Genesys App Automation Platform Help

Integrating GAAP and Customer Environments

12/13/2025

Contents

- 1 Integrating GAAP and Customer Environments
  - 1.1 iHub Interface
  - 1.2 Processes
  - 1.3 Deploy to Production
  - 1.4 Import
  - 1.5 Export
  - 1.6 Scripting Commands

Integrating GAAP and Customer Environments

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

iHub Interface

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

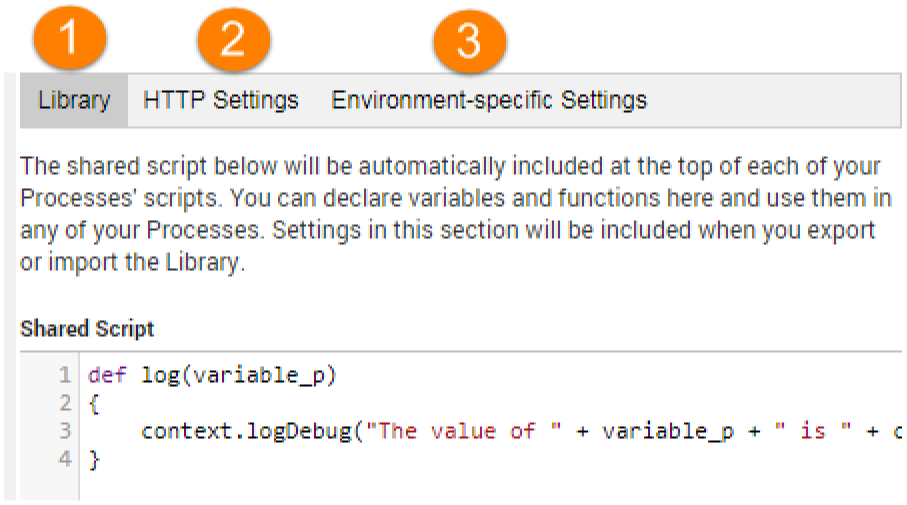
iHub has four tabs, as follows:

- **Processes**—In this tab, you define the Integration Processes that will handle HTTP(S) requests that are received from the GAAP 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

You can define common properties, such as variables, functions, and environment settings, that can be used for all of your processes as required. In addition, you can specify the host on which the processes run and to which the VUI must connect, even securing the connection if necessary. You can do all this on the Processes tab, as indicated in the diagram below. The numbers correspond to the tab descriptions that immediately follow the diagram.



The shared script below will be automatically included at the top of each of your Processes' scripts. You can declare variables and functions here and use them in any of your Processes. Settings in this section will be included when you export or import the Library.

Shared Script

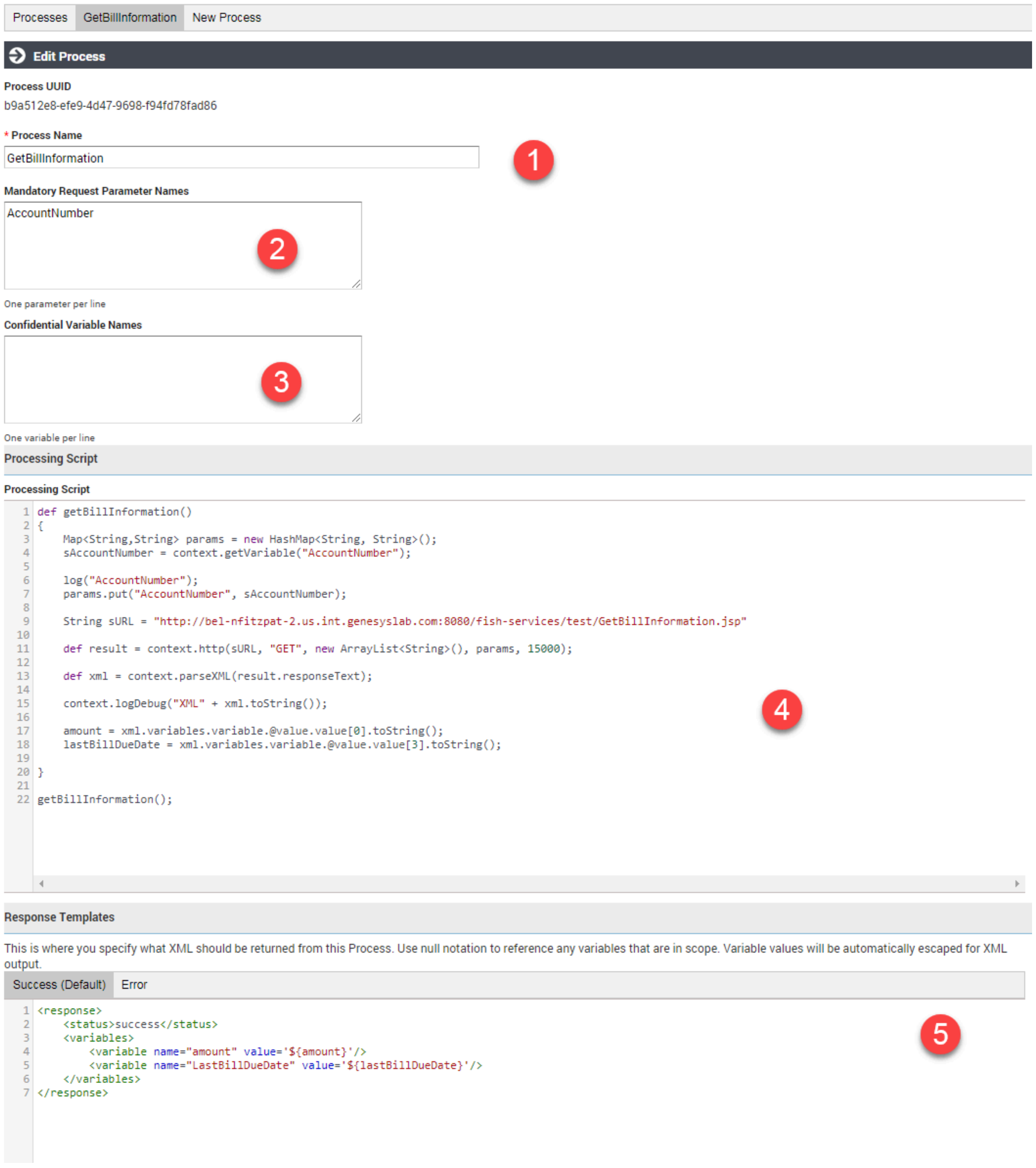
```
def log(variable_p)
{
    context.logDebug("The value of " + variable_p + " is " + c
}
```

1. Create a library 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.
2. Identify any Company-specific configuration information that would apply to a given GAAP callflow engine installation, such as HTTP. If you want to secure the connection using TLS (or SSL), and/or use client-side authentication, enter the appropriate information here. For more information about these security concepts, refer to the *Genesys Security Deployment Guide*.
3. Define any settings that are specific to the environment in which you are working, such as Lab, Test, or Production.

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

Creating a Process

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



Process UUID
b9a512e8-efe9-4d47-9698-f94fd78fad86

* Process Name
GetBillInformation

Mandatory Request Parameter Names
AccountNumber

One parameter per line

Confidential Variable Names

One variable per line

Processing Script

Processing Script

```
def getBillInformation()
{
    Map<String,String> params = new HashMap<String, String>();
    sAccountNumber = context.getVariable("AccountNumber");

    log("AccountNumber");
    params.put("AccountNumber", sAccountNumber);

    String sURL = "http://bel-nfitzpat-2.us.int.genesyslab.com:8080/fish-services/test/GetBillInformation.jsp"

    def result = context.http(sURL, "GET", new ArrayList<String>(), params, 15000);

    def xml = context.parseXML(result.responseText);

    context.logDebug("XML" + xml.toString());

    amount = xml.variables.variable.@value.value[0].toString();
    lastBillDueDate = xml.variables.variable.@value.value[3].toString();

}

getBillInformation();
```

Response Templates

This is where you specify what XML should be returned from this Process. Use null notation to reference any variables that are in scope. Variable values will be automatically escaped for XML output.

Success (Default) | Error

```
<response>
    <status>success</status>
    <variables>
        <variable name="amount" value='${amount}'/>
        <variable name="LastBillDueDate" value='${lastBillDueDate}'/>
    </variables>
</response>
```

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**. GAAP 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 GAAP, 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.

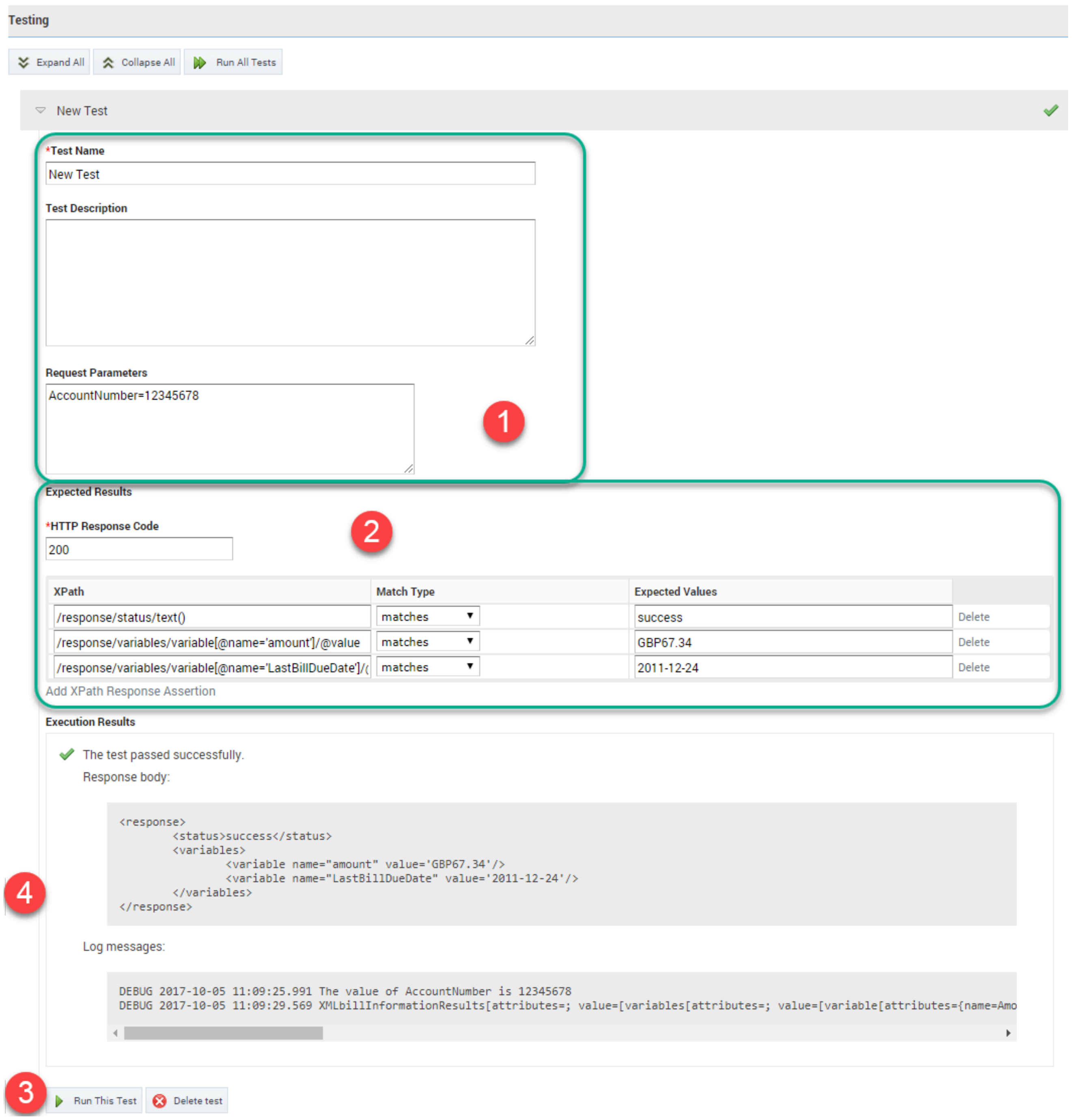
Testing a Process

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.



*Test Name: New Test

Test Description:

Request Parameters: AccountNumber=12345678

Expected Results

*HTTP Response Code: 200

XPath	Match Type	Expected Values	
/response/status/text()	matches	success	Delete
/response/variables/variable[@name='amount']/@value	matches	GBP67.34	Delete
/response/variables/variable[@name='LastBillDueDate']/(	matches	2011-12-24	Delete

Add XPath Response Assertion

Execution Results

The test passed successfully.

Response body:

```
<response>
        <status>success</status>
        <variables>
                <variable name="amount" value='GBP67.34'/>
                <variable name="LastBillDueDate" value='2011-12-24'/>
        </variables>
</response>
```

Log messages:

```
DEBUG 2017-10-05 11:09:25.991 The value of AccountNumber is 12345678
DEBUG 2017-10-05 11:09:29.569 XMLbillInformationResults[attributes=; value=[variables[attributes=; value=[variable[attributes={name=Amo
```

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.

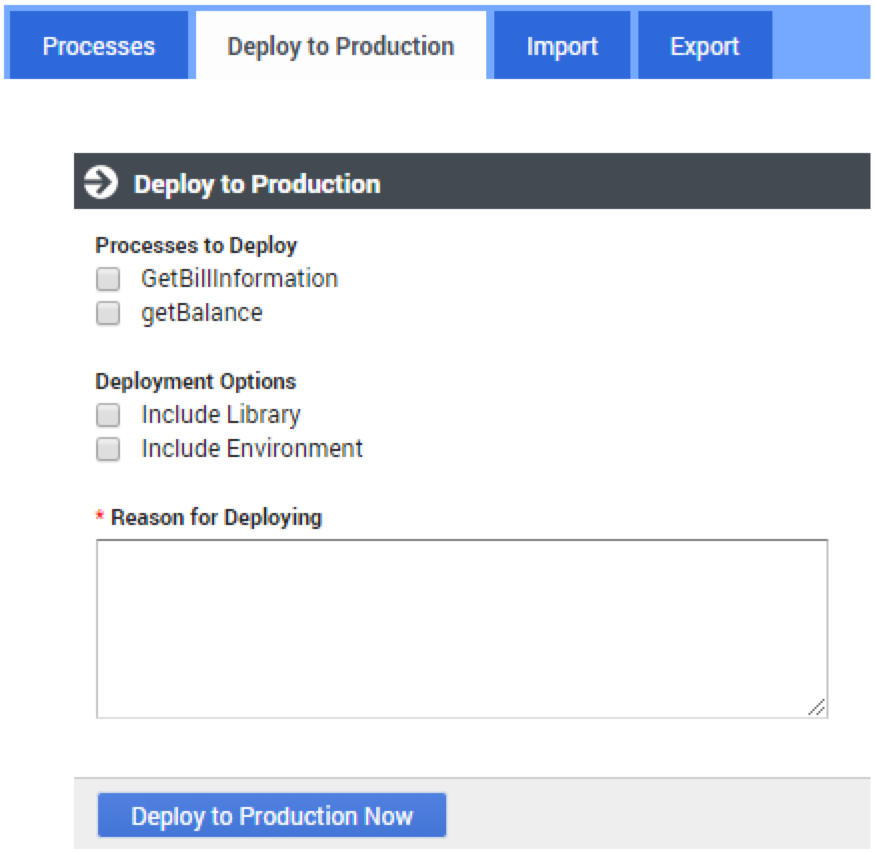
2. Specify the expected **HTTP Response Code** (the default, `200`, indicates success); and optionally, the **XPath Response Assertions**, in which known elements or variables should match or not match specified values. The variety of assertions is based on whatever data is returned by the test stubs. 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.



Deploy to Production

Processes to Deploy

- ☐ GetBillInformation
- ☐ getBalance

Deployment Options

- ☐ Include Library
- ☐ Include Environment

* Reason for Deploying

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.

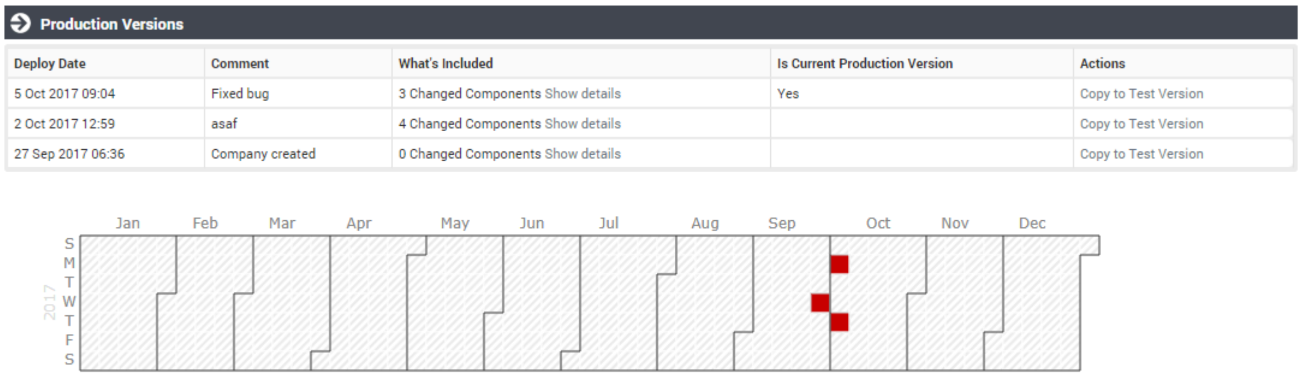
The **Production Version** section (see the diagram below) displays a history of your deployment activities, including which deployment is your **Current Production Version**.

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.

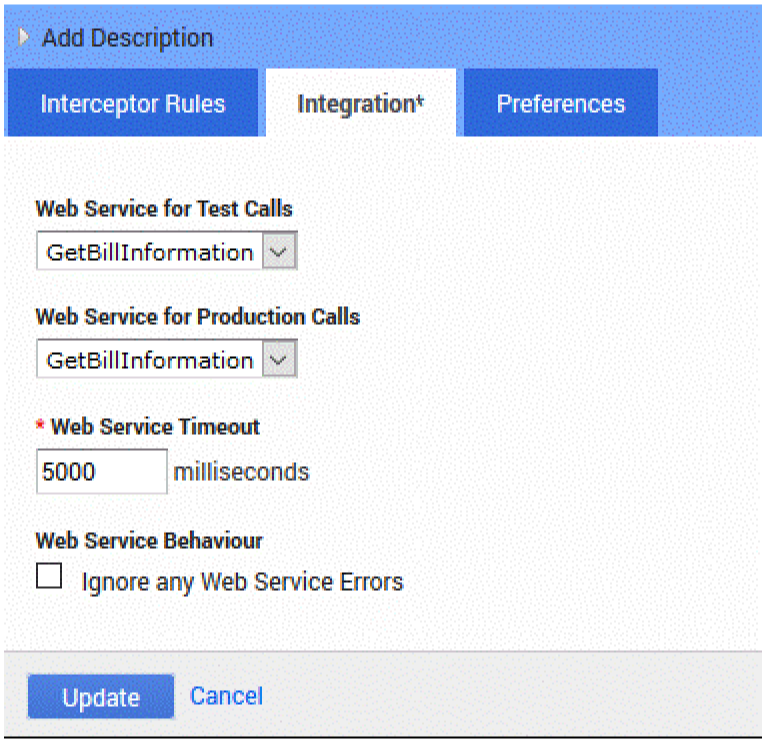


Production Versions

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

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:



Web Service for Test Calls
GetBillInformation

Web Service for Production Calls
GetBillInformation

* Web Service Timeout
5000 milliseconds

Web Service Behaviour
☐ Ignore any Web Service Errors

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>[/<name>]
   ```

   where:

   - `Process_ID` is the Process UUID shown at the top of the **Processes** tab.
   - name (optional) is the name of the process.

   Important

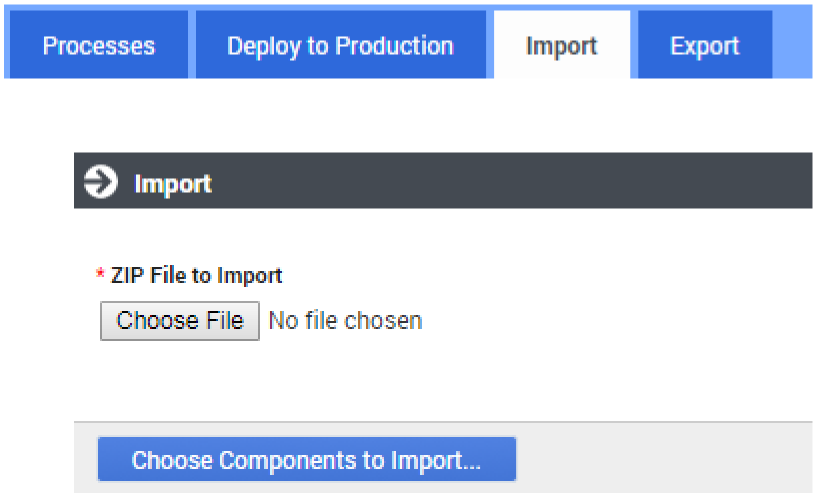
   If you you using any of the pre-built modules (Business Process, Utility Module, Security Module), they will not appear in the **Web Service URL for Test Calls** and **Web Service URL for Production Calls** fields. Instead, just enter their URL in the same format as above.

2. Change the value of the **Web Service Timeout** field, if needed.

Import

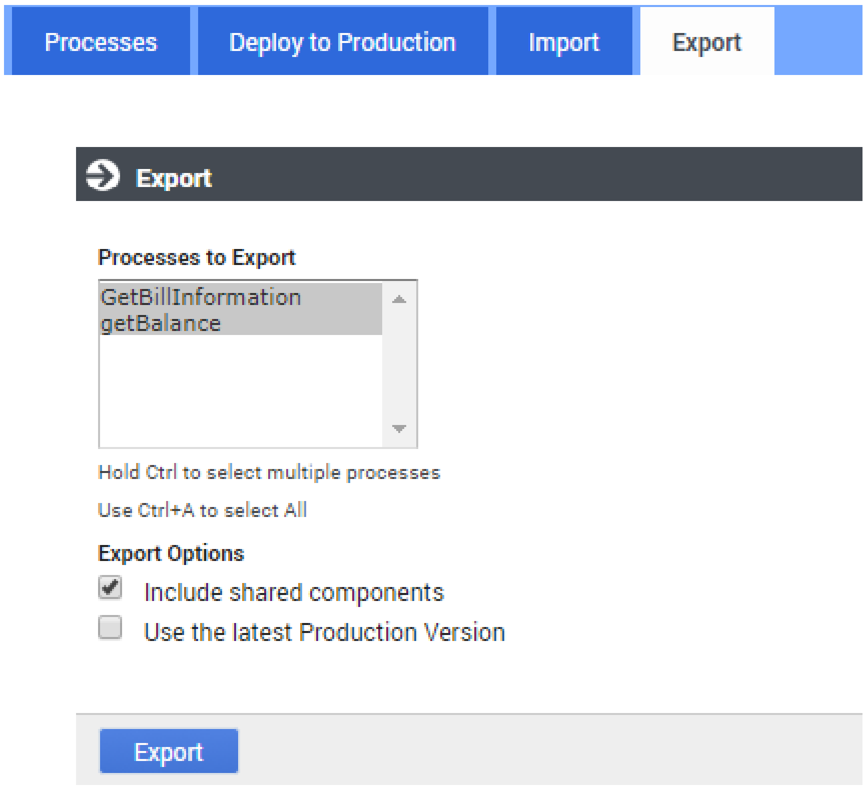
On the **Import** tab, you can import iHub Scripts from other processes in the same or a different environment. The input file must be created by using the Export tab.

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.



Processes to Export

GetBillInformation
getBalance

Hold Ctrl to select multiple processes

Use Ctrl+A to select All

Export Options

- [✓] Include shared components
- [ ] Use the latest Production Version

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

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

Command	Description
context.cacheGet(String)	Retrieve an item placed in the user cache.
context.cachePut(String, Serializable, int)	Place an item in the user cache.
context.escapeJavaScript(String)	Free up memory used by the given JavaScript string.
context.escapeUrl(String)	Free up memory used by the given URL.
context.escapeXml(String)	Free up memory used by the given XML string.
context.formatCurrency(String, String)	Create a correctly formatted currency String by passing in an ISO 4217 currency code and a String

Command	Description
	amount.
context.formatCurrency(String, BigDecimal)	Create a correctly formatted currency String by passing in an ISO 4217 currency code and a BigDecimal amount.
context.formatDate(Date)	Convert a date format to the standard GAAP format (yyyy-mm-dd) based on the system timezone.
context.formatDate(Date, String)	Convert a date format to the specified format.
context.formatDate(Date, ZoneId)	Format a date into a specified format using the system timezone.
context.formatDate(Date, String, ZoneId)	Format a date into a specified format with a specified timezone.
context.getCurrencyAmount(String)	Retrieve the amount from a GAAP formatted currency String.
context.getCurrencyCode(String)	Retrieve the ISO 4217 from a GAAP formatted currency String.
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(String)	Return the ZoneId value of a specified timezone name.
context.getVariable(String)	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(String, String, List<String>, String, String, int)	Send an HTTP request to a specified web service URL containing a request body and content type.
context.http(String, String, List<String>, Map<String, String>, int)	Send an HTTP request to a specified web service URL containing key/value pair parameters.
context.logDebug(String, Object...)	Write a debug statement to the logs.
context.logError(String, Object...)	Write an error statement to the logs.
context.logError(Throwable, String, Object...)	Write an error statement to the logs including a Throwable object.
context.logInfo(String, Object...)	Write an info statement to the logs.
context.logWarning(String, Object...)	Write a warning statement to the logs.
context.parseDate(String)	Parse a date in the standard GAAP format (yyyy-MM-dd) based on the time zone used by the system.
context.parseDate(String, ZoneId)	Parse a date in the standard GAAP format (yyyy-MM-dd) based on the given time zone.
context.parseDate(String, String)	Parse a date in a specified format, based on the time zone used by the system..

Command	Description
context.parseDate(String, String, ZoneId)	Parse a date with a specified format and timezone.
context.parseJSON(String)	Parse a provided JSON String into a JSON object.
context.parseXML(String)	Parse a provided XML String into a Node object.
context.selectResponseTemplate(String)	Specify the response template to use when responding to the VUI.
context.sendEmail(List<String>, String, String, String)	Send an email.
context.sendSMS(String, String, String)	Send an SMS.
context.setVariable(String, Object)	Add a variable to the session.
context.unescapeJavaScript(String)	Allocate memory for the given JavaScript string.
context.unescapeUrl(String)	Allocate memory for the given URL.
context.unescapeXml(String)	Allocate memory for the given XML string.