defines the names of each field.
- **Column Delimiter**: we indicate the separator character (comma, semicolon, etc.).
- **Content in quotation marks**: if the values are in quotation marks, we activate this option and define the type of quotation marks.



Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales

Nombre: textActivity1
Descripción:
☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios Seguimiento: Registrar traza de las entradas y salidas de la actividad

Propiedades

Delimitador de Fila: CR + LF
☑ Primera Fila Contiene Nombre de Columnas

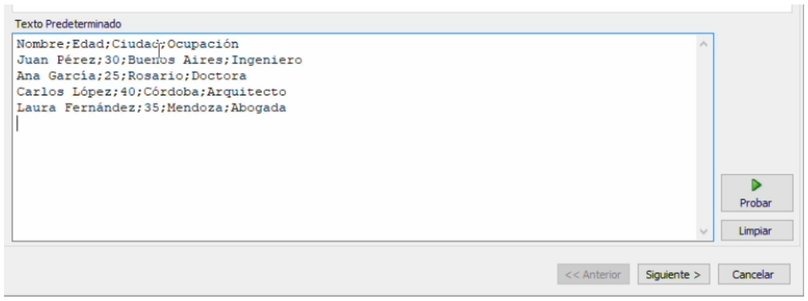
Valores Separados por Coma (CSV) | Columnas de Ancho Fijo

Delimitador de Columnas: semi-colon (;)
☐ Contenido entre Comillas
Double Quotes (")

Texto Predeterminado

Example with CSV

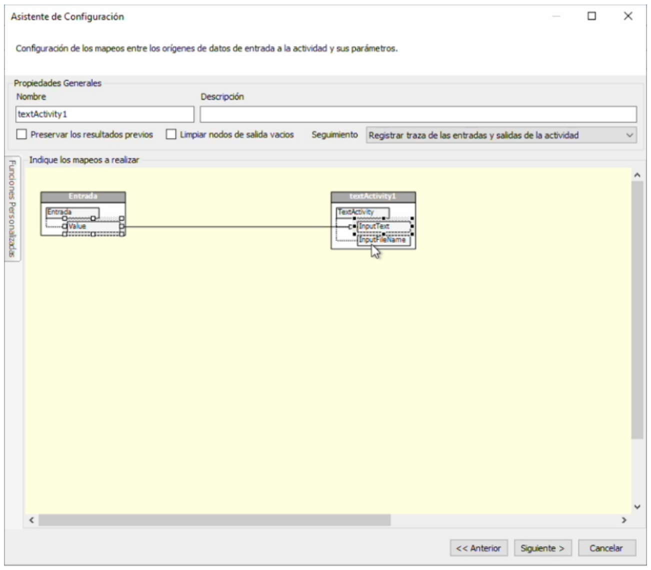
We load a default text with columns such as Name, Age, City, and Occupation. When testing the activity, we see how each row in the file is transformed into an XML node, and each value into a corresponding subnode.



Texto Predeterminado

```
Nombre;Edad;Ciudad;Ocupación
Juan Pérez;30;Buenos Aires;Ingeniero
Ana García;25;Rosario;Doctora
Carlos López;40;Córdoba;Arquitecto
Laura Fernández;35;Mendoza;Abogada
```

Then, we move on to the next step of the setup wizard. In our case, we select the parameter from the Input process and map it to the InputText node of the TextActivity parameter. (There's also the option to use InputFileName for files, but we don't apply it in this example.)


Asistente de Configuración

Configuración de los mapeos entre los orígenes de datos de entrada a la actividad y sus parámetros.

Propiedades Generales

Nombre: textActivity1

Descripción:

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios

Seguimiento: Registrar traza de las entradas y salidas de la actividad

Indique los mapeos a realizar

We assign the CSV as the default value and run the process. We verify that the output parameter—for example, Output—contains the name of the first employee, obtained after transformation and mapping.

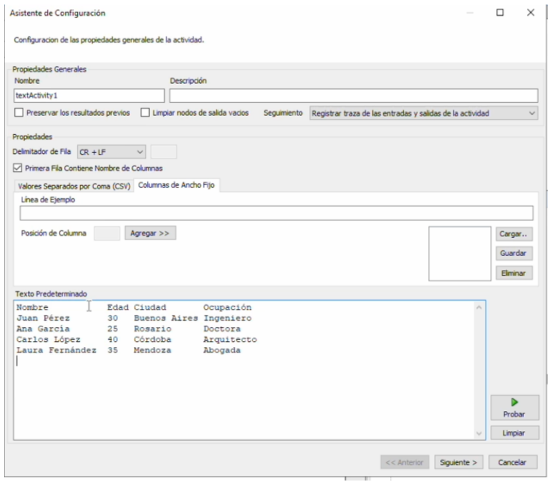
If we wanted to, we could use an Iteration activity to loop through each record individually and apply specific logic over each row.

Fixed Width Configuration for Columns

If we are working with files in which each field has a predefined width, we select the "Fixed Width Columns" option.

Example of Fixed Width

We load an example text with aligned fields, such as Name, Age, City, and Occupation. In the "Example Line" box, we copy a line from the file and use the mouse to indicate where each field ends. By clicking Add, we inform the activity how to interpret each segment.


Asistente de Configuración

Configuracion de las propiedades generales de la actividad.

Propiedades Generales

Nombre	Descripción
textActivity1	

☐ Preservar los resultados previos ☐ Limpiar nodos de salida vacios Seguimiento: Registrar traza de las entradas y salidas de la actividad

Propiedades

Delimitador de Fila: CR + LF

☑ Primera Fila Contiene Nombre de Columnas

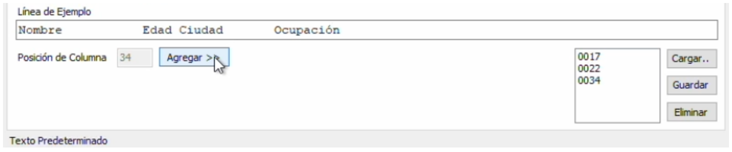
Valores Separados por Coma (CSV) | Columnas de Ancho Fijo

Línea de Ejemplo

Posición de Columna

Texto Predeterminado

```
Nombre          Edad Ciudad       Ocupación
Juan Pérez      30   Buenos Aires Ingeniero
Ana García      25   Rosario      Doctora
Carlos López    40   Córdoba      Arquitecto
Laura Fernández 35   Mendoza      Abogada
```


Línea de Ejemplo

```
Nombre          Edad Ciudad       Ocupación
```

Posición de Columna: 34

```
0017
0022
0034
```

Texto Predeterminado

Once set up, we test the activity and watch as each line of text is converted into a structured XML document that maintains the logic of the original file.

Again, we map the Process Input parameter to the InputText node of the TextActivity parameter, assign a fixed-width text as the default, and execute. When finished, we see that the Output parameter contains the name of the first employee, successfully transformed from the original text.

Conversion Test

During testing, we verify that:

- The conversion is done without errors.
- The generated XML is correctly structured.
- Data is kept complete and orderly, ready to be integrated or processed.

The Plain Text activity in BIZUIT allows us to transform plaintext data into organized XML structures, facilitating integration with other systems.

Whether we work with .csv or fixed-width files, this activity gives us the tools to automate the processing and conversion of information, ensuring its structure and compatibility with the rest of the flow.

Thanks to this functionality, any organization that still manages plaintext information can seamlessly integrate into modern digital ecosystems, using BIZUIT as a bridge between legacy formats and current structures.

Conclusion

Throughout this unit, we saw how BIZUIT allows us to work with multiple data formats, adapting them according to the needs of the systems we integrate with. Thanks to these transformations, we can connect different platforms from business environments to health systems, ensuring effective communication, regardless of the original or target format.

Transforming data isn't just a technical issue: it's a way to ensure that information gets where it needs to go, in the right form and structure. In short, it is what makes real and functional integration possible.

Chapter Summary

We start with the essentials: the basics of data integration, transformation, and mapping. We learned what integration activities are and why they are essential to connect BIZUIT with external systems, ensuring smooth communication between different platforms. Then, we get into the transformation activities. We discovered how to adapt the data to the formats required by other systems and how mappings allow that information to be organized so that each system can read and understand it without errors.

On the practical side, we set up connections to databases, REST and SOAP APIs, FTP servers, and even services that use the TCP protocol. These integrations allow us to work with real-time data or synchronize information in complex processes. Finally, we delve into the transformation of formats: from JSON and XML, to CSV, fixed-width plaintext, and HL7 messages. Each of these formats poses different challenges, and with BIZUIT's tools we saw how to address them effectively.

This chapter gave us a comprehensive overview of how to leverage BIZUIT in real integration projects. We now have the skills to automate processes, connect diverse systems, and transform data as required by each environment. Every unit we walked through is a key piece in building robust, scalable, and well-designed workflows. What we learned here not only helps us master the tool, but also think more strategically about how to connect information in a digital ecosystem.

Thank you for joining us here. We greatly appreciate the time and dedication you put into every step of this chapter. We hope that these tools will drive them to continue developing solutions with BIZUIT, to innovate in their work environments, and to face new integration challenges with security and creativity.