Genesys Intelligent Automation Help

Using iHub

12/13/2025

Contents

Using iHub

This page describes how to set up Integration Processes to integrate the customer's backend resources, such as web services and databases, with Genesys Intelligent Automation.

Integration Hub is a simple and powerful way to do this, and brings many other benefits such as support for multiple environments' endpoints (dev/test/production, for example) and the ability to create automated test scripts. Using Integration Hub is a good way to keep your Intelligent Automation apps and their assorted presentation logic separate from the details of how to call onto your enterprise's backend systems.

Script blocks are great for adding 'presentation' logic to your app - comparing values and branching out to different blocks, performing date calculations, combining prompts. It's possible to invoke RESTful web services directly from a Script block, and this may be sufficient if the services are simple. It is advised not to the Script block for tasks like handling security certificates, or calling onto databases or SOAP-based web services

> **Tip**
>
> For information on working with Groovy, refer Apache Groovy documentation.

iHub Interface

The iHub interface is where you set up the Processes that integrate Intelligent Automation with the customer's resources, such as databases and web services.

iHub has four tabs, as follows:

- **Processes**—In this tab, you define the Integration Processes that will handle HTTP(S) and JDBC requests that are received from the Intelligent Automation VUI. Use the scripting commands described in Scripting Commands to write any Process and Library scripts.
- **Deploy to Production**—After you have created and tested your Processes, you put them into production using this tab. Deployments in the current calendar year are displayed in tabular and calendar (weekly) format. From the table of deployments, you can also rollback to previous productions if needed.
- **Import**—You can import Process scripts and Shared scripts from other Processes.
- **Export**—You can export Process scripts, including the latest production versions, and library entries into a compressed (**.zip**) file; compatible for use with the Import tab.

Processes

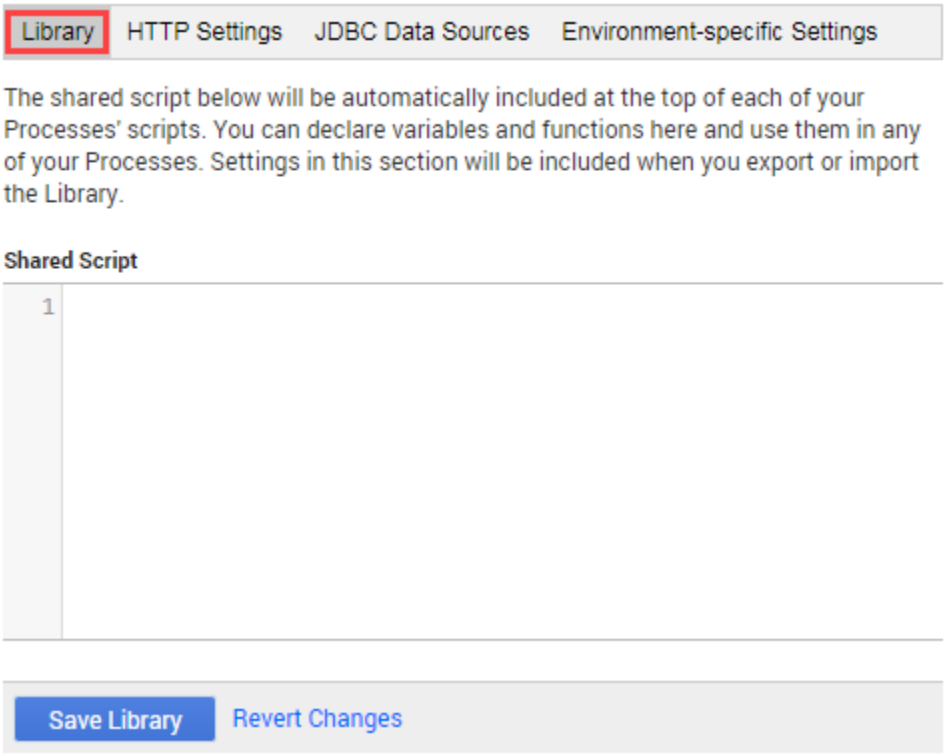
Defining Common Properties

To simplify your work, Intelligent Automation allows you to specify common properties, such as variables, functions, and environment settings, that can be used for all of your processes as required. In addition, if any Company-specific configuration information applies to a specific callflow engine and resides on a web site or in a database, you can specify the location and access credentials of the site or database to which the VUI must connect, even securing the connection if necessary. You can do all this on the right-hand panel of the Processes tab.

> ### Tip
> These settings can be modified or deleted at any time by opening the **Processes** tab and making the necessary changes.

Library tab

On the Library tab, create a list of additional variables and functions that can be shared between Processes. The Library is inserted at the top of a Process script, so you can reference the variables and functions throughout the script, as needed.

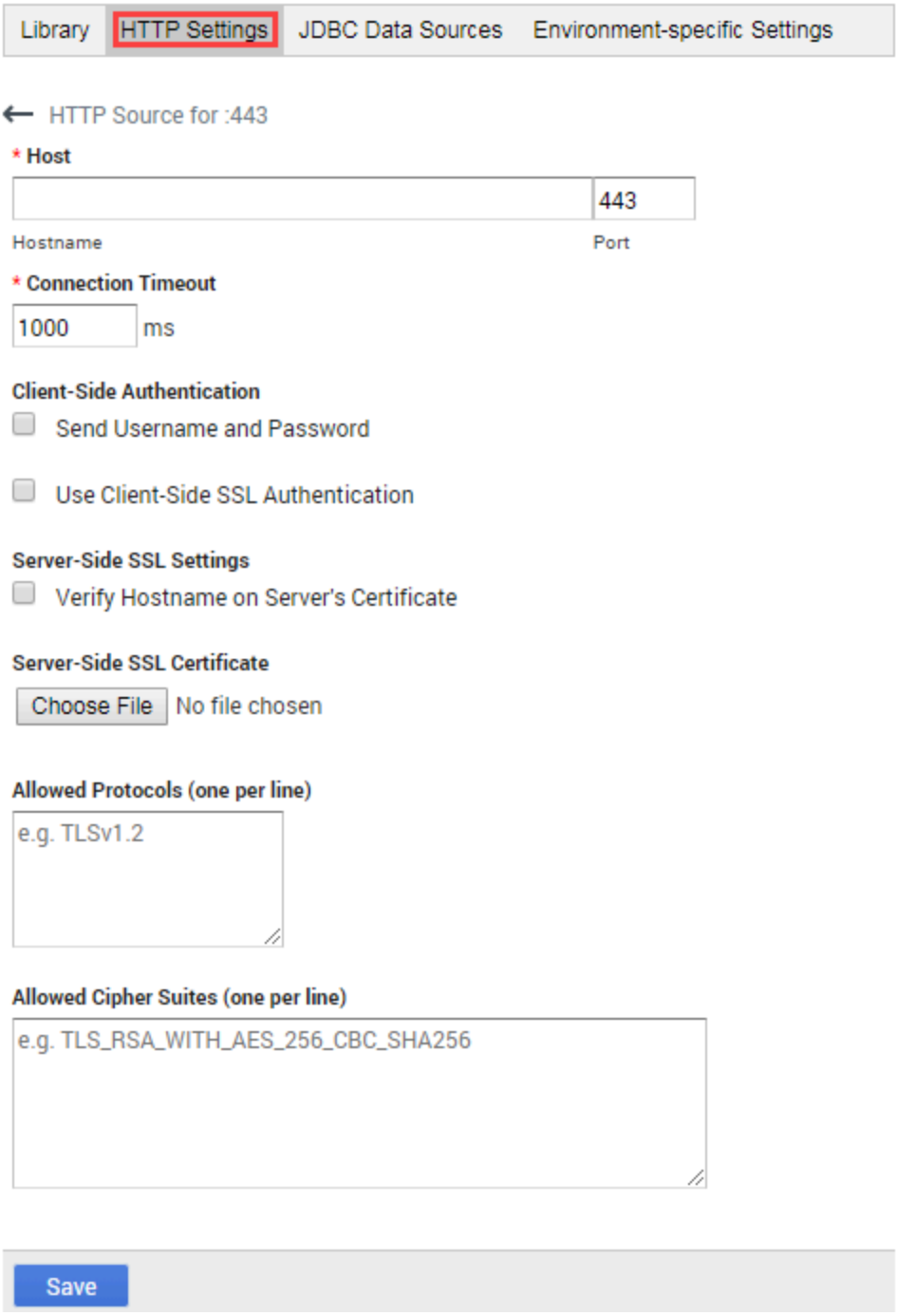


HTTP Settings tab

On the HTTP Settings tab, provide the Company-specific configuration information that applies to a given Intelligent Automation callflow engine installation on a website.

You can set up multiple secure connections between Intelligent Automation and external websites.

Enter only a **Hostname**, **Port** (defaults to the port used by the installation in which you are working), and the **Connection Timeout**. If you want to set up a secure connection, you will have to provide additional information. For more information about the security concepts, refer to the *Genesys Security Deployment Guide*.



When calling an external web site using Groovy, the list of trusted SSL certificates must be provided. Intelligent Automation supports both client-side authentication and server-side authentication.

The **Send Username and Password** option allows access to external servers.

When using **Client-Side Authentication**, Intelligent Automation allows uploading the client

certificate including the public key certificate and a private key along with the password. IA stores this certificate information and uses it for verification.

You can upload a server certificate also. The **Verify Hostname on Server's Certificate** option verifies if the host name matches with the certificate.

IA supports both client-side and server-side authentication individually and also together.

JDBC Data Sources tab

If any of this Company-specific information resides in a database or in SQL connection pools, define the JDBC parameters on this tab. The contents of this tab can differ depending on what you specify in the **Database Type** field, and by what you enter in other fields. Enter the required information in this tab, noting the following:

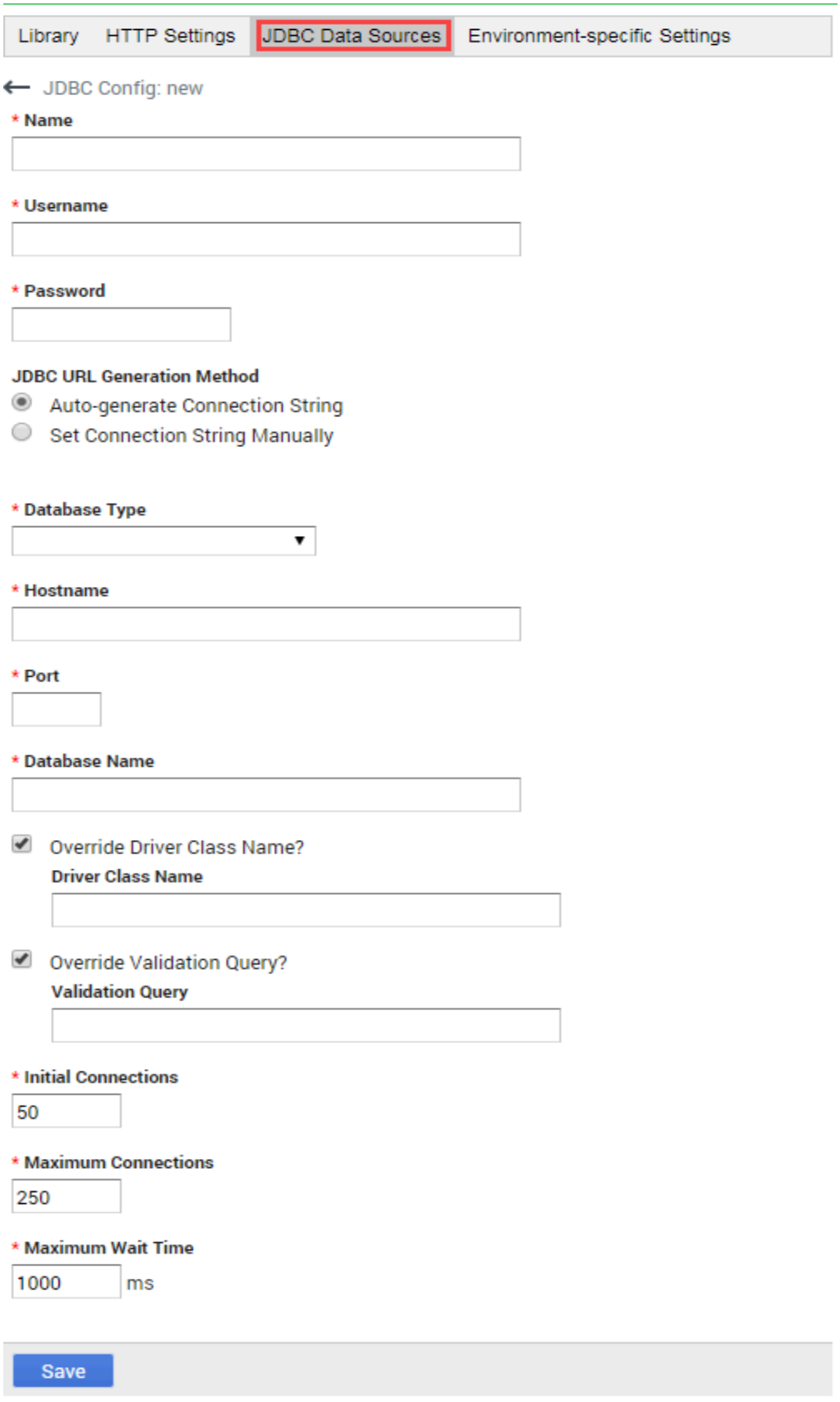
- The entry in the **Name** field is unique, but is for internal use only.
- You can enter the JDBC URL manually by selecting **Set Connection String Manually** or have Intelligent Automation construct it from the information you enter in the fields that open when you click **JDBC URL Generation Method**.
- The are four supported **Database Type**s—SQL Server, PostgreSQL, MySQL, and Oracle (using either a System ID or a Service Name).

> **Warning**
>
> Genesys does not ship with a driver for MySQL and PostgreSQL, so if you are using the MySQL or PostgreSQL DBMS, you will have to obtain that separately.

- Enter a driver name in **Override Driver Class Name** only if you need to override the driver class name, for example if you are using an older or custom version of a driver.
- If you do not want to use the standard validation query specific to the database type (for example, `SELECT  1` for SQL Server) that is used by the connection pool to determine if the connection is working, you can specify a custom one, perhaps based on the Company name, in **Override Validation Query**.
- The **Initial Connections** field indicates the number of connections that are made initially when connecting to the database. The allowed permissible range is `0` to the value configured in **Indy.ConnectionPooling.JDBC.MaxSize**.
- The **Maximum Connections** field indicates the maximum number of connections to the database. The default value is `10`.

> **Important**
>
> To increase the maximum number of connections, modify the **Indy.ConnectionPooling.JDBC.MaxSize** parameter (**Administration** > **Default Server Settings**). The allowed permissible range is `0` to the value configured in **Indy.ConnectionPooling.JDBC.MaxSize**.

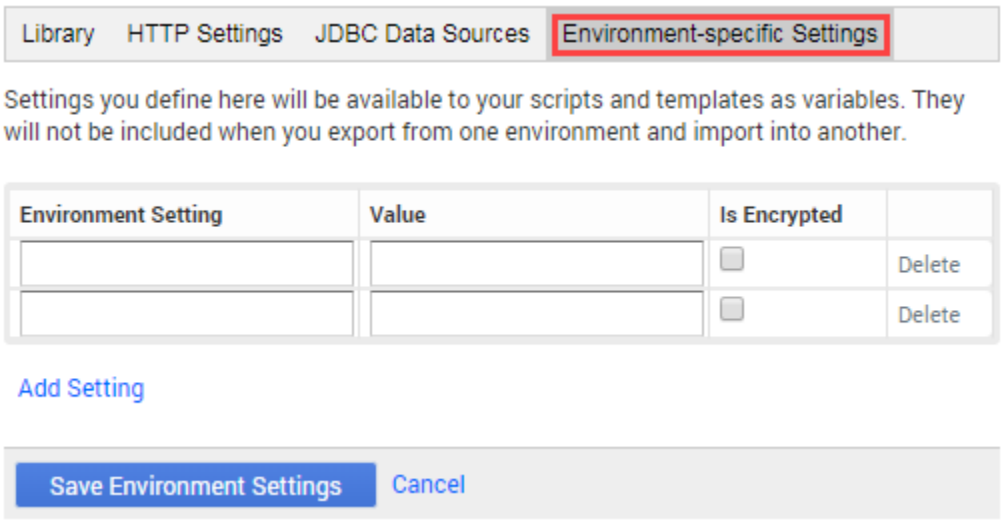
- The **Maximum Wait Time** field (or maximum time to wait for a pool connection) is set by default as 1 and maximum of 300000 milliseconds.


Library
HTTP Settings
JDBC Data Sources
Environment-specific Settings
JDBC Config: new
* Name
* Username
* Password
JDBC URL Generation Method
Auto-generate Connection String
Set Connection String Manually
* Database Type
* Hostname
* Port
* Database Name
Override Driver Class Name?
Driver Class Name
Override Validation Query?
Validation Query
* Initial Connections
50
* Maximum Connections
250
* Maximum Wait Time
1000
ms
Save

Environment-specific Settings tab

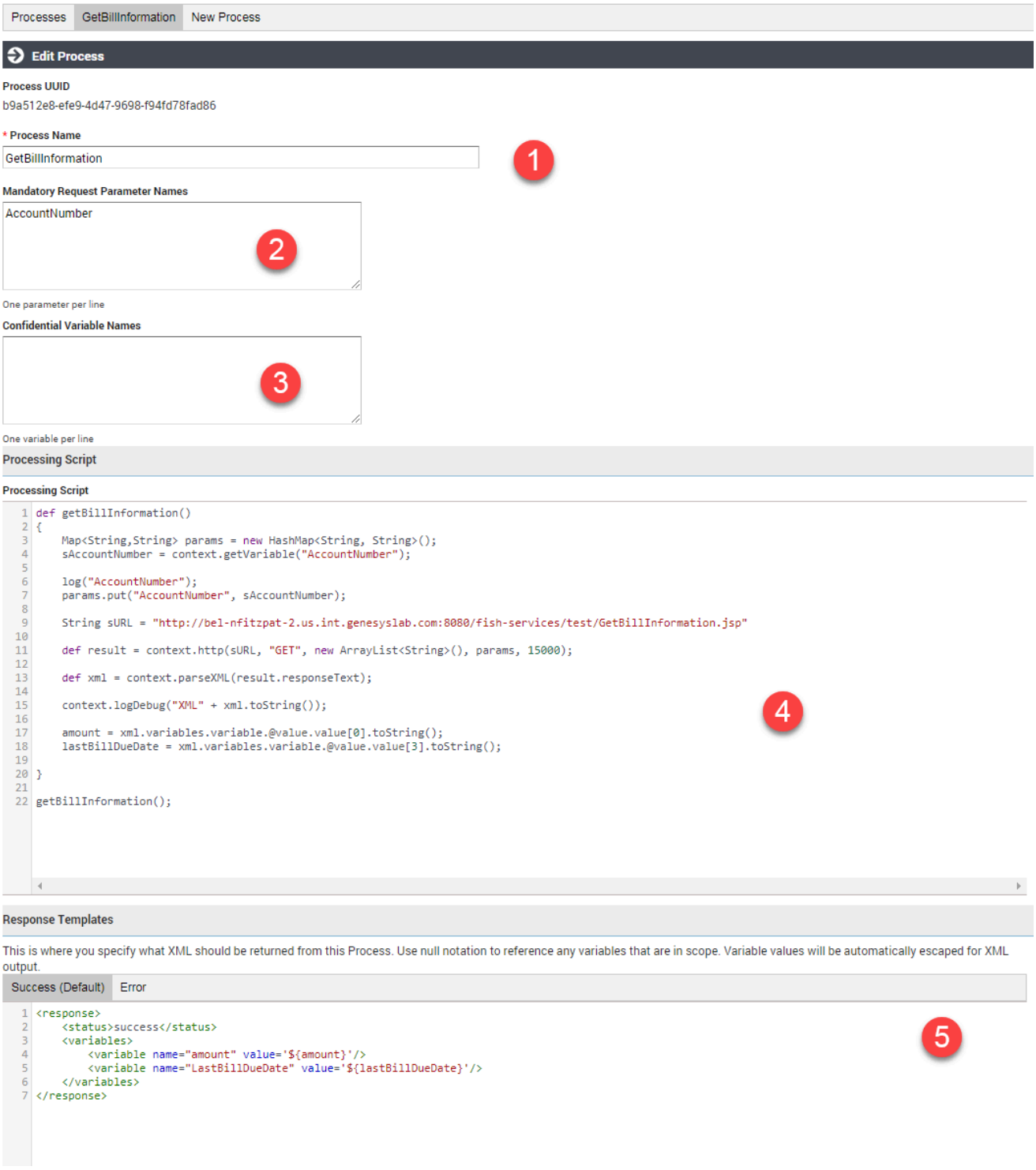
In this tab, define any settings that are specific to the environment in which you are working, such as Lab, Test, or Production.

Variables used during processing (like hostname and port) can be declared here and accessed by Groovy/Java scripts later on.



Creating a Process

Create a new Process in the **Processes** tab, as indicated in the following diagram:



After you click **New Process** in the tab's header bar or click the **Create a New Process** button:

1. Enter a unique **Process Name**. Intelligent Automation automatically assigns a **Process UUID** for a new

process. This UUID stays with this Process until the Process is deleted.

2. Enter any parameters that will be required in the request from VUI. These parameters will be used by the process as it is executing. If the request does not contain these parameters, it will be rejected.
3. Enter any confidential parameters. These parameters might contain customer-sensitive data, such as passwords or credit card numbers, and will not be written to logs or reporting databases.
4. Using the scripting commands provided with Intelligent Automation, enter the code for the script that will process the request.
5. Create the **Response Template** that will be returned by this Process.You must create at least one Response Templates - one for successful Process execution (**Success**). A Response Template for unsuccessful execution (**Error**) is optional.

Modifying a Saved Process

To modify a saved (existing) process, even its name, click Processes on the **Processes** tab to view the list of Processes. Select it by name on the Processes tab, and make the appropriate changes.

Deleting an Existing Process

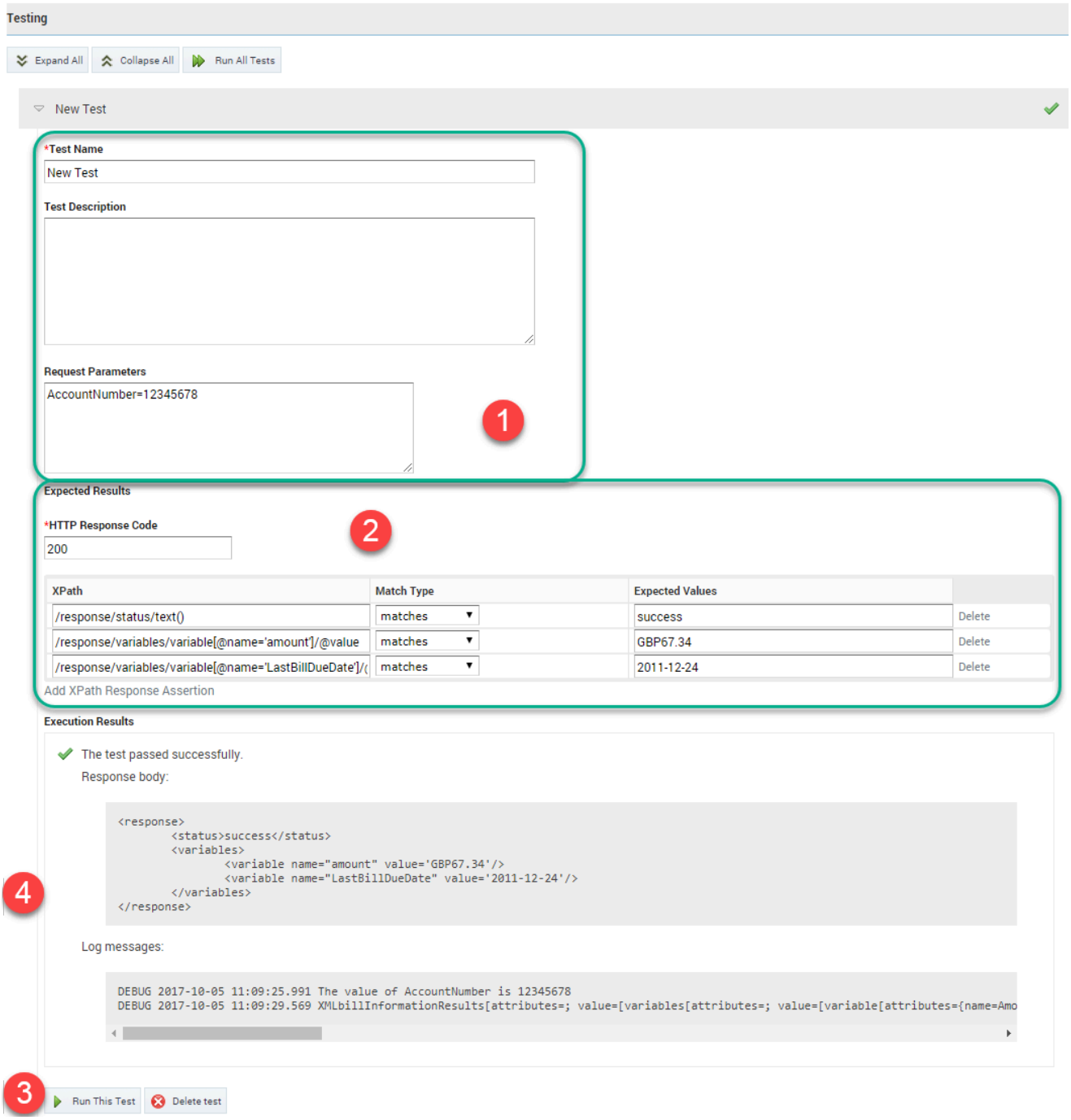
To delete a process, click Processes on the **Processes** tab to view the list of Processes. Select the process you want to delete in the list, and click the garbage can icon to delete it.

Testing a Process

iHub allows you to test your Processes by defining test cases and running them against any Processes you have created, *before* the Processes are put into production. You can also test a production version by copying the currently deployed version of a Process to the test side of the environment.

> ### Warning
> If a Process by that name already exists, the test version will be overwritten by the copied production version, and any undeployed changes to the original test version will be lost.

Refer to the following diagram to create, run, and examine the results of a test case. The numbers in the diagram correspond to the tasks that follow.



Creating a Test Case

After clicking **New Test Case**:

1. Define the test case by giving it a specific **Test Name**, an optional **Test Description**, and values of any **Request Parameters** required by the process.

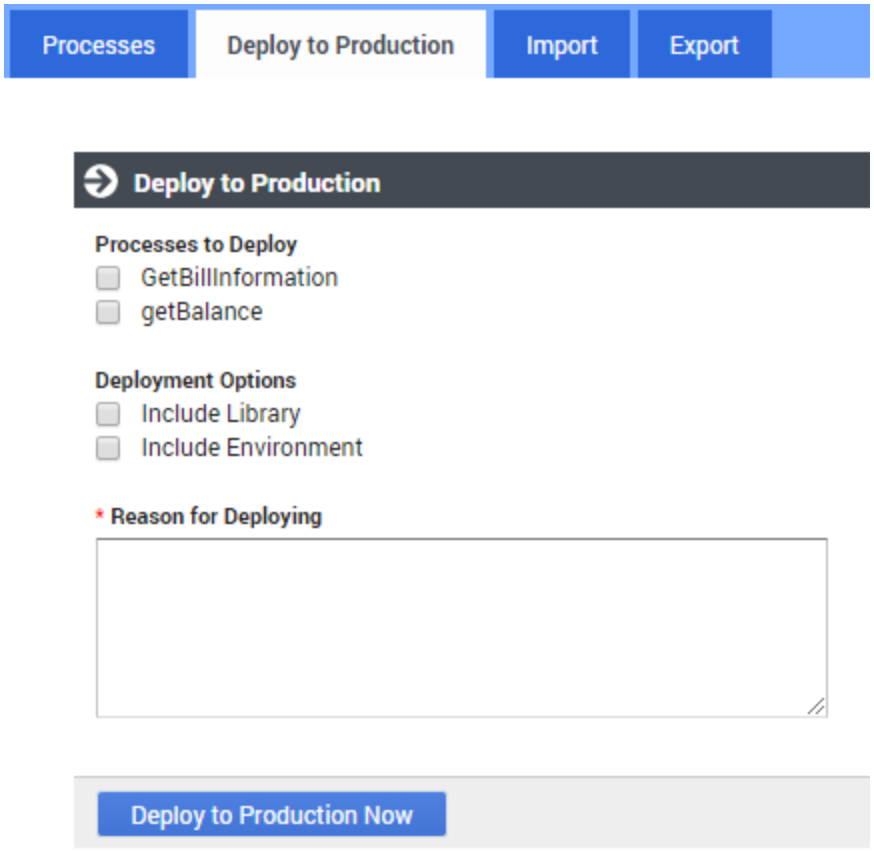
2. Specify the expected **HTTP Response Code** (the default, `200`, indicates success); and optionally, the **XPath Response Assertions**, in which known elements or variables should match or not match specified values. The variety of assertions is based on whatever data is returned by the test stubs. The Author can write as many XPath assertions as they wish to inspect the generated XML, as long as he or she knows what data is returned by a test web service for a particular input. For example:
   - **XPath** is `/response/status/text()`
   - **Match Type** is `matches`
   - **Expected Values** is `success`

Running and Evaluating a Test Case

1. Click **Run this Test**.
2. View the **Execution Results**, including the information from any Response Templates associated with this process.

Deploy to Production

After you have created the Process and tested it, it is ready to be put into the production side of the environment in which you are working. Note that when you deploy a process to production, it is not changed nor removed from the testing environment.



To deploy a process to production:

1. Select the **Processes to Deploy**. If there is more more than one listed, you can deploy one, some, or all of those that are listed.
2. **Deployment Options** give you the option to include the Library and/or Environment information that you entered on the **Processes** tab.

> **Tip**
> If you included any of the library variables or environment information in any of the processes to be deployed, you must include the corresponding deployment option here.

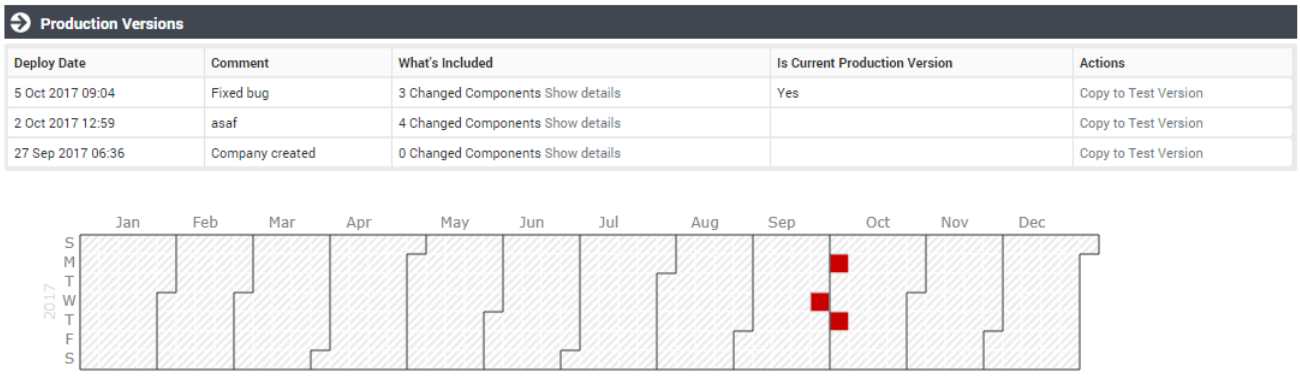
3. You must provide a **Reason for Deployment**, if for no other reason than to identify this particular deployment.

The **Production Version** section (see the diagram below) displays a history of your deployment activities, including which deployment is your **Current Production Version**.

You can also copy a deployment to a test version, effectively overwriting the test version with a copy of the deployed version.

> **Tip**
>
> If your current deployment is not operating as expected, you can copy the previously deployed version to the test side of the environment, test it to make sure that it is running properly, then redeploy this version. This version then becomes the Current Production Version, taking over this position from the more updated, but faulty version. Then, you can work to debug the latest (updated but faulty) version, and deploy it when tested.

The twelve-month calendar for the current year illustrates the distribution of all deployments throughout the current calendar year. Hovering over a red box displays the date and number of deployments on that date; clicking on the red box highlights the deployment in the table above the calendar.



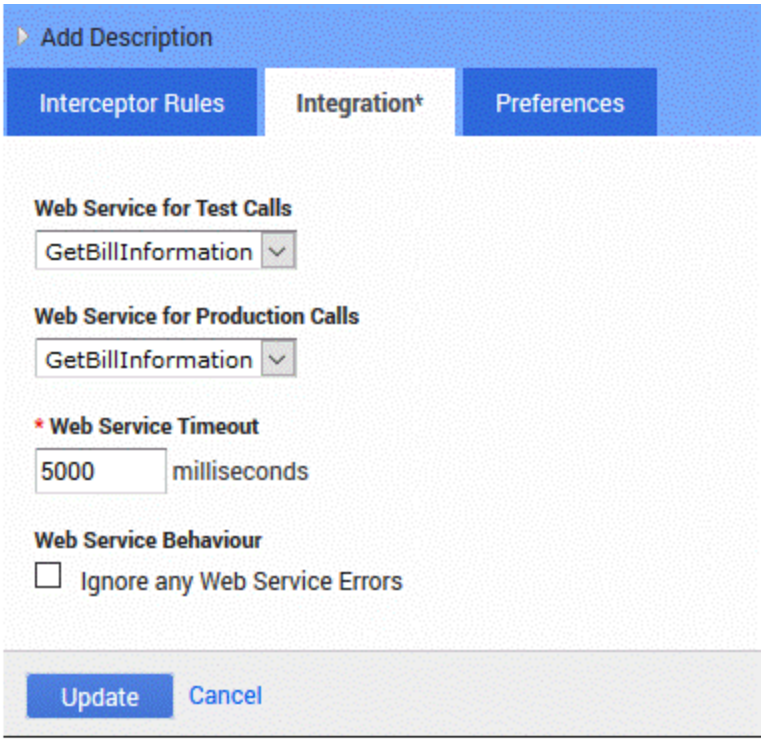
Production Versions

Deploy Date	Comment	What's Included	Is Current Production Version	Actions
5 Oct 2017 09:04	Fixed bug	3 Changed Components Show details	Yes	Copy to Test Version
2 Oct 2017 12:59	asaf	4 Changed Components Show details		Copy to Test Version
27 Sep 2017 06:36	Company created	0 Changed Components Show details		Copy to Test Version

> **Warning**
>
> The Copy-to-test feature in iHub replaces all test processes with the production processes that was in the specific deployment.

Adding Integration Processes to a Callflow

After you have created and tested an Integration Process, you can incorporate it into your callflow by adding it to an Interceptor block, as shown in the following diagram:



Open the **Integration** tab of the Interceptor block, and do the following:

1. Replace the values of the **Web Service URL for Test Calls** and **Web Service URL for Production Calls** fields with the following:

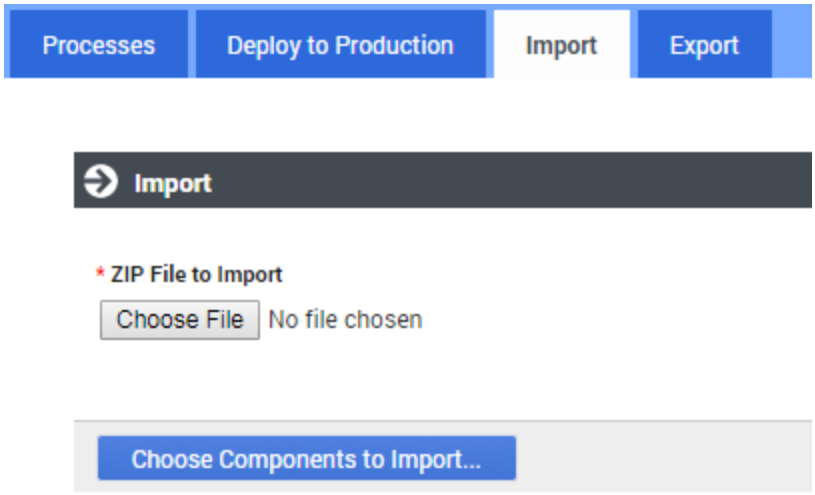
   ```
   ihub://<Process_ID>
   ```

   Where `Process_ID` is the Process UUID shown at the top of the **Processes** tab. You can also add additional parameters, if required.
2. Change the value of the **Web Service Timeout** field, if needed.

Import

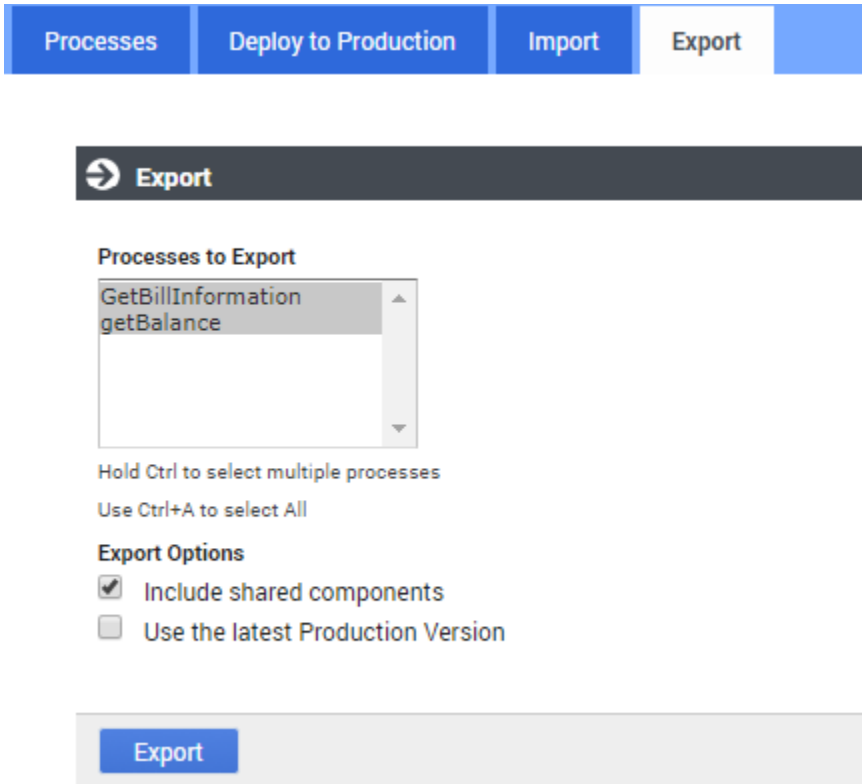
On the **Import** tab, you can import iHub Scripts from other processes in the same or a different environment. The input file must be created by using the Export tab.

After you have selected the **.zip** files for importing, click **Choose Components for Import**. You can then choose to use the file contents to create a new process, or to overwrite the corresponding parts of an existing process.



Export

On the **Export** tab, you can export processes and shared scripts to a **.zip** file. This file can be used for such things as backup or storage, or to import back into iHub, perhaps into another environment.



You must select one or more processes and/or one or more shared component to export. If any process you selected is also in production, you can also choose to export the latest production version of it instead of the test version.

When you click **Export**, the export operation creates a file called **Integration Export <datestamp> <timestamp>.zip**. This file can then be imported using the Import tab.

Scripting Commands

> **Important**
> You can download a JavaDoc version of the scripting commands here.

iHub provides the following commands for use in Process and Library Scripts:

Command	Descriptionhere
context.cacheGet(java.lang.String sKey_p)	Retrieve an item placed in the user cache.
context.cachePut(java.lang.String sKey_p, java.io.Serializable value_p, int ExpiryTimeSecs_p)	Place an item in the user cache.
context.escapeJavaScript(java.lang.String sText_p)	Free up memory used by the given JavaScript string.
context.escapeUrl(java.lang.String sText_p)	Free up memory used by the given URL.
context.escapeXml(java.lang.String sText_p)	Free up memory used by the given XML string.
context.formatCurrency(java.lang.String sCurrencyCode_p, java.math.BigDecimal currencyAmount_p)	Create a correctly formatted currency string by passing in an ISO 4217 currency code and a BigDecimal amount.
context.formatCurrency(java.lang.String sCurrencyCode_p, java.lang.String sCurrencyAmount_p)	Create a correctly formatted currency string by passing in an ISO 4217 currency code and a String amount.
context.formatDate(java.util.Date date_p)	Format a date in the standard Intelligent Automation format (yyyy-mm-dd) using the system timezone.
context.formatDate(java.util.Date date_p, java.lang.String sFormat_p)	Format a date into the standard Intelligent Automationformat (i.e yyyy-MM-dd)
context.formatDate(java.util.Date date_p, java.time.ZoneId timezone_p)	Format a date into a specified format using the system timezone.
context.formatDate(java.util.Date date_p, java.lang.String sFormat_p, java.time.ZoneId timezone_p)	Format a date into a specified format with a specified timezone.
context.getCurrencyAmount(java.lang.String sSpeechStormCurrency_p)	Retrieve the amount from an Intelligent Automation formatted currency string.
context.getCurrencyCode(java.lang.String sSpeechStormCurrency_p)	Retrieve the ISO 4217 from an Intelligent Automation formatted currency string.

Command	Descriptionhere
context.getLastBackendCallResult()	Return the result of the last backend request.
context.getRandomPercentage()	Return a random percentage value that can be used for routing a given number of calls in different directions.
context.getResponseTemplateNames()	Return the set of Response Template names that are configured in the iHub
context.getTimeZone(java.lang.String sTimeZoneName_p)	Return the ZoneId value of a specified timezone name.
context.getVariable(java.lang.String sName_p)	Return the value of a variable held in session by it's name
context.getVariableNames()	Return a Collection of the variable names currently held in session.
context.http(java.lang.String sURL_p, java.lang.String sMethod_p, java.util.List<java.lang.String> headers_p, java.util.Map<java.lang.String,java.lang.String> params_p, int iTimeoutMillis_p)	Send an HTTP request to a specified web service URL containing key/value pair parameters. Important If the specified method requires information to be sent in the body rather than in the query string, iHub automatically adds the parameters to the request body as key/value pairs. The port number listed in the HTTP Settings must be included in the **sURL_p** string, e.g., https://service.domain.com:443/application.
context.http(java.lang.String sURL_p, java.lang.String sMethod_p, java.util.List<java.lang.String> headers_p, java.lang.String sRequestBody_p, java.lang.String sContentType_p, int iTimeoutMillis_p)	Send an HTTP request to a specified web service URL containing a request body and content type. Important When a context.http call is used, an SSL handshake is performed to establish a secure connection (with TLS) with the configured host URL. The port number listed in the HTTP Settings must be included in the **sURL_p** string, e.g., https://service.domain.com:443/application.
context.logDebug(java.lang.String sMessage_p, java.lang.Object... additionalItems_p)	Write a debug statement to the logs.
context.logError(java.lang.String sMessage_p, java.lang.Object... additionalItems_p)	Write an error statement to the logs.
context.logError(java.lang.Throwable error_p, java.lang.String sMessage_p, java.lang.Object... additionalItems_p)	Write an error statement to the logs including a Throwable object.
context.logInfo(java.lang.String sMessage_p, java.lang.Object... additionalItems_p)	Write an info statement to the logs.
context.logWarning(java.lang.String sMessage_p, java.lang.Object... additionalItems_p)	Write a warning statement to the logs.
context.parseDate(java.lang.String sSpeechStormDate_p)	Parse a date in the Intelligent Automation format (yyyy-MM-dd) using the system time zone.
context.parseDate(java.lang.String sDate_p, java.lang.String sFormat_p, java.time.ZoneId timezone_p)	Parse a date in the Intelligent Automation format (yyyy-MM-dd) using the given time zone.
context.parseDate(java.lang.String sDate_p,	Parse a date in a specified format using the system

Command	Descriptionhere
java.lang.String sFormat_p)	time zone.
context.parseDate(java.lang.String sDate_p, java.lang.String sFormat_p, java.time.ZoneId timezone_p)	Parse a date with a specified format and timezone.
context.parseJSON(java.lang.String sJSON_p)	Parse a provided JSON string into a JSON object.
context.parseXML(java.lang.String sXML_p)	Parse a provided XML string into a Node object.
context.selectResponseTemplate(java.lang.String sResponseTemplateName_p)	Specify the response template to use when responding to the VUI.
context.sendEmail(java.util.List<java.lang.String> recipients_p, java.lang.String sFromAddress_p, java.lang.String sSubject_p, java.lang.String sMessage_p)	Send an email.
context.sendSMS(java.lang.String sRecipientNumber_p, java.lang.String sSenderNumber_p, java.lang.String sMessage_p)	Send an SMS.
context.setVariable(java.lang.String sName_p, java.lang.Object value_p)	Add a variable to the session.
context.sqlDelete(java.lang.String sConnectionName_p, int iTimeoutMillis_p, java.lang.String sQuery_p, java.lang.Object... parameters_p)	Perform a DELETE query in the database.
context.sqlInsert(java.lang.String sConnectionName_p, int iTimeoutMillis_p, java.lang.String sQuery_p, java.lang.Object... parameters_p)	Perform an INSERT query in the database.
context.sqlSelect(java.lang.String sConnectionName_p, int iTimeoutMillis_p, java.lang.String sQuery_p, java.lang.Object... parameters_p)	Perform a SELECT query in the database.
context.sqlStoredProcedure(java.lang.String sConnectionName_p, int iTimeoutMillis_p, java.lang.String sStoredProcedureName_p, java.lang.Object... parameters_p)	Run the specified stored procedure.
context.sqlUpdate(java.lang.String sConnectionName_p, int iTimeoutMillis_p, java.lang.String sQuery_p, java.lang.Object... parameters_p)	Perform a UPDATE query in the database.
context.unescapeJavaScript(java.lang.String sText_p)	Allocate memory for the given JavaScript string.
context.sqlDelete(String sConnectionName_p, int iTimeoutMillis_p, String sQuery_p, List<Object> parameters_p)	Perform a DELETE query in the database.
context.sqlInsert(String sConnectionName_p, int iTimeoutMillis_p, String sQuery_p, List<Object> parameters_p)	Perform an INSERT query in the database.
context.sqlSelect(String sConnectionName_p, int iTimeoutMillis_p, String sQuery_p, List<Object> parameters_p)	Perform a SELECT query in the database.

Command	Descriptionhere
context.sqlUpdate(String sConnectionName_p, int iTimeoutMillis_p, String sQuery_p, List<Object> parameters_p)	Perform an UPDATE query in the database
context.sqlStoredProcedure(String sConnectionName_p, int iTimeoutMillis_p, String sStoredProcedureName_p, List<Object> parameters_p)	Run the specified stored procedure.