




# **ix** SCRIPTING WITH GROOVY

UNITED PLANET INTREXX XTREME  
RELEASE 4.5








## Contents


<b>1. Introduction</b> .....	<b>3</b>
<b>2. Composing and executing scripts</b> .....	<b>3</b>
2.1. Set user permissions .....	3
2.2. Execute script within the Process Manager .....	3
2.3. Executing a script within a web service .....	5
<b>3. Basics</b> .....	<b>6</b>
3.1. Objects available for use .....	6
3.2. Calling up <code>g_ws.invoke()</code> .....	6
3.3. Available namespaces .....	6
3.3.1. Overview .....	6
3.3.2. Read and write .....	9
<b>4. Objects</b> .....	<b>9</b>
4.1. <code>g_dbConnections</code> .....	10
4.2. <code>g_dbQuery</code> .....	10
4.3. <code>g_record</code> .....	10
4.4. <code>g_rtCache</code> .....	11
4.5. <code>g_session</code> .....	11
4.6. <code>g_request</code> .....	12
<b>5. @Scriptable classes</b> .....	<b>12</b>
<b>6. Check web service parameters</b> .....	<b>12</b>
6.1. Check request parameters .....	13
6.2. Check return parameters .....	13
<b>7. Exception handling</b> .....	<b>14</b>

## Writing Conventions

In this document, text passages will be displayed in *Italics* when they refer to settings in the displayed dialogs. Menu items that are available in context menus are also always reachable from the main menu. Main menu items will not be described, unless they are not available from the context menu. A description of the general main menu items can be found in the  *Center* handbook. Programming code in the text will be displayed in the Courier font.

`<xtreme>` refers in the following to your Intrex Xtreme installation path, for example in Windows `C:\xtreme\`, or in Linux `/opt/xtreme/`. The following symbols will be used to designate special kinds of information:

-  Important notes
-  Tips and background information
-  References to additional information in an Intrex Xtreme handbook
-  Directories
-  URLs
-  Buttons in dialogs or assistants
-  Context menus are designated with this symbol and can be opened by clicking with the right mouse button on the element described.

Passages with the  **[Copy-Paste]** tip can be directly used from this document in order to avoid typing errors.

## Previous Knowledge

In order to understand this document, knowledge in working with Intrex Xtreme, as well as in programming with Java and/or Groovy, is required.

**1. Introduction**

With Version 4.5, Intrex Xtreme offers you a multitude of new possibilities for server-side scripting. With the integration of the object-oriented script language Groovy, it is now possible to create individual scripts when working with web services or within the Process Manager, thereby allowing a panoply of innovative processes to be realized.

The Intrex Java-API documenty for Groovy objects can be found at



 [www.intrex.com/groovydoc](http://www.intrex.com/groovydoc)

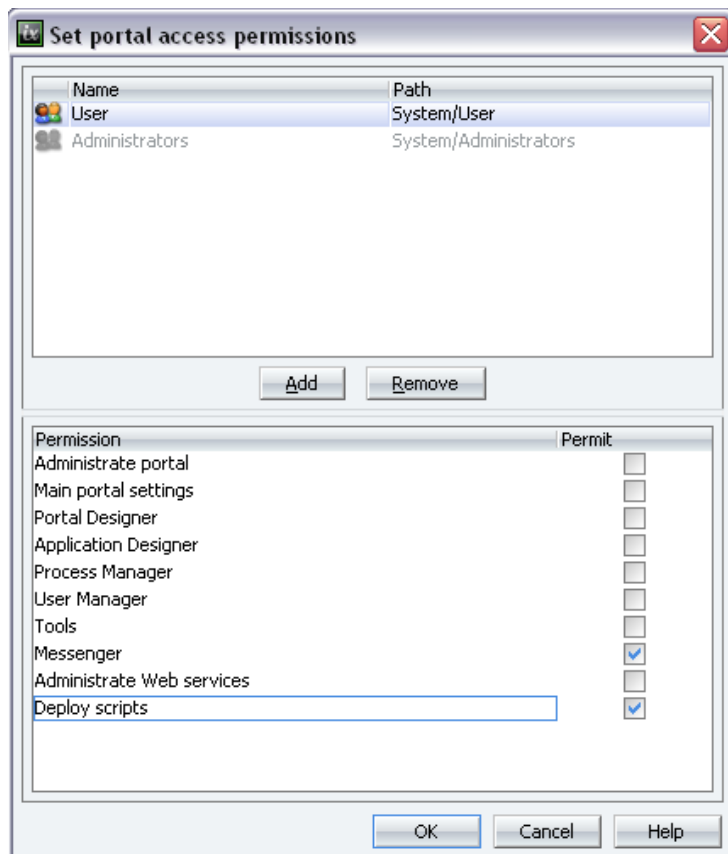
You can find additional information, a number of tutorials, and an API overview for Groovy at

 <http://groovy.codehaus.org/>

**2. Composing and executing scripts**

**2.1. Set user permissions**

In order to create Groovy scripts in Intrex Xtreme, a user requires corresponding permissions. Administrators can assign the permissions as follows: in the Intrex Center area of the Portal Manager, select the menu item  *Extras*  *Portal Access Permissions*. The corresponding permission can be set for users and/or user groups by selecting the checkbox for *Deploy scripts*.



**2.2. Execute script within the Process Manager**

In order to execute a script, various possibilities are available. One possibility is to run a script within the Process Manager.

This makes it possible to react to data group or timer events. The new symbol *Groovy Script* exists for this purpose.



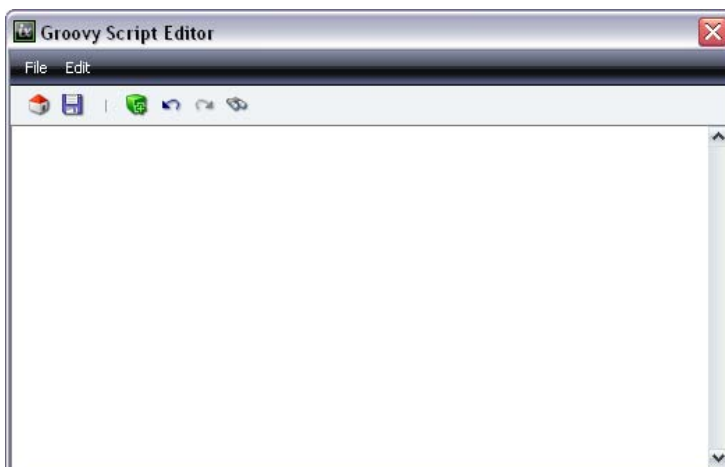
With this action, one proceeds similarly to other actions in the Process Manager. Create an event that will start the process. Additionally, a condition will be created, and lastly the Groovy action to be executed will be defined.



Double-clicking on the Groovy action will open the window to *Edit Action*.



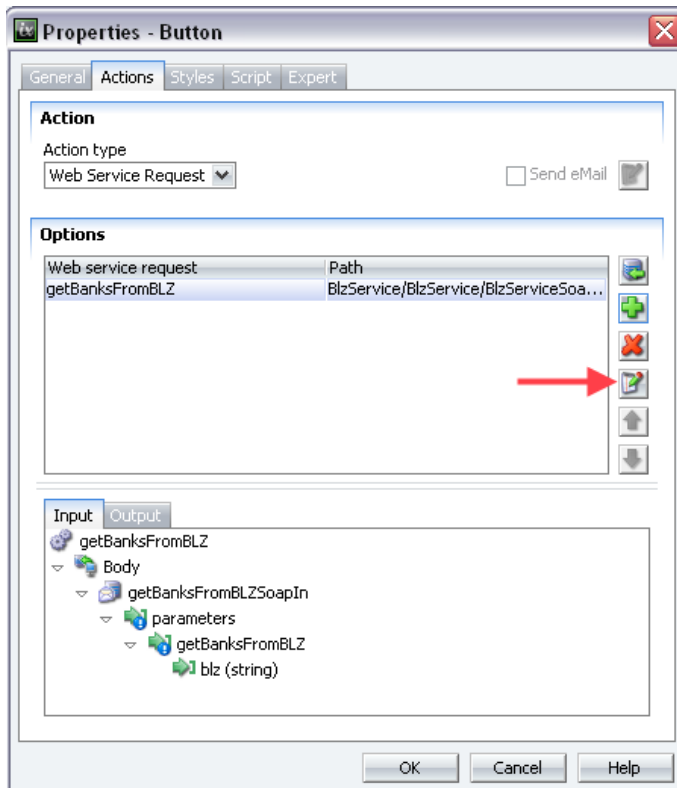
Clicking on the Groovy script button will open the Groovy script editor. A script of your choice can be entered now.



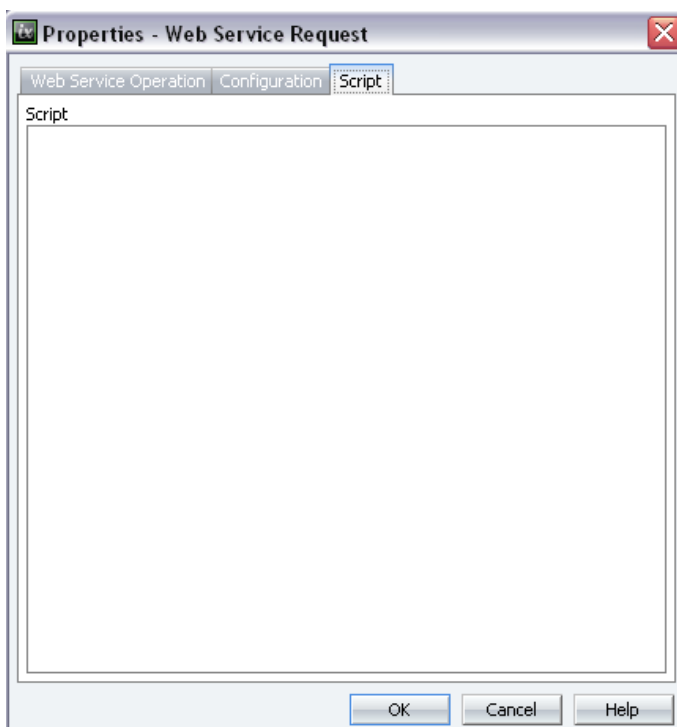
The possibility to enter scripts within a web service action in the Process Manager exists as well.

**2.3. Executing a script within a web service**

Scripts can not just be entered within the Process Manager, rather, they can also be executed within a web service. To do so, a previously defined web service must be entered into the Intrex Xtreme Application Designer. The button to *Edit Web Service Request* is located in the properties dialog.



The *Script* tab is located in the subsequently opening properties dialog of the web service. In this location, the script will be entered that will be called when a web service is run.



### 3. Basics

#### 3.1. Objects available for use

In the underlying script context, the following objects are available for use.

Object Name	Description
<i>g_ctx</i>	This object contains all variables available in the context. The variables that are available will be defined via the namespace entered. (For information on this topic, see <i>Available namespaces</i> chapter)
<i>g_ws</i>	The global web service object.
<i>g_log</i>	The logger object. With this, it is possible to write outputs to a logfile ( <i>portal.log</i> ). <code>g_log.info("TEXT") //TEXT</code> will be written to <i>portal.log</i> .
<i>g_dbConnections</i>	The database object. This enables both access to the database connection used in Intrex Xtreme and access to external data sources.
<i>g_session</i>	The global session object. Contains the session timeout, creation date of the session, and other information.
<i>g_record</i>	Object for data records. Contains record ID, parent ID (if it exists), corresponding application GUID, and other information.
<i>g_request</i>	Object for the context of the request values.
<i>g_dbQuery</i>	Object for the preparation and execution of SQL statements.

#### 3.2. Calling up *g\_ws.invoke()*

Within Intrex Xtreme, calling up a web service will be implemented with the method *g\_ws.invoke()*:

```

...           \\ Can be individual code
g_ws.invoke() \\ Call for the global web service object
...           \\ Can be individual code

```

The web service will first be invoked when the method call of the global web service object is run. This makes it possible to execute individual code before and after a web service call. The monitoring of input and output parameters of a web service are named as examples here.

### 3.3. Available namespaces

#### 3.3.1. Overview

The previously mentioned object *g\_ctx* and its variables are available in the entire context, and thereby for all web service calls and scripts belonging to them. The variables available within an object for use can be entered via the integrated fixed namespaces. In order to receive access to such a namespace, the following definition of an access method is necessary:

```
g_ctx.getNamespaceName()
```

As *NamespaceName* one of the following namespaces must be entered. An overview can also be found in the Intrex Java-API documentation in class

```
de.uplanet.util.VARIABLE_NAMESPACE
```

### Namespace

#### ConstantVars

<http://schemas.unitedplanet.de/intrexx/variable/named-constant/>

### Description

Namespace for named constants or static values.  
Read access is possible.

### Namespace

#### UserVars

<http://schemas.unitedplanet.de/intrexx/variable/system-value/user/>

### Description

Namespace for properties of the currently logged in user.  
Read access is possible.

### Namespace

#### SessionVars

<http://schemas.unitedplanet.de/intrexx/variable/session/>

### Description

Namespace for variables for the current session.  
Read and write access is possible for:

- Language
- Layout
- Last requested URL

Only read access is possible for:

- Session ID
- Anonymous flag
- Session timeout
- Creation date of session

### Namespace

#### RequestVars

<http://schemas.unitedplanet.de/intrexx/variable/request/>

### Description

Namespace for request variables.  
Read access is possible.

### Namespace

SrcPageVars

<http://schemas.unitedplanet.de/intrexx/variable/source-page-value/>

### Description

Namespace for variables of the source page being queried.  
Read access is possible.

### Namespace

BpeeVars

<http://schemas.unitedplanet.de/intrexx/variable/bpee/>

### Description

Namespace for the variables in Bpee context.  
Read access is possible.

### Namespace

BpeeDynamicVars

<http://schemas.unitedplanet.de/intrexx/variable/bpee-dynamic/>

### Description

Namespace for addressing return values in Bpee context.  
Read and write access is possible.

### Namespace

TempVars

<http://schemas.unitedplanet.de/intrexx/variable/temporary/>

### Description

Namespace for temporary variables.  
Read and write access is possible.

**Namespace**

DatafieldVars

```
http://schemas.unitedplanet.de/intrex/variable/datafield/
```

**Description**

Namespace for values from data fields. Available in processes.  
Read and write access is possible.

**3.3.2. Read and write**

In order to read values of a variable from a namespace, the following command can be used:

```
g_ctx.namespaceName.localIdentifier
```

Under `localIdentifier` you will enter the value of the variables that you wish to read out.

You can find out which variables will be read out, and if possible, those that can be described, by using

```
g_log.info(g_ctx.dump())
```

or

```
g_ctx.orderedVariableKeySet.each {g_log.info(it)}
```

to write the execution context to `portal.log` or output this information in the console. You will receive here an overview of the namespaces with their respective assigned variables.



Please note that no access method exists for the Velocity namespace. Changes to the Velocity context should not be executed. United Planet takes no responsibility for changes you make here.

If you are working with a variable from a namespace with write access, the variables can be assigned new values.

```
g_ctx.getNamespaceName().putAt("localIdentifier ", newValue)
```

or

```
g_ctx.namespaceName.localIdentifier = newValue
```

**4. Objects**

In the following, the objects and methods that are available for use will be listed and explained.

Following these will be a reference to in which class all methods of the corresponding object can be found.

#### 4.1. **g\_dbConnections**

This object is available within a web service script and within the Process Manager for use.

Command	Description
<code>g_dbConnections.getSystemConnection</code>	Returns the Intrex system connection.
<code>g_dbConnections["ConnectionName"]</code>	Returns the data source connection entered to the Intrex Integration Center under the name <i>ConnectionName</i> .

#### 4.2. **g\_dbQuery**

This object is available within a web service script and within the Process Manager for use.

The functions of the `g_dbQuery` object that can be used can be found in the Intrex Java API documentation in the classes

```
de.uplanet.lucy.server.util.db.DbQuery
```

and

```
de.uplanet.lucy.server.util.db.DbPreparedStatement
```

Example:

```
def l_conn = g_dbConnections.systemConnection
def l_stmtSelect = g_dbQuery.prepare(l_conn, "SELECT ? FROM XTABLE05D1B1B9")
l_stmtSelect.setString(1, "ID")
l_stmtSelect.executeQuery()
```

As result of `executeQuery()`, one receives a result set, which can be further edited with methods such as *each*, *every*, etc. within brackets.

```
l_rsResult = l_stmtSelect.executeQuery()
l_rsResult.each
{
    println it.value(1)
}
```

A result set consists of result rows. These result lines and the corresponding data do not only exist at the point in time of their processing, rather they exist also after they are processed and thereby can also be used at a later time as well if needed.

It is not necessary to close the opened connections and queries with `close()` at the end of a script. This action will be directly accomplished by Intrex.

#### 4.3. **g\_record**

This object is available from within the Process Manager for use.

You can gain access to a current data record, by calling up `g_record` in connection with the GUID of a data field (`g_record["LINK GUID"]`).

With the Process Manager, you can show the available data fields of the data record by clicking on the record with the right mouse button. After selecting a field, `g_record["LINK GUID"]` will be automatically generated.

The functions of the `g_record` object that can be used can be found in the Intrex Java API documentation in the class

```
de.uplanet.lucy.server.workflow.GroovyRecord
```

Example:

```
//Read out the LID of a data record  
l_vhLid = g_record["B4BBF891435C538329D66994CEF99CF0F81B164F"].value
```

#### 4.4. **g\_rtCache**

This object is available within a web service script and within the Process Manager for use.

The Groovy RtCache is a central cache in which a multitude of information about data groups, applications, fields, controls, references, and so on are available.

The functions of the `g_rtCache` object available for use can be found in the Intrex Java API documentation in the class

```
de.uplanet.lucy.server.businesslogic.GroovyRtCache
```

Example:

```
//Returns an overview of the names of all applications  
g_rtCache.getApplications().each{g_log.info it.titles}
```

#### 4.5. **g\_session**

This object is available for use within the Process Manager.

This makes information about the currently active session available for use, such as *CreationTime*, *CurrentUser*, etc.

The available functions of the `g_session` object can be found in the Intrex Java API documentation in the class

```
de.uplanet.lucy.server.portalserver.GroovySession
```

Example:

```
//Returns the last requested URL  
l_value = g_ctx.getSessionVars().getAt("rc_LastRequestedURL")  
  
//Write value  
g_ctx.getSessionVars().putAt("rc_LastRequestedURL", "www.intrex.com")
```

#### 4.6. **g\_request**

This object is available within the Process Manager for use.

It provides information about the currently available request variables.

You can find the available functions of the `g_request` object in the Intrex Java API documentation in the class

```
de.uplanet.lucy.server.connector.GroovyServerBridgeRequest
```

Example:

```
//Set request parameter
g_request.setProperty("rq_myParam", "Groovy")

//Read request parameter
def l_paramValue = g_request.rq_myParam
```

#### 5. **@Scriptable classes**

The previously presented objects use classes that have been provided for use with the help of the attribute *@Scriptable* for Groovy scripts.

If you wish to make additional classes from United Planet available within Groovy scripts, you can achieve this with manual adjustments to the folder

```
<xtreme>/org/<portalname>/internal/cfg/scripting/
```

Due to security concerns and in order to maintain functionality upon update, only classes whose names begin with *de.uplanet.* can be enabled.

If you possess additional uniquely created Java classes, they must not be specially enabled.

To enable classes, create the file *scripting.cfg* in this folder. The inclusion of classes takes place according to the pattern in the *scripting.xsd* file in the same folder.



Please note that this manual adjustment takes place at your own risk. United Planet GmbH takes no responsibility for damages and problems that occur in connection with individually enabled Scriptable classes.

Example for enabling a class:

```
<?xml version="1.0" encoding="UTF-8"?>
<scripting
  xmlns="urn:schemas-unitedplanet-de:lucy:server:scripting"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:schemas-unitedplanet-de:lucy:server:scripting
  scripting.xsd">

  <scriptable name="de.uplanet.lucy.server.exampleClass"/>

</scripting>
```

The difference to methods of callables is that through the Scriptable enabling process, it will be input which classes within a Groovy script are allowed to be instanced. In contrast to this, with *callables.cfg*, it will be defined which objects can be prepared through the creation of the Velocity context and are available at any time directly via their corresponding variable names for use.

#### 6. **Check web service parameters**

A possible area of use is to test parameters that will be transferred to a web service. Do the parameters fulfill the required conditions like length, format, or size? What should take place if a condition is not fulfilled?

This makes it possible to implement individual ways of dealing with errors when they occur, and not to be referred to the error messages of the web service alone.

In the following, a simple example should show how an input parameter can be checked and, depending on the result, call up a web service or not.

The *Currency Convertor* web service will be used.

 <http://www.webservices.net/CurrencyConvertor.asmx?wsdl>

### 6.1. Check request parameters

This web service expects two currencies as input and delivers as return value the current exchange rate.

In order to only call up the web service if a correct input is available, the properties of the entered inputs will be researched in advance.

The input fields of a web service can be found in the namespace *Request-Vars*. The return value can be found in the current BPEE context.


Replace the name displaced in italics with the correct name of your input fields and your view field.

```
l_currFrom = g_ctx.requestVars.textcontrolD72A9620
l_currTo = g_ctx.requestVars.textcontrol801FB1F7

if(l_currFrom != l_currFrom.toUpperCase() ||
   l_currTo != l_currTo.toUpperCase())
{
  g_ctx.bpeeVars.textvcontrol72EF4A0B = "Please input only uppercase letters"
}
else
{
  g_ws.invoke()
}
```

In this case, it will be checked whether the currencies have been entered in capital letters, as the web service requires uppercase. If an incorrect entry is made, an error message will be output. The output will be sent to the location in the field where the return value of the web service would be sent.

Additional checks, such as only allowing specific characters as inputs (with RegEx) or similar, are possible as well and can be combined according to your requirements.

 Please note that at present, only edit and view fields can be described that were defined as output field in the web service request. Filling an unconnected field is not yet possible at the present time.

### 6.2. Check return parameters

If the web service has been correctly invoked, it is possible as well to analyze the return value and execute specific reactions, depending on value.

As an example, an appropriate text response should be output, depending on the exchange value:

```
if(g_ctx.bpeeVars.textvcontrol72EF4A0B.value < 0.5)
{
  g_ctx.bpeeVars.textvcontrol72EF4A0B =
  "Less than 0.5: " + g_ctx.bpeeVars.textvcontrol72EF4A0B.value
}
else
{
  g_ctx.bpeeVars.textvcontrol72EF4A0B =
  "Larger than 0.5: " + g_ctx.bpeeVars.textvcontrol72EF4A0B.value
}
```

**7. Exception handling**

The exception handling in Groovy is to be used in the same way as it is used in Java. Try-catch-finally constructs can be created in both.

With these, individual error routines can be realized, just as in programming with Java.

In the case of an incorrect input of a web service parameter, you will receive an appropriate error message from Intrex in your portal.

In order to anticipate this error handling, the script will be expanded with a try-catch-block and an individually defined error message will be output to the user.

```
try
{
    g_ws.invoke()
    ...
    ... //additional code
    ...
}
catch(Exception e)
{
    g_ctx.bpeeVars.textvcontrol172EF4A0B = "An error has occurred."
}
```



Please note that currently an individually generated error message can only be written to a created view field. Responses to users with the help of a popup or a tooltip are not possible.