Home

A very short introduction

Since SIL has become more important and it was included in all our scripted fields and plugins, this has become be the reference space for all those add-ons using the language.

All JIRA SIL routines are also listed here (including some of those added by the other add-ons). Please check the availability of each routine and the plugin in which it appears.

Navigate space

What's new in SIL 3.0

Following our initial plan, we've managed to include some pretty awesome features for SIL 3.0.

Defining global variables

Back in SIL 2.x, the scripts had a standard structure. Inclusions, function definitions and then the actual code. This made it hard for user-defined functions to use global variables. Due to the standard structure, one could not define variables before function definitions, so functions had to take some additional parameters that did not really change from one call to another.

In order to address this, the new syntax allows for variables to be defined **before** the function definitions, but no other code though.

```
number HOURS_PER_DAY = 8;
number DAYS_PER_WEEK = 5;
function hourPerWeek() {
     return HOURS_PER_DAY * DAYS_PER_WEEK;
}
```

Constants

Even though one may now be able to define some global variables, that does not guarantee that the values of these variables are not (accidentally) modified.

To lock the value in place and prevent future modifications, one can now define the variable as constant.

```
const number HOURS_PER_DAY = 8;
```

In the case of arrays and structures, this also reflects to inner elements and fields.

```
const string [][] matrix = { {"11", "12"}, {"21", "22"}};
matrix[0][0] = "00"; // throws exception
```

So if the matrix variable was declared as const, the rows (which are string []) will inherit this property and prevent future changes.

Constant Parameters

The parameters of functions (user-defined routines) can also be declared as constants to prevent changes within the function body.

```
function arrayContains(const string [] array, string element) {
    array[0] = element; // throws exception!
 ...
}
```

New Types

Multi-dimensional arrays

You may have noticed the use of multi-dimensional arrays (arrays of arrays) in one of the examples for constants.

```
string [][] matrix = { {"11", "12"}, {"21", "22"}};
string [] row = matrix[0];
string element1 = row[0];
string element2 = matrix[0][0];
```

Arrays of arrays can have up to 32 dimensions and are pretty similar to simple arrays, except the elements themselves are arrays of dimension n-1. All array-handling routines can also handle multi-dimensional arrays.

Structures

Structures are user-defined blocks of data structured by fields. A structure can have a number of fields, each with its own type, including arrays, structures, or simple types and can also be used to create arrays of structures.

For more information, please see Structures.

Error handling

Due to popular demand, we've also introduced try-catch block to be used for error handling. For more information, please see the dedicated section Error Handling.

Operators

SIL 3.0 can now handle the following operators:

- Pre/Post Increment/Decrement: ++i, i++, --i, i--
- Ternary if condition <condition> ? <ifTrueValue> : <ifFalseValue>

Philosophy behind SIL

[Back in 2010 ...]

It all started when a customer requested us to do the same validator in 3 different ways. Then, the company (a bank) thought that it would be nice to add a very complicated logic behind, logic that should be changed anyway on the real JIRA production system. In the meantime, another customer came and asked us to do the same, but in a slightly different way.

Because those functionalities were needed 'yesterday', we implemented it like it was requested, then we took a deep breath. Here's what I wrote in an 'Business Requirements' document at that time:

> *The problem with current functionalities is that we developed too many post-functions doing basically the same thing but linked to a certain field (for instance assignee, or reporter). The same observation is true for conditions and post-functions.*
>
> *This is wrong, since the function is basically the same: we set a field value to a constant (including here the null value) , to another field's value or to a calculated value. The result is not flexible, certainly not configurable, and minimizes our chances when configuring a JIRA installation.*

The above is always true with any plugin that offers functions related to a certain field. But this is not the only problem that we had.

Of course, one could ask "But why didn't just use the ScriptRunner plugin?" (or any other). To tell you sincerely, we tried. But customers do not like groovy and to deal with all those JIRA internals (yeah, some people really hate the idea that tab defines the block!). Plus, when upgrading JIRA, they needed to upgrade all the scripts and re-test them. We heard our customers moaning.

Actually, we just needed something else: for us it made no sense to write import statements, or take care of the objects when all you needed to do is to set the assignee of an issue on a certain user. It was clear that we needed a new language. And because every language needs to bear a name, we called it **SIL** (Simple Issue Language) and the containing plugin **JJupin** (JIRA Master - *jupin* is an archaic romanian word for 'master'). Lack of inspiration, you may say.

Not quite.

From the very beginning, SIL was developed independently of JIRA, despite of the name (as a side-note, our installers actually run for every plugin that needs to be installed - guess what? - a SIL script). On top of the language - which can be used as a standard language script, like bean shell, groovy, jyton, etc, we have developed variable resolvers that are able to extract JIRA issue fields and deal with them accordingly.

It's true, syntax was very crude, the editors were not very pleasant to use. But it worked so well ...

[... in the meantime ...]

Armed with this tool we had some huge JIRA customization projects. There are JIRA instances out-there on which one would need for sure to take a second look to convince herself / himself that it is a JIRA for REAL !

... and we just got better and better, simplifying the syntax, adding functionality over functionality, routine over routine, support for many more custom fields ...

[... and the present]

We established ourselves as a provider for very complex JIRA solutions. SIL has just made its way in 4 of our public plugins:

- JJupin - http://jira-plugins.kepler-rominfo.com/x/product/id/3
- Blitz Actions - http://jira-plugins.kepler-rominfo.com/x/product/id/9
- Kepler CF - http://jira-plugins.kepler-rominfo.com/x/product/id/8
- Database CF - http://jira-plugins.kepler-rominfo.com/x/product/id/5

... and still we're getting better.

Add-ons using SIL

We are dedicated plugin developers and we are much hooked into the Atlassian community as well.
We received a lot from the JIRA community and we want to give it something back. The is why some very useful plugins are free.

Our plugins are collaborating between them to offer our customers a valuable integrated environment.
The glue of this composition is the Simple Issue Language (SIL).

While **JJupin** and **Blitz Actions** are centered on SIL, the rest of them offer enriched functionality to your JIRA installation.

Here is a complete list of JIRA add-ons (plugins) that are using SIL:

Add-on name	Product page	Documentation	Usage
JJupin (commercial)	JJupin product page	JJupin Documentation	• Write workflow conditions, validators or post-functions • Write (Live Fields) scripts for client browser scripting (avoiding java script!). One can intercept create, edit, view, transition screens. • Write scripts to run regularly to be used in SIL Services • Write event listening SIL scripts • Write scripts to be run on user request with SIL Script Runner
Blitz Actions (commercial)	BA product page	Kepler Blitz Actions Documentation	• Highly customizable dialog-based conversation inside JIRA. • Write custom actions to be called within issue view screen • Contains all required steps: conditions, screen definition, data validation and action itself
Kepler Custom Fields (free)	KCF product page	Kepler Custom Fields Documentation	• SIL Script Custom Field uses the language to define calculated custom fields • Interval CF and Regex CF may be used by other plugins (read/write)
Database Custom Field (free)	DBCF product page	Database Custom Field Documentation	• Data Table Custom Field can be configured to display data from a SIL script data source • Plugin provides routines to manipulate the data from Data Table CF programmatically • Database Custom Fields may be used in SIL scripts (read/write)
Kontinuum (commercial)	Kontinuum product page	Kontinuum Documentation	• Routines are provided to allow users to manipulate the underneath data
User Group Picker PRO (commercial)	User Group Picker PRO page	User Group Picker PRO Documentation	• SIL User Picker CF uses SIL scripting for retrieving restricted users in a Jira style user picker field

The SIL language is one for all add-ons. It defines an unique syntax, the same variables and programming statements and routines.
However there are specific routines (functions) defined to be used in context of unique some add-on.

Special routines are supplementary defined for:

- **JJupin** (live fields routines)
- **Blitz Actions** (screen definition routines)
- **Data Table Custom Field** (data retrieval routines)
- **Kontinuum** (calendar data retrieval)

See the routines section specific to each plugin for further details.

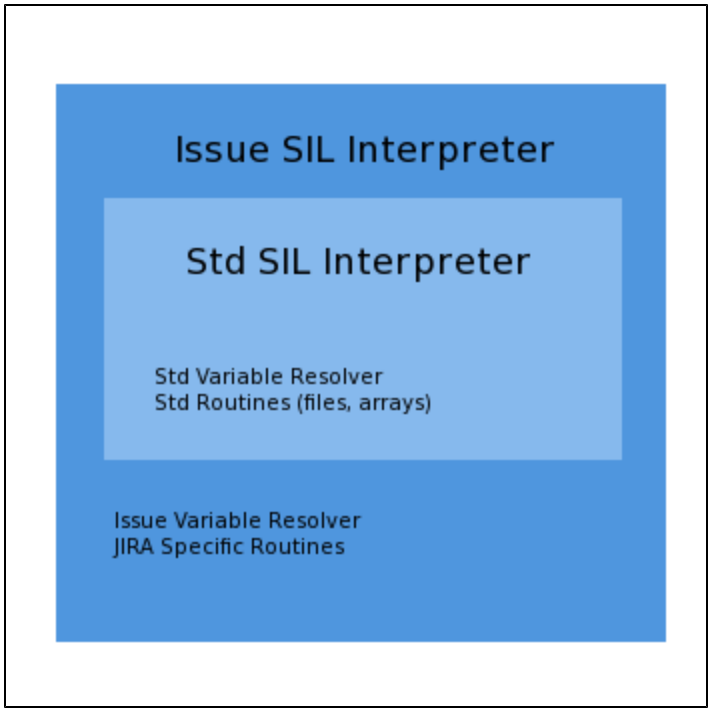
How It Works

On this page:

Even though SIL works just fine as a standalone scripting language outside of JIRA (e.g. our installers use SIL to copy the plugins to your JIRA directory - yes, we use SIL to install SIL), where it really makes a difference is inside JIRA.

Architecture

The SIL language is actually independent of JIRA. It can be used for any purpose, it may, for instance be adapted for the Confluence (it's just an example, there's no plugin planned at this time). It can be adapted for basically anything with ease. On top of the basic interpreter we added a JIRA-specific interpreter which predefines the standard variables (for instance *key* - which represents the key of the issue).



As you can easily see, the JIRA SIL interpreter (or Issue SIL interpreter, how we call it internally) basically extends the capabilities of the standard interpreter. The standard SIL Interpreter provides a registry for all the routines and the Issue SIL Interpreter adds into this registry the additional routines involving JIRA interactions.

Contexts

A standalone SIL program constains, just like any other programming language, variables, functions (we often call them "**routines**") and conditional and repetitive clauses. Putting this inside a JIRA context will allow you to use JIRA-related routines like createIssue or linkIssue (so the script "**has a JIRA context**"). Adding an issue context inside the JIRA context (so the script "**has an issue context**"), will enable you to use field values stored on the issue, be they standard fields like summary, description, assignee or custom fields.

These will be predefined and ready to use without any need for variable definition. When the interpreter (yes, it's interpreted) finds a variable, it looks for it pretty much the same way any other language does. Each block of code is a variable context. Nested blocks will push these contexts into a stack. When looking for a variable, it starts form the current block of code and moves downwards in the stack. It the variable is not found, then it will attempt to match it against fields on the issue (consider this at the bottom of the stack). Let's make it visual using pseudocode.

Outside JIRA	Inside JIRA

```
start block // context_0 -
this is your program
 declare variable a;
 start block // push
context_1
  declare variable b;
  start block // push
context_2
   declare variable c;
   use variable b; // from
context_1
  end block // pop context_2
  // no more variable c here
 end block // pop context_2
 // no more variable b here
end block
```

```
start block // JIRA context
// defines JIRA-related routines
like createIssue(...)
 start block // issue context -
transparent to the user
 // this context contains the
issue field definitions
 // you can imagine this contains
instructions like
 // string summary = "the summary
of the issue";
  start block
   // context_0 - this is your
program
   declare variable a;
   start block // push context_1
    declare variable b;
    start block // push context_2
     declare variable c;
     use variable b; // from
context_1
    end block // pop context_2
    // no more variable c here
   end block // pop context_2
   // no more variable b here
  end block // this is the end of
the program
 // behind-the-scene
post-processing
 end block // pop the issue
context
end block // pop the JIRA context
```

Notice that SIL works pretty much the same way inside and outside of an issue context. The two additional contexts (JIRA and issue) bring more functionality that you can use right away inside your program.

The context rule
All the scripts used by any of our JIRA plugins **have a JIRA context**, but **not necessarily an issue context**. This is important since not having an issue context will not allow you to use variables like summary, assignee or custom fields right away.

Post-processing

Note that when running inside a JIRA context, there is a step called "**behind-the-scene post-processing**". During the interpretation of the script, the SIL Engine (or Interpreter) creates volatile clones (not persistent) of all issues that were modified and the respective changes. This allows us to control two aspects of a program:

- Should the program be read-only ? If a program is read-only (like a workflow condition or validator), then the changes recorded by the Interpreter will NOT be persisted to the database/index. Otherwise, the "**behind-the-scene post-processing**" will ensure that all the recorded changes are persisted to the database/index.
- What happens in case of an error ? If a program throws an exception, we do not want to corrupt the issue data by storing only the changes that were made before the error occurred, so we just throw it all out and report the error, leaving the issue as it was.

Routines exception
At the time of this writing, some routines (like createIssue) cannot be undone automatically and will persist their changes regardless of whether the script was read-only or if there was an error.

What can you do with SIL

There are many things that SIL language can used for within our plugins.

Here is a high level feature list across all plugins using SIL.

Feature	Using Plugin
Workflow transition conditions, validators and postfunctions	JJUPIN
Change issue field values during workflow transitions	JJUPIN
Change issue field values outside workflow (operations/actions)	Blitz Actions
Event listeners	JJUPIN
JIRA Services	JJUPIN
On-screen (view screen) manipulation of fields	Blitz Actions
Run SIL scripts	JJUPIN
Calculated fields	SIL Script Custom Field
Fetch database data based on JIRA data and SIL expressions	Database Custom Field JJUPIN Blitz Actions KCF Pro SIL Excel Reporting
Configurable user picker	User Group Picker User Group Picker Pro KCF Pro
Programatically manage JIRA Agile sprints, epics and other features	JJUPIN Agile (requires JJUPIN)
Configure dynamic select list	KCF Pro
Execute an action when an option is selected	KCF Pro
Create Reports (or just extract data) from JIRA or external databases to be mastered by Excel's analytical and graphing features	SIL Excel Reporting

Simple Issue Language Usage

Why SIL ?

If you need to customize JIRA really deep, choose SIL through our one-of-the-kind plugins JJUPIN and / or Blitz Actions. The top reasons to use it are:

1. really delivers fast.
2. shields you form the JIRA internals,
3. consistent from version to version
4. extensible (routines and custom fields)
5. covers many other commercial plugins functionalities.

Choosing SIL is a smart thing to do:

- If you really care about standard programming practices, you can move workflows from test to production environments very quick.
- Scripts can be easily adapted and modified on the fly
- You pay only once, then you can cover countless customizations, which otherwise would require a dozen of plugins
- It is supported and YOU have the power.

In our opinion (biased, of course) it is the best option you could have for JIRA customization.

An Example

First of all, here is a short example so you know what to expect:

```
string k;

assignee = "admin";
reporter = assignee;
created = currentDate();
description = "some description";
dueDate = currentDate() + "1d";
env = "environment";
estimate = "2d" + "3h";
originalEstimate = "1d 4h" + "21h";
priority = "Critical";

if(not contains(summary, "test")){
    summary = "test " + summary;
} else {
    summary = "random summary assigned";
}

spent = "2d";
updated = currentDate() - "1d";
votes = votes + 1;
workflow = "TWFLScheme";

if(issueType == "Bug"){
    issueType = "Task";
} else {
    issueType = "Bug";
}
project = "TSTP";

//Custom fields here
UPPG = "admin";

//time interval custom field
if(isNull(tt1)) {
    tt1 = estimate + "1h";
} else {
    tt1 = tt1 + "1h";
}

//number custom field
if(isNull(cfnumber)){
```

```
    cfnumber = 1;
} else {
    cfnumber = cfnumber + 1;
}

//create routine
k = createIssue("TSTP", "", issueType, "auto-created issue");
%k%.votes = %k%.votes + 1;

//autotransition
autotransition(721,key);
```

```
//or:
//autotransition("Send report","PRJ-123");
```

Do not worry if all is not clear from the start, it will all make sense after you finish reading this guide.

Structure Of A SIL Program

Structure Of A SIL Program

The structure of a SIL program should look like:

```
[inclusions]
[variable declarations]
[user-defined routines declarations]
[actual code]
```

Inclusions

Include statements must be the first statements in your program. These allow you to import libraries of user-defined routines or execute certain fragments of code.

> **Learn more**
> For more information see Inclusions.

User-Defined Routines declarations

Here you can define any functions you want to use in the code. These can considerably improve the readability and maintainability of the code.

> **Learn more**
> For more information see User-Defined Routines (UDR).

Actual code

This is the body of the program. Here you will do any necessary modifications to the issue. The body can also contain definition of local variables and calls to the imported or defined routines in the steps above.

> **Learn more**
> For more information see Syntax.

Syntax

On this page:

- Routine Declaration
- Type Casts

Just like with any scripting language, there are a few simple syntax rules you must follow when using SIL.

General rules

- All instructions end with a semicolon **;**
- Comments are introduced using //comment_until_end_of_line or /* comment */

Example

```
// single line comment
/*
this comment
spans over
multiple lines
*/
```

Types

Basic Types

SIL defines the following base types:

- **string** - defines a string, example literal is "abc"
- **boolean** - defines boolean values true and false
- **number** - any numeric (integer or real) value
- **date** - a date, like "2010-03-30"
- **interval** - a time interval, like "2d 3h"

Important notes:

- **interval** data type does not represent the **working interval** (i.e. 8h), but the full interval. When we created SIL we thought about if we should introduce a working interval type (i.e. 8h / day) but this would have some undesired implications (how long is your workday? in France, for instance, Friday is only 6h). With our custom fields, it's a matter of interpretation of the interval, and we have routines to help you transform a 24h interval to a working interval, or you can simply write your own.
- **date** data type can be used to represent dates and date-times.
- **number** data type internal storage is represented only by real numbers (double). However, the language accepts for historical reasons type declarations as "integer", "int", "real", "double", "float" (treating them as "number", of course). Until now we have found no limitations regarding the uniform treatment of numbers.

Arrays

Multi-value (arrays) types are also supported and composed of the base type followed by the array symbols **[ ]** for each dimension of the array type. This translates to the following being valid array types:

- **string[]**
- **boolean[]**
- **number[][]**
- **date[][][]**
- **interval[]**
- **Person[][] -** where Person is a user-defined structure

Key-Value Arrays (Maps)

Arrays are actually maps where values are keyed by their position in the list, and so key-value arrays are declared the same as regular arrays.

Learn more about maps and the indexing operator.

Structures

The general syntax for defining a structure is:

```
struct <name> {
 <type> <fieldName>;
}
```

Structures are defined by their **name** and pairs of **field type** and **field name.** A structure can have any number of fields. The type of each field can be a simple type, a user-defined structure (including the structure being defined) or arrays of these types.

Example:

```
struct Person {
 string id;
 string name;
 date birthDate;
 Person manager;
 Person [] subordinates;
}
```

To access the value of a field from a variable, the syntax is:

```
<varName>.<fieldName>
```

Example

```
// considering structure Person defined above
Person p;
p.name = "John Doe";
p.id = "1234567";
string hisManager = p.manager.name;
string employee = p.subordinates[0].name;
```

Constants

> This topic is about representing constant values, NOT about the read-only attribute of variables.

Single Value Constants

For **numbers** and **boolean** values, constants are represented without any additional alterations.

Example

```
number n = 1;
number pi = 3.14;
boolean iLikeSIL = true;
```

Constants for **string**, **date** and **interval** are provided between double quotes (").

- The full format for **date** is **"yyyy-MM-ddTHH:mmZ"**, and may include the time-zone information at the end. The short format **"yyyy-MM-dd"** is accepted as well.
- The **intervals** should be provided in the JIRA standard format: **"1w 2d 3h 4m"**.

Example

```
// valid dates
"2010-12-31"
"2010-12-31T24:59:59"
"2010-12-31T24:59:59+0200"

// interval
"1w 2d 3h 4m"
```

Array Constants

Arrays are created using the following construct:

```
{<value1>, <value2>, ..., <value3>}
```

Example

```
{"this", "is", "a", "string", "array"} // string array
{{0,1}, {2,3}, {4,5}} // number matrix
{true, false, false} // boolean array
{"1d", "2d", "3d"} // interval array
{ variable1, variable2, variable3 } // array of the type of variable#,
where variable# can also be an array
```

Note
We currently do not support constant representation of key-value arrays. You will need to build one using the indexing operator.

Variable Declaration

The general syntax for declaring a variable is:

```
<type> varname; // declaration without initialization
// or
<type> varname = <expression>; // declaration with initialization
```

Example

```
string name = "John Doe";
number random = 2;
number pi = 3.14;
boolean valid;
date today = currentDate();
interval spent = "1h 30m";
interval estimate = "2d" - spent;
number [][] matrix = {{0,1}, {2,3}, {4,5}};
```

Constant variables

Variables can be made read-only by adding the keyword "**const**" before the type when the variable is first defined.

```
const string s;
```

If set to an array or structure, the read-only attribute will be propagated to all the elements of the array or fields of the structure.

Routine Declaration

In addition to the routines provided out-of-the-box by SIL, you can define your local routines (user-defined routines or UDRs for short) which use the following syntax.

```
function <name>(<type> param1, <type> param2, ...) {
   Instruction1;
   ...
   InstructionN;
   return <value>;
}
```

Learn more about UDRs.

Type Casts

Type casts allow you explicitly convert a variable to a specific type. The general syntax for casting is:

```
(<target_type>)varname
```

Example

```
number n = 1;
string s = "2";
return n + s; // what will this return? 3 or "12"?
return (string)n + s; // now we know that adding 2 strings will result in
"12"
return n + (number)s; // adding 2 numbers will definitely return a number
(3)
```

Note
Not all type casts are valid. For example it makes no sense to convert an interval to a date. See Type Conversion for more information on valid conversions.

Operators

Arithmetic, Logical and Comparison Operators

Here is a list of the available operators in a SIL program and detailed usage information.

Arithmetic operators

In version 2.5 (and 3.0 yet again) we have reworked the way operators handle different types and conversions by putting an emphasis on the left-hand side (LHS) value of the operation. Consequently, only the right-hand side (RHS) operand will be converted (if necessary) to a type that is valid for the current operation with the LHS type.

For each operator and LHS type, we define two lists: valid types and convertible types. The list of convertible types is based on Type Conversion. This creates three cases:

1. If the RHS type is one of the valid types, then the operation will proceed as it is and the result calculated according to the table below.
2. If the RHS type is one of the convertible types, then it will attempt to convert it to one of the valid types (**in the order the valid types are presented in the table below**) until the conversion succeeds and then proceeds according to case 1.
3. If the RHS type is neither valid nor convertible, the program will end in error.

Operator	LHS type	Valid Types	Result	Explanations and Usage
+	number	number	number	standard addition of numbers
	string	string	string	adds the string representation of the RHS operand at the end of the LHS string
	boolean	**Unsupported**		
	date	interval	date	adds the interval to the date and returns the result
	interval	interval	interval	adds the two intervals and returns the result
		date	date	adds the interval to the date and returns the result
	<type> []	<type>	<type> []	adds the RHS value to the LHS array, where <type> can be any type.
-	number	number	number	standard subtraction of numbers
	string	string	string	removes all occurrences of the RHS string in the LHS value
	boolean	**Unsupported**		
	date	date	interval	returns the interval difference between the two dates
		interval	date	subtracts the RHS interval from the date and returns the value
	interval	interval	interval	standard interval subtraction
	<type>[]	<type>	<type>[]	removes the first occurrence of the RHS value from the array; <type> can be any type.
*	number	number	number	standard multiplication of numbers

		interval	interval	multiplies the interval by the number of times specified by the LHS operand and returns the result
	number []	number	number []	multiplies each value in the LHS array with the number in the RHS operand and returns the result
	string	**Unsupported**		
	boolean	**Unsupported**		
	date	**Unsupported**		
	interval	number	interval	multiplies the interval by the number of times specified by the RHS operand and returns the result
	<type>[] (except number)	**Unsupported**		
/	number	number	number	standard division of numbers
	number []	number	number []	divides each value in the LHS array by the number in the RHS operand and returns the result
	string	**Unsupported**		
	boolean	**Unsupported**		
	date	**Unsupported**		
	interval	number	interval	divides the interval by the number specified in the RHS operand
	<type>[] (except number)	**Unsupported**		
%	number	number	number	standard modulo division of numbers
	number []	number	number []	applies the modulo operator on each number in the LHS array with the number in the RHS operand and returns the results
	string	**Unsupported**		
	boolean	**Unsupported**		
	date	**Unsupported**		
	interval	**Unsupported**		
	<type>[] (except number)	**Unsupported**		

You can also use the combined forms of the arithmetic operators with the attrib operator: **+=**, **-=**, ***=** and **/=**.

```
string s = "a";
s = s + "b";
// is equivalent with
s += "b";
```

Numbers also support the pre/post increment/decrement operators.

```
--i;
i--;
++i;
i++;
```

Comparison operators

Operator	Operands	Return	Explanations and Usage
== (alias 'eq')	Right operand will be casted to the left operand's type if their types are different	true/false	Returns true if the values are equal and false otherwise.
!= (alias 'neq')	Right operand will be casted to the left operand's type if their types are different	true/false	Returns false if the values are equal and true otherwise.
< (alias 'lt')	Right operand will be casted to the left operand's type if their types are different	true/false	Returns true if the first value is lower than the second one and false otherwise.
> (alias 'gt')	Right operand will be casted to the left operand's type if their types are different	true/false	Returns true if the first value is greater than the second one and false otherwise.
<= (alias 'le')	Right operand will be casted to the left operand's type if their types are different	true/false	Returns true if the first value is lower then or equal to the second one and false otherwise.
>= (alias 'gt')	Right operand will be casted to the left operand's type if their types are different	true/false	Returns true if the first value is greater than or equal to the second one and false otherwise.

Logical operators

Operator	Operands	Return	Explanations and Usage
&& (alias 'and')	boolean && boolean	true/false	Logical "and"
|| (alias 'or')	boolean || boolean	true/false	Logical "or"
! (alias 'not')	boolean	true/false	Logically complements of the specified boolean.
! (alias 'not')	number	number	Complements the number in relation to 0. (returns number * -1)

Ternary operators

Operator	Operands	Return	Explanations and Usage
?:	boolean, <type>, <type>	<type>	`<condition> ? <ifTrueValue> : <ifFalseValue>` If <condition> is true, returns <ifTrueValue>, otherwise returns <ifFalseValue>. Note that <ifTrueValue> and <ifFalseValue> must have same type.

The Indexing Operator

The basic use of the indexing operator (introduced in version 2.5) is to allow you to easily access the contents of a list.

General syntax

```
string [] arr = {"a", "b", "c"};
string contents = arr[0] + arr[1] + arr[2] + arr[3]; // as user friendly as
arrayGetElement, if index is out of bounds, will return an empty value.
arr[3] = "d";
```

Moreover, it allows you to create key-value lists (maps) and retrieve values from them by using a string inside the operator (instead of a number).

```
number [] map;
map["one"] = 1;
map["two"] = 2;
print(map["one"]);
// note that when used in a for-loop, map only has 2 elements: 1 and 2. The
keys will be hidden.
print(map); // will print 1|2
```

When accessing a map as a list, the contents are retrieved in the order they were added. You can still use the operator with an integer value **even on maps.**

```
date [] days;
days["yesterday"] = currentDate() - "1d";  // added as first element
days["today"] = currentDate();     // added as second element
days["tomorrow"] = currentDate() + "1d";   // added as third element
if(days["today"] == days[1]){
 print("Today is the second element in the array"); // yes, this will be
printed
}
```

The indexing operator can also be used with certain simple types (not arrays) and a specific value to provide additional functionality.

SIL type	Index Value	Get	Get Return Type	Set
string	a number	Gets the character at the specified index in the string	string	**Inserts** a character at the specified index. If given a longer string, it will insert the first character of the string.
date	"DAY"	Gets the day of month from the date	number	**Unsupported**
	"MONTH"	Gets the month from the date	number	**Unsupported**
	"YEAR"	Gets the year from the date	number	**Unsupported**
	"HOUR"	Gets the hour (0-23) from the date	number	**Unsupported**
	"MINUTE"	Gets the minute (0-59) from the date	number	**Unsupported**
	"SECOND"	Gets the seconds (0-59) from the date	number	**Unsupported**
	"MILLISECOND"	Gets the milliseconds (0-999) from the date	number	**Unsupported**
	"WEEK"	Gets the week number within the current year	number	**Unsupported**
	"WEEKINMONTH"	Gets the week number within the current month	number	**Unsupported**
	"TOMILLIS"	Gets the current time as UTC milliseconds from the epoch.	number	**Unsupported**
	"DAYOFWEEK"	Gets the three letter format for the day of the week (e.g. "Mon", "Tue", "Wed" etc)	string	**Unsupported**
	"MONTHNAME"	Gets the three letter format for the name of the month (e.g. "Jan", "Feb", "Mar", etc.)	string	**Unsupported**
interval	"WEEK"	Gets the number of whole weeks inside the interval (e.g. interval i = "1w 14d"; i["WEEK"] will return 3)	number	**Unsupported**
	"DAY"	Gets the number of whole days inside the interval (e.g. interval i = "1d 48h"; i["DAY"] will return 3)	number	**Unsupported**
	"HOUR"	Gets the number of whole hours inside the interval (e.g. interval i = "1h 120m"; i["HOUR"] will return 3)	number	**Unsupported**
	"MINUTE"	Gets the number of whole miuntes inside the interval (e.g. interval i = "1m 120s"; i["MINUTE"] will return 3)	number	**Unsupported**

	"SECOND"	Gets the number of seconds inside the interval (e.g. interval i = "1h 1m 3s"; i["SECOND"] will return 3)	number	**Unsupported**
	"TOMILLIS"	Gets the number of milliseconds inside the interval	number	**Unsupported**

See also:

- Syntax
- Variable Resolution
- Statements

Predefined Structure Types

Here is a list of predefined structures that are used throughout SIL (usually with specific routines). Not enough for you? You can also define custom structures! Check our example in the Syntax section and roll your own.

Note that the structure fields can be modified, but the changes are not saved in the database. If you change a field for a project version variable, the project version will not be affected. Simply put, the structures are 'detached'.

Name	Field	Type
JVersion	id	number
	name	string
	description	string
	projectKey	string
	startDate	date
	releaseDate	date
	archived	boolean
	released	boolean
JComment	id	string
	text	string
	author	string
	created	date
	updatedBy	string
	updated	date
	securityLevel	string
JProject	id	number
	key	string
	name	string
	description	string
	lead	string
	url	string
	unassignedByDefault	boolean
	avatarId	number
	category	string

	projecttype (since version 3.1.0)	string
JLdapUserStruct	DN	string
	attributes	JLdapUserAttribute []
JLdapUserAttribute	name	string
	value	string []
JUser **(since 3.0.1)**	key	string
	username	string
	displayname	string
	email	string
JWorklog **(since 3.0.2)**	id	number
	author	string
	startDate	date
	timeSpent	interval
	comment	string
	issue	string
JComponent **(since 3.0.2)**	id	number
	name	string
	description	string
	lead	string
JIssueLink **(since 3.0.13 for JIRA 6.x use, since 3.1.0 for JIRA 7.x use)**	id	number
	name	string
	direction	number
	description (since 3.1.1 for JIRA 7.x use)	string
	issue	string

Variable Resolution

Referencing an Issue

By default, the issue standard variables are pre-declared along with their synonyms. For instance,the **key** is an issue standard variable. You should avoid to re-declare it, since it will lead to subtle errors. We recommend you to **prefix** your variables with some string, e.g. my_.

There are some constructs that reflect the issue structure, e.g.:

- **parent.id** - refers to the parent id, and it is valid only if this issue is a subtask
- **TSTP-123.id** - refers to the issue TSTP-123 id (on the project TSTP)
- **%k%.id** - (Substitution, see below) refers to the issue designed by *k*, where *k* is a string variable already defined, containing a valid issue key, like in this example:

```
string k = "TSTP-123";
%k%.reporter = currentUser();
```

Tip
You will see the last construct comes in hand when used in cycles (for, while, do-while) or when creating an issue from SIL.

Substitution

It often happens that you need the value of a variable designated by another value. For example, we might have a custom field that specifies which other field contains some data. The general syntax for substituting variables is:

```
%varname%
```

The above expression will evaluate to the value *designated* by the variable name **contained inside the value** of varname.

Example

```
string myCustomField = "customfield_10000"; // myCustomField is a local
variable
%myCustomField% = "a value";
```

Standard Variables

Here is a list of the variables that are already defined in a SIL environment and which you can use right away. Note that these are all standard issue fields.

Note
Standard variables can be used without explicitly referencing the issue only when the script has an issue context.

Warning
Please do not attempt to redefine them. Redefinition is possible with the loss of the original meaning, so you will have subtle errors in your SIL code.

Variable	Aliases	Type	Readonly	Explanations and Usage
affectedVersions	affectedVersion affectedVers affectedVer affVers affVer	string []	no	The affected versions field of the issue. If attempting to add an invalid value, it will be ignored.
assignee	-	string	no	The assignee of the issue. Represents the username, not the real name.
attachments	attach	string []	yes	The filenames of the attachments. There are routines to modify those
components	component comps comp	string []	no	The components of the issue. If attempting to add an invalid value, it will be ignored.
created	-	date	no	The date when the issue was created.

description	desc	string	no	The description of the issue.
dueDate	due	date	no	The due date of the issue.
estimate	est remaining	interval	no	This is displayed as "Remaining" in the JIRA interface. Requires TimeTracking.
environment	env	string	no	The environment of the issue.
fixVersions	fixVersion fixVers fixVer	string []	no	The fix versions field of the issue. If attempting to add an invalid value, it will be ignored.
id	-	number	yes	The id of the issue.
issueType	type	string	no	The name of the issue type.
issueTypeId	-	string	no	The id of the issue type.
key	-	string	yes	The key of the issue.
labels	-	string []	no	The labels of the issue.
originalEstimate	origEstimate origEst	interval	no	The original estimate of the issue. Requires TimeTracking.
parent	-	string	no (yes, since 1.4.3)	The key of the parent issue.
parentId	-	number	no (yes, since 1.4.3)	The id of the parent issue.
priority	prio	string	no	The name of the priority.
priorityId	-	string	no	The id of the priority.
project	prj	string	no	The key of the project.
projectId	-	number	no	The id of the project.
reporter	-	string	no	Represents the username, not the real name of the reporter.
resolution	res resol	string	no	The name of the resolution. When set will also modify the resolution date.
resolutionDate	-	date	no	The current resolution date. If the resolution is modified, the date will also be updated.
resolutionId	resId resolId	string	no	The id of the resolution. When set will also modify the resolution date.
securityLevel	security	string	no	The name of the security level.
securityLevelId	securityId	number	no	The id of the security level.
status	-	string	yes	The name of the status.
statusId	-	string	yes	The id of the status.
summary	-	string	no	The summary of the issue.
timeSpent	spent	interval	no	The time spent (logged work) on the issue. Requires TimeTracking.
updated	-	date	yes	The date when this issue was last updated. It will update automatically after the current transition.
votes	-	number	no	The number of votes this issue has.
watchers	-	string []	no	The watchers of the issue. The elements in the array are usernames.
workflow	wrkflw wf	string	no (yes, since 1.4.3)	The workflow name of the issue.
workflowId	-	number	no (yes, since 1.4.3)	The workflow id of the issue.

Custom Fields

Aside from the standard issue fields, you can also access custom fields from SIL in one of three ways:

- by id, using the construct **customfield_xxxxx** - where xxxxx is the ID of the custom field.
- by name - don't forget to use **#{** and **}** if the name contains spaces.
- by its alias

Examples

```
customfield_10000 = "http://bugs.kepler-rominfo.com/browse/TST-1"; //
referencing by id
External = "http://bugs.kepler-rominfo.com/browse/TST-1"; // referencing by
name
#{External URL} = "http://bugs.kepler-rominfo.com/browse/TST-1"; //
referencing by name with spaces
```

Note
If you have multiple custom fields with the same name, the JIRA API returns the first one it finds.

Learn more
See Custom Field Types for a list of SIL data types used by different custom fields.

Statements

Introduction:

Statements provide a convenient method to execute conditional or repetitive operations. This section contains these statements

Summary:

If-else statement	The **if-else** statement offers support for conditionally executing operations.
While statement	The **while** statement offers support for repeated executions. This form evaluates the condition first and then executes the instructions in the body.
Do-while statement	The **do-while** statement is similar to the while statement, except that the condition is evaluated after the execution of the encapsulated block. So,even if the condition is false, the instructions will still be evaluated once.
For statement	The for statement provides a compact way to iterate over a range of values. Programmers often refer to it as the *for loop* because of the way in which it repeatedly loops until a particular condition is satisfied.

If-else statement

Info:

The **if-else** statement offers support for conditionally executing operations.

Syntax

The general syntax for this is:

```
if(condition) {
    Instruction1;
    ...
    InstructionN;
} else {
    Instruction1:
    ...
    InstructionN;
}
```

Note: the **else** branch can be omitted

Example:

```
if (isNotNull(fixVersions) and affectedVersions == {"1.1"})
{
    affectedVersions = {"1.1", "1.0" , "1.2"};
    fixVersions = {"1.2" , "1.2" , "1.3"} ;
}
else
{
    affectedVersions = {"1.1"};
    fixVersions = {"1.0"};
}
```

While statement

Info:

The **while** statement offers support for repeated executions. This form evaluates the condition first and then executes the instructions in the body.

Syntax:

```
while(condition) {
    Instruction1;
    ...
    InstructionN;
}
```

Example:

```
number i = 1;
while(i <= 3) {
    multisel = arrayAddElement(multisel,"value" + i);
    i = i + 1;
}
```

See Also:

Do-While statement

Do-While statement

Info:

The **do-while** statement is similar to the while statement, except that the condition is evaluated after the execution of the encapsulated block. So,even if the condition is false, the instructions will still be evaluated once.

Syntax:

The general syntax is:

```
do {
    Instruction1;
    ...
    InstructionN;
} while(condition);
```

Example:

```
number i = 1;
string [] people;
do {
    people = arrayAddElement(people,
                             arrayGetElement(watchers, i));
    i = i + 1;
} while(i < 5);
```

See Also:

While statement

For statement

Info:

The for statement provides a compact way to iterate over a range of values. Programmers often refer to it as the *for loop* because of the way in which it repeatedly loops until a particular condition is satisfied.

In SIL the **for** statement has two forms:

- the **standard for** or simply **for** form
- the **for-each** form

The standard form:

Similar to C, C++ or Java of the for statement can be expressed as follows:

```
for(<init>; <condition>; <increment>){
   Instruction1;
   ...
   InstructionN;
}
```

For this first form, *<init>* can be an attribution of an already defined variable or a definition for a new variable.

The —for-each form

The **for-each** form can be expressed as follows:

```
for(<variable_definition> in #{array}){
   Instruction1;
   ...
   InstructionN;
}
```

Example 1 (standard form):

Prints the numbers from 0 to 9*

```
for(number i = 0; i < 10; i = i + 1){
   print(i);
}
```

Example 2 (for-each form):

Prints the list of the watchers of a specific issue*

```
for(string user in watchers){
   print(user);
}
```

See also:

While statement

Do-While statement

Type Conversion

General Rules:

Type conversion is usually done when using the attrib (=) operator or type casting. The types convertible to a target type are listed in the table below:

Target Type	Convertible types	Additional Information / Examples
number	string	"1" to 1
string	ALL	
boolean	string, number	"true" to true value of number != 0
date	string, number	Contructing a date from number assumes the number is a "to millis" representation of a date
interval	string, number	Constructing a date from number assumes the number is a "to millis" representation of an interval 60000 to "1m" "120m" (string) to "2h" (interval)

When converting a string to an array type, the string is first transformed to a string[] by splitting it at every "|" character, and then the conversion is done from string [] to the target type array.

The conversion from string to a target type is done by parsing the string as a text representation of the target type.

Converting arrays

Converting arrays of different types is possible if the element types (inner type of the array) are convertible according to the the rules explained on this page. For example you can cast a number[] to a string[] because number is convertible to string, however you cannot convert a date[] to an interval[] because date cannot be converted to interval.

Converting structures

Structures can only be converted directly to **string** or an **array type**. Converting a structure to an array requires that each field of the structure is convertible to the element type (inner type) of the array.

It may be possible for some structures to be convertible to other types by using one or more intermediate types, however this is nod done implicitly and users are required to use explicit casting in such cases. For example, a structure containing a single number field may be converted to a number by first casting it to a string then the resulting string to a number.

Casting a structure to a **string** is equivalent to casting to a string [] and then the resulting array to string.

Calling routines
When calling a routine or UDR, if the given parameter's type is not the same as what the routine is expecting, the conversion will be done automatically according to the rules above.

```
function f(string s){
 print(s);
}
f(2); //number automatically converted to string
f(true); // boolean automatically converted to string
```

Compatibility
These conversions cover the functionality implemented before v2.5, so they are backwards compatible.

Error Handling

Even though we've tried to keep the language as user-friendly as possible by returning empty values rather than throwing exceptions where possible, there are still cases when exceptions need to be thrown.

Additionally, users may want to throw the errors themselves (e.g. from common user-defined functions) and handle error results differently depending on context.

There are two types of exceptions:

1. Java Exceptions - thrown from the java code running behind SIL
2. SIL objects thrown using **throw** (explained below)

Throwing SIL Objects

The general syntax for throwing an exception from SIL is:

```
throw <value>;
```

Any SIL value can be thrown as exception. Values can either be constants, variables or expressions.

```
throw errorCode;
throw 2;
throw array[0].fieldName;
throw ("Fiend value invalid: " + structure.field);
throw "Generic error!";
```

Try-catch block

Syntax

```
try {
 <instructions>
} catch <type> <varName> {
 // handle <varName>
} catch {
 // handle other error
}
```

The try-catch block contains the following components:

- one **try block** containing instructions to execute.
- zero, one or more **typed catch clauses.**
- zero or one **catch all clause**

If any instruction in the try block throws an exception, execution of instructions within the try block is stopped. That is, any instructions within the try block after the one that throws the exception will not be executed.

Next, SIL will iterate over the catch clauses **in the order they were declared** looking for a catch clause that matches the **type** of the error that was thrown:

- Typed catch clauses will match if the <type> declared in the catch clause matches the type of the exception. That means that **typed catch clauses can only catch SIL objects thrown using throw**. The caught value is available inside the catch block using the defined <varname>
- The catch all clause will catch any exception (SIL Objects or Java exceptions) regardless of type.

If a matching catch clause was found, the instructions within the catch block are executed and the program will continue. If no matching catch clause is found, the exception is thrown inside the outer code block. This makes it possible to nest try-catch blocks within one another and throw exceptions from the inner block to be caught by the topmost one.

Examples

Type matching

```
try {
 number errorCode = 2;
 throw errorCode;
} catch string s {
 // will not match
} catch number err {
 // will match because a number was thrown
    //err = 2
}
```

Catch all

```
try {
 number errorCode = 2;
 throw errorCode;
} catch string s {
 // will not match
} catch {
 // will match, but the error code is not available in this context
}
```

Nesting

```
try {
 try {
  throw "Error!";
 } catch number errCode {
  // will not match because "Error!" is a string
 }
} catch string err {
 // matches
 // err = "Error!"
}
```

Java Exceptions

```
try {
 runAs("user_that_does_not_exist");
} catch string err {
 // will not match
} catch {
 // will match
}
```

Custom Field Types

SIL automatically converts custom field values to SIL values. It does so following the guidelines in the table below. If you wish to write your own custom field compatibility plugin, see the Extension manual.

Custom Field Type	SIL Type	Explanations and Usage
Date Time	date	

Date Picker	date	
Number Field	number	
URL Field	string	
User Picker	string	String representing the username, not the full name of the user.
Group Picker	string	
Multi Checkboxes	string []	String array containing the selected checkboxes
Multi Select	string []	String array containing the selected values
Radio Buttons	string	String containing the selected value
Select List	string	String containing the selected value
Text Field	string	String containing the text
Text Area	string	String containing the text
Multi Group Picker	string []	String array containing the selected values
Multi User Picker	string []	String array containing the selected values (usernames)
Cascading Select	string []	String array with 2 elements the root and the child. If given invalid values, it will ignore them. If given only one value, that will be the root.
Project Picker	string	String containing the selected project key (available since katl-commons 2.5.15 and 2.6.7)
Version Field	string	String representing the version name
Multi Version Field	string []	String array containing the version names
Labels Field	string []	String array containing the labels

See Also
SIL Custom Field Descriptors

Routines

Introduction

The language defines a library of standard routines. These routines are listed below. If you would like to write routines of your own, see the Extension manual.

Summary

Category	Description
Array Routines	contains a collection of routines for handling arrays
JIRA Integration	contains routines that interact with the JIRA environment
System Integration	includes routines for System integration
File Manipulation Routines	contains a collection of routines for handling directories and files

String Routines	contains a collection of routines for handling strings
Date routines	contains a collection of routines for handling date
Basic Routines	includes routines that provide basic language behavior
Parameter Routines	This section includes the routines meant to help the user define and configure the starting parameters for the SIL scripts used in this plugin and to retrieve values from the parameters defined by the input routines. A similar set of routines may be found in another one of our plugins, namely the BA plugin. Aliases for these routines can be found in Sil Excel Reporting plugin.
User-Defined Routines (UDR)	

See Also

- Custom Field Types
- Operators
- Statements
- Syntax
- Variable Resolution

Basic Routines

Introduction

This section includes routines that provide basic language behavior.

Routines summary

Routine	Description
isNull	Checks if the provided variable is null or has no value associated and returns true. Otherwise returns false.
isNotNull	Checks if the provided variable is not null or has a value associated and returns true. Otherwise returns false.
print	Returns the printable string in the application log.
logPrint	This routine is used to print a message in the configured JIRA logs on the specified level: **TRACE, DEBUG, INFO, WARN, ERROR** , **FATAL**.

See Also:

- Routines
- Syntax
- Variable Resolution

isNull

Availability

This routine is available since **katl-commons 1.0 .**

Syntax:

isNull(variable)

Description:

Checks if the provided variable is null or has no value associated and returns true. Otherwise returns false.

Parameters:

Parameter name	Required	Description
variable	Yes	The value that you want tested. The value argument can be a string or a variable of any type.

Return type:

boolean (true/false)

Example:

```
if(isNull(assignee)) {
  assignee = reporter;
}
```

Notes:

If **isNull** returns true the variable has no value attached. **isNull** will return false for **zero** for numeric variables or **blank** for character/string

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

isNotNull

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

isNotNull(variable)

Description:

Checks if the provided variable is not null or has a value associated and returns true. Otherwise returns false.

Parameters:

Parameter name	Required	Description
variable	Yes	The value that you want tested. The value argument can be a string or a variable of any type.

Return type:

boolean (true/false)

Example:

```
if(isNotNull(reporter))
{
  assignee = reporter;
}
```

Notes:

isNotNull returns true if the variable has a value attached (including **zero** for numeric variables or **blank** for character/string)

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

print

Availability
This routine is available since **katl-commons 1.0**, changed in **2.5 .**

Syntax:

print(variable)

Description:

Since **jjupin 2.5** and **katl-commons 2.5** this routine has been changed. Before this change was done the print routine was printing the message in the application log, on **INFO** level.

The arguments are converted to string and then printed. Since the new version no configuration for logging is necessary because the arguments (converted to string) are always shown in the application log.

This routine is similar to **printing a string on the console** in any programming language.
Returns the printable string in the application log.

Parameters:

Parameter name	Required	Description
variable	Yes	The value that you want printed. The value argument can be any printable characters: strings, chars, numbers, dates, intervals, etc.

Return type:

None

Example:

```
if(isNotNull(dueDate))
{
  print("You should complete this task before " + dueDate);
}
else
  print("This task has no deadline!");
```

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

logPrint

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

logPrint(logLevel, #{message})

Description:

This routine is used to print a message in the configured JIRA logs on the specified level: **TRACE, DEBUG, INFO, WARN, ERROR**, **FATAL**.

Logging the above messages is subject to the same configurations of the plugin as described here.

Parameters:

Parameter name	Required	Description
logLevel	Yes	The minimum level for logging for which the message is written.
message	Yes	The argument converted to a string that will be printed.

Return type:

None

Configuration

To configure the **JIRA Logging level** you can do this in two ways:

1) **Temporarily** - (the logging level will be reset after a reboot of Jira):

a) Log in in Jira using an administrator account.

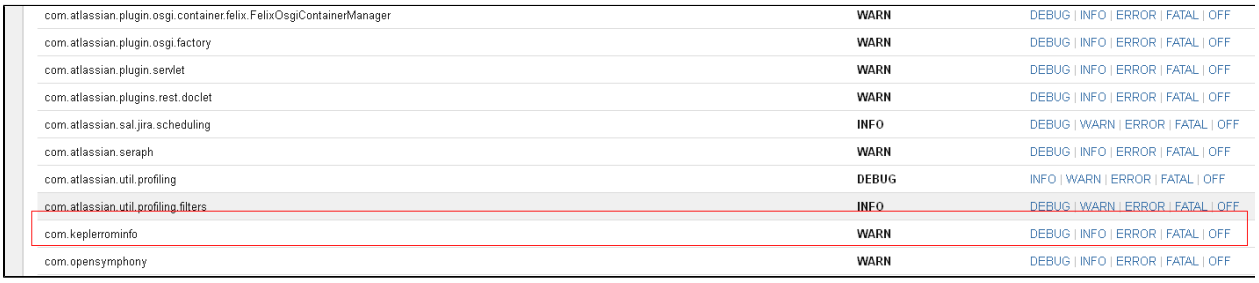
b) Go to "**Administration**".

c) Go to "**System**" tab and select the "**Troubleshooting and Support**" option.

d) Select the "**Logging & Profiling**" tab.

e) Here you'll find a paragraph named "**Default Loggers**" with a lot of package names and their **Logging level**.

f) Search in this list for the line "**com.keplerrominfo**". And select from the "**Set Logging Level"** your desired level.

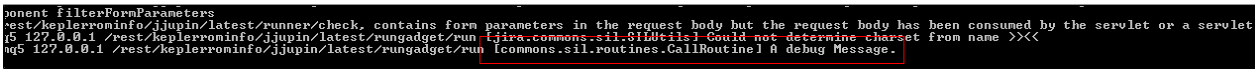


2) **Permanently** - (the logging level will **NOT** be reset after a reboot of Jira), described here: http://confluence.kepler-rominfo.com/display/JJUP/Configure+JIRA+Logging

Example

```
logPrint("DEBUG", "A debug Message.");
```

If the configuration level is one of: **TRACE** or **DEBUG** then the message will be printed out.



Otherwise, if the configuration level is: **INFO, WARN, ERROR** or **FATAL** the the message will not be displayed.

For other usages you can consult the priority level list above.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

Array Routines

Introduction

This section contains a collection of routines for handling arrays.

Routines summary

Routine	Description
arrayAddElement	If **elem** is an element of the array type, returns a new array that includes the specified element at the end.
arrayAddElementIfNotExist	If **elem** is an element of the array type, returns a **new** array that includes the specified element at the end. The element is added **only if it is not already** in the array.
arrayDeleteElement	If ***—elem*** is an element of the array type, returns a new array without the specified element.
arrayDeleteElementAt	If ***—index*** is less than the array size, returns a new array without the element at the specified index.
arrayElementExists	Returns true if the element exists in the array, false otherwise.
arrayGetElement	Returns the element at the specified ***—index***
arraysConcat	Adds the elements of the second array to the first one.
arraySetElement	If ***—elem*** is an element of the array type, returns a new array with the specified element on position ***—index1*** . If **index1** is greater than the array size, it will add empty elements on the missing positions.
arraySize	Returns the size of the given array.
arrayToSet	Converts an array to a set of (unique) elements.
arrayUnion	Union between two arrays.
arrayIntersect	Intersect between two arrays.
arrayDiff	Difference between two arrays. Returns the elements from the first array which don't exist in the second array.
arrayFind	Finds an element inside the collection and returns its index. If the element is not found, it returns -1.
arraySort	Sorts the elements from an array.
arrayKeys	Returns the keys of the array, if array is using the new syntax in 2.5

Notes:

1. Arrays are created by using this construct:
   {<value1>, <value2>, ..., <value3>}
2. Multi-dimensional arrays are not supported.

See Also:

Syntax
Variable Resolution

Operators

arrayAddElement

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

arrayAddElement(arrayName, elem)

Description:

If **elem** is an element of the array type, returns a new array that includes the specified element at the end.

Alias:

addElement(arrayName, elem)

Parameters:

Parameter name	Type	Required	Description
arrayName	array	Yes	The array to which the new element is added
elem	any	Yes	The element to be added. Must be the same type as the array type

Return Type:

array

Example:

Example 1:

```
watchers2 = addElement(watchers, currentUser());
```

The result returned by the routine is assigned to a new array ***—watchers2***, so the initial array ***—watchers*** will not be modified.

Example 2:

```
watchers = addElement(watchers, currentUser());
```

The result returned by the routine is assigned to the same array ***—watchers***, so the initial array ***—watchers*** will be modified.

Note

If **elem** is not the same type as declared in the array definition, the routine returns error.

katl-commons 2.5 specific
Since 2.5 version is easier for you to just use the '+' operator to add elements to an array. **array = array + element** it is a simple and more meaningful way to express it. **array += element** is even better !

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

arrayAddElementIfNotExist

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

arrayAddElementIfNotExist(arrayName, elem)

Description:

If **elem** is an element of the array type, returns a **new** array that includes the specified element at the end. The element is added **only if it is not already** in the array.

Alias:

addElementIfNotExist(arrayName, elem)

Parameters:

Parameter name	Type	Required	Description
arrayName	array	Yes	The array to which the new element is added
elem	any	Yes	The element to be added. Must be the same type as the array type

Return Type:

array

Example:

example 1:

```
watchers2 = addElementIfNotExist(watchers, currentUser());
```

Adds **currentUser** to the **watchers** array if **currentUser** is not already present. The routine returns a new array **watchers2**, so the initial array will not be modified.

example 2:

```
watchers = addElementIfNotExist(watchers, currentUser());
```

The initial array will be modified as a result of the **=** operator and **NOT of the routine call**.

Notes:

1. If **arrayName** is not defined as an array, the routine returns error.
2. If **elem** is not the same type as declared in the array definition, the routine returns error.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

arrayDeleteElement

Availability
This routine is available since **katl-commons 1.0 .**

Description:

If ***—elem*** is an element of the array type, returns a new array without the specified element.

Alias:

deleteElement(arrayName, elem)

Parameters:

Parameter name	Type	Required	Description
arrayName	array	Yes	The array from which the new element is deleted
elem	any	Yes	The element to be deleted. Must be the same type as the array type

Return Type:

array

Example:

```
watchers = deleteElement(watchers, currentUser());
```

The result returned by the routine is assigned to the the same array ***—watchers***, so the initial array ***—watchers*** will be modified.

Notes:

1. If ***—array*** is not defined as an array, the routine returns error.
2. If ***—elem*** is not the same type as declared in the array definition, the routine returns error.

katl-commons 2.5 specific
Since version 2.5, there is a more powerful way to express these operations: **array = array - element;** or even better **array -= element;**

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

katl-commons 2.5 specific

arrayDeleteElementAt

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Description:

If ***—index*** is less than the array size, returns a new array without the element at the specified index.

Alias:

deleteElementAt(array, index)

Parameters:

Parameter name	Type	Required	Description
array	array	Yes	The array from which the element at the specified index is deleted.
index	number	Yes	The index of the element to be deleted.

Return Type:

array

Example:

Example 1:

```
if(size(watchers) >= 2){
 watchers2 = deleteElementAt(watchers, 1);
}
```

The result returned by the routine is assigned to a new array ***—watchers2***, so the initial array ***—watchers*** will not be modified.

Example 2:

```
if(size(watchers) >= 2){
 watchers = deleteElementAt(watchers, 1);
}
```

The result returned by the routine is assigned to the the same array ***—watchers***, so the initial array ***—watchers*** will be modified.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

arrayDiff

> **Availability**
> This routine is available since **katl-commons 2.5.15 / 2.6.7**.

Syntax:

arrayDiff(arrayName1, arrayName2)

Description:

Difference between two arrays. Returns the elements from the first array which don't exist in the second array.

Return Type:

array

Example:

Example 1:

```
string[] array1 = {"a", "b", "c"};
string[] array2 = {"c", "d"};
return arrayDiff(array1, array2);
```

The result will be an array containing elements "a" and "b".

Example 2:

```
string[] developers= usersInGroups({"jira-developers"});
string[] administrators = usersInGroups({"jira-administrators"});
return arrayDiff(developers, administrators);
```

The result is an array which contains only developers that are not also administrators.

arrayElementExists

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

arrayElementExists(array, elem)

Description:

Returns true if the element exists in the array, false otherwise.

Alias:

elementExists(array, elem)

Parameters:

Parameter name	Type	Required	Description
array	array	Yes	The array where the element existence is checked.
element	any	Yes	The element whose existence is checked.

Returns:

boolean

Example:

```
if( elementExists(watchers, currentUser()) ){
    print("You are watching this issue.");
}
```

The routine returns true if ***—currentUser*** is in the ***—watchers*** array and if the result is true a message is printed

Notes:

If ***—elem*** is not the same type as declared in the array definition, the routine returns error.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

arrayFind

Availability
This routine is available since **katl-commons 2.5.7**.

Syntax:

arrayFind(arrayName, element)

Description:

Finds an element inside the collection and returns its index. If the element is not found, it returns -1.

Alias:

arrayFind(arrayName, element)

Parameters:

Parameter name	Type	Required	Description
arrayName	array	Yes	The array.
element	type of array element	Yes	The element to find in the array.

Return Type:

number

Example:

```
return arrayFind(usersInGroups({"jira-users"}), "admin");
```

The result of the routine is the index of the searched element in the array returned by usersInGroups.

If the array contains duplicates, the routine returns the index of the first occurrence of the searched element.

arrayFindBinary

Availability
This routine is available since **katl-commons 2.5.7.**

Syntax:

arrayFindBinary(arrayName, element)

Description:

Binary search on sorted array. If the element is not found, returns -1.

Alias:

arrayFindBinary(arrayName, element)

Parameters:

Parameter name	Type	Required	Description
arrayName	array	Yes	The array.
element	type of array element	Yes	The element to find in the array.

Return Type:

number

Example:

```
return arrayFindBinary(arraySort(usersInGroups({"jira-users"}), false),
"admin");
```

The result of the routine is the index of the searched element in the array returned by arraySort.

If the array contains duplicates, the routine returns the index of the first occurrence of the searched element.

arrayGetElement

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

arrayGetElement(array, index)

Description:

Returns the element at the specified ***—index***

Alias:

getElement(array, index)

Parameters:

Parameter name	Type	Required	Description
array	array	Yes	The array where the element at the specified index is searched.
index	number	Yes	The index of the element to be returned.

Returns:

Element of the array type

Example:

```
for(number i = 0; i < size(watchers); i = i + 1){
    print(getElement(watchers, i) + " is watching this issue.");
}
```

Prints all the elements of the array ***—watchers***

Notes:

1. If **index** is not number or has negative value the routine returns error.
2. If **index** is greater than the size of the array, the routine will return an empty value of the respective type.

katl-commons 2.5 specific
Since 2.5 we added the indexing operator. It will work on arrays, strings, dates and interval.

It was greatly awaited and it's now there.

You can simply write in your programs **watchers[0]** to refer to the first element in the watchers array.

See Also

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

arrayIntersect

Availability
This routine is available since **katl-commons 2.5.7**.

Syntax:

arrayIntersect(arrayName1, arrayName2)

Description:

Intersect between two arrays.

Parameters:

Parameter name	Type	Required	Description
arrayName1	array	Yes	The first array.
arrayName2	array	Yes	The second array.

Return Type:

array

Example:

```
string[] developers= usersInGroups({"jira-developers"});
string[] administrators = usersInGroups({"jira-administrators"});
return arrayIntersect(developers, administrators);
```

The result returned by the routine is an array which contains the unique elements found in both developers and administrators groups.

The above code can be written also as shown below:

```
return usersInGroups({"jira-developers", "jira-administrators"});
```

That's it.

arrayKeys

> **Availability**
> This routine is available since **katl-commons 2.5.8**.

Syntax:

arrayKeys(array)

Description:

Returns the keys of the array, if array is using the new syntax in 2.5

You should remember that keys are not sorted and that index in the returned array do not correspond to index in the original array. The routine only returns a list of added keys.

If the array contains no keys, an empty array is returned.

Parameters:

Parameter name	Type	Required	Description
array	array	Yes	The array.

Return Type:

array of strings

Example:

```
number []arr;
arr['one'] = 1;
arr['two'] = 2;
arr['three'] = 3;
arr[3] = 4;

string [] arrkeys = arrayKeys(arr); // contains strings 'one', 'two',
'three' but not necessary in that order !
```

arraysConcat

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

arraysConcat(array1, array2)

Description:

Adds the elements of the second array to the first one.

Parameters:

Parameter name	Type	Required	Description
array1	array	Yes	The first array to be added.
array2	array	Yes	The second array to be added.

Returns:

array

Example:

```
group={"user1", "user2"};
arraysConcat(watchers, group);
```

Adds the elements of the **group** array to the **watchers** array.

Notes:

> If the array types are incompatible, the routine returns error.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

arraySetElement

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

arraySetElement(array, index1, elem)

Description:

If ***—elem*** is an element of the array type, returns a new array with the specified element on position ***—index1*** . If **index1** is greater than the array size, it will add empty elements on the missing positions.

Alias:

setElement(array, index1, elem)

Parameters:

Parameter name	Type	Required	Description
array	array	Yes	The array where to add the element
index1	number	Yes	Position where to add the element

elem	type of array element	Yes	Element to add to the array

Returns:

Element of the array type

Example:

```
watchers = setElement(watchers, 12, currentUser());
```

Sets the value ***—currentUser*** for the 13 th element of the array ***—watchers***

Notes:

1. If **array** is not defined as an array, the routine returns error.
2. If **index** is not number the routine returns error.

katl-commons 2.5 specific
Since 2.5 we added the indexing operator. It will work on arrays, strings, dates and interval.

It was greatly awaited and it's now there.

You can simply write in your programs **watchers[12] = currentUser()** to refer to the 13 th element in the watchers array.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

arraySize

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

arraySize(array)

Description:

Returns the size of the given array.

Alias:

size(array)

Parameters:

Parameter name	Type	Required	Description
array	array	Yes	The array whose size will be returned

Returns:

number

Example:

```
size(watchers); // returns the number of watchers
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

arraySort

> **Availability**
> This routine is available since **katl-commons 2.5.7**.

Syntax:

arraySort(arrayName, reverse)

Description:

Sorts the elements from an array.

Alias:

arraySort(arrayName, reverse)

Parameters:

Parameter name	Type	Required	Description
arrayName	array	Yes	The array.
reverse	boolean	Yes	true if reverse order, false otherwise

Return Type:

array

Example:

```
watchers = arraySort(watchers, false);
```

The result returned by the routine is an array which contains the elements from watchers sorted in the ascending order.

arrayToSet

> **Availability**
> This routine is available since **katl-commons 2.5.7**.

Syntax:

arrayToSet(arrayName)

Description:

Converts an array to a set of (unique) elements.

Alias:

arrayToSet(arrayName)

Parameters:

Parameter name	Type	Required	Description
arrayName	array	Yes	The array to be converted to a set of elements.

Return Type:

array

Example:

```
watchers = arrayToSet(watchers);
```

The result returned by the routine is an array which contains the elements from watchers without duplicates.

arrayUnion

> **Availability**
> This routine is available since **katl-commons 2.5.7**.

Syntax:

arrayUnion(arrayName1, arrayName2)

Description:

Union between two arrays.

Parameters:

Parameter name	Type	Required	Description
arrayName1	array	Yes	The first array.
arrayName2	array	Yes	The second array.

Return Type:

array

Example:

```
string[] developers= usersInGroups({"jira-developers"});
string[] administrators = usersInGroups({"jira-administrators"});
return arrayUnion(developers, administrators);
```

The result returned by the routine is an array which contains the unique elements from developers and administrators.

JIRA Integration

Introduction

This section contains routines that interact with the JIRA environment.

Routines summary

Routine	Description

addComment	Posts a comment on the specified issue on behalf of the specified user. Returns a number representing the id of the comment.
allProjects	Returns a string array with the keys of all the projects in JIRA.
attachFile	Add an attachment to a selected issue.
autotransition	Executes a transition and moves to the specified step. It will execute the transition only if the transition is valid for the current status of the issue.
changeSubtaskOrder	Changes subtask position.
cloneIssue	Duplicates the issue and returns the key of the duplicated issue. If specified, it also creates a link to the duplicated issue. Starting with katl-commons version 2.5.15 / 2.6.7 it accepts a 3rd parameter. However, **be warned** that using this parameter you can a) create subtasks in some other projects and b) create infinite subtasks. This may render **JIRA unusable**. *Make sure the parent and the subtask will be in the same project !Make sure you do not create a subtask with another subtask as parent!*
createIssue	Creates an issue based on the provided arguments.
createWebLink	Creates a web link on the issue.
currentUser	Returns the key for the user that invoked the script containing **currentUser**.
deleteIssue	Deletes the selected issue.
fieldHistory	Returns all the pairs **date + value** for the selected field from the selected issue's history.
getAvailableTransitions	Retrieves all available transitions for the current user from the given issue state.
getWorkflowStatusIds	Retrieves an unique list of statuses (ids) for a given workflow
getComment	Returns the comment entered in a transition screen, or an empty string if no comment has been entered or the transition doesn't have a screen.
getFieldOptions	Returns a string array representing the list of options for a custom field. This routine only handles CFs of the following types: single select, multi select, radio buttons and checkboxes (otherwise throws an exception).
getIssueFields	Returns a map with all standard and custom fields of an issue. The map contains pairs of field name and field values.
getTeamLeaders	Returns the **team leaders** on the specified project (all the **component leads**).
getProjectComponentLead	Returns the **leader** of the specified component from the specified project.
getTimeSpent	Gets the time spent (logged) on an issue by a certain user or group.
getUserProperty	Retrieves properties of users.
userLanguage	Returns the language for an user.
groupExists	Verifies if the selected group is a registered JIRA group.
hasInput	Verifies if a field **had input in the transition screen**. Should **only** be used in **transitions** that require a screen.
hasPermission	Checks if a user has the specified permission
hasUserProperty	Checks if the user has the given property set.
isAnyUserAuthenticated	Verifies if there is a logged in user.
isIssueContext	Verifies if the script is running in an issue context
isTeamLeader	Verifies if the specified user is a **team leader** on the project (if it is a **component lead**).
lastFieldHistory	Returns the last change details (**user | date | field | oldValue | newValue**) from the selected issue's history.
linkIssue	Links two issues by a specified link type name. First issue will have the outward description of the link type, the second issue will have the inward description of the link type.
linkedIssues	Returns an array with the Issue names linked with the specified one.

projectMembers	Returns a list with all the user names of the users who have a role in the specified project.
projectsForPM	Returns all the projects that the selected user has the role of **project manager** (**project lead**).
projectsForUser	Returns all the projects for which this user has permission to **assign** or to **be assigned** issues.
projectPM	Returns the username of the **project manager** (**project lead**) of the selected project, if exists.
raiseEvent	Triggers an event to be processed by listeners.
runAs	Assumes a user when running a script.
selectIssues	Returns an array with the keys of the issues that matched the search query.
selectIssuesByFilter	Returns an array with the keys of the issues obtained by running the given filter.
setUserProperty	Sets properties of users.
subtasks	Get the list of sub tasks linked to the parent issue.
unlinkIssue	Removes the specified link between two issues.
userEmailAddress	Returns the email address of the selected user. The email address may be needed to supply it to various external systems.
userExists	Verifies if the selected user is registered JIRA user.
userFullName	Returns the full name (firstname, lastname) of the user
userGroups	Returns the groups in which the selected user belongs to.
userInGroup	Verifies if the selected user is in the selected group(s).
userRoles	Returns the roles of the provided user in the project.
usersInGroups	Returns a list of users **common to all** the **specified groups**.
usersInRole	Returns the users that corespond to a certain role on the specified project.
issueStatePush	Saves fields of an issue (standard and custom fields) into a stack. You can later pop these states from the stack to set the saved values back on the issue.
issueStatePop	Removes and retrieves the top issue state from the stack (the latest saved state).
issueStatePopAndSet	Removes, retrieves and sets the top issue state from the stack (the latest saved state).
issueStateFlush	Removes all the saved states for a given issue.
issueTypesForProject	Retrieves the issue types for the project with the given key.
projectsWithPermissionForUser	Retrieves the project keys on which the given user has the given permission.
getAllCommentIds	Gets all the comment ids which are already entered on an issue. Opposed to the getComment() routine, this one returns what is already entered on the issue. Comments are guaranteed to be sorted from the oldest one to the newest one (i.e. the last comment is at the end of the array, first comment is the first in the list).
getCommentById	Gets all the comment properties for a given id.
getUserDirectoryName	Returns the directory name to which the user belongs to.
userHasAccessToComment	Verifies if a comment is visible for an user.
renderWiki	Returns the html code for the wiki text.

addComment

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

addComment(issue, username, comment)

or

addComment(issue, username, comment, securityLevel)

Description:

Posts a comment on the specified issue on behalf of the specified user. Returns a number representing the id of the comment.

Parameters:

Parameter name	Type	Required	Description
issue key	String	Yes	the key of the selected issue
user name	String	Yes	the user name of the selected user
comment	String	Yes	the comment that will be post on the selected issue
securityLevel	String	No	the security level of comment

Return type:

number

The returned number represents the id of the comment. We decided to return this for future use or for third-party custom SIL routines.

Example:

Example 1:

```
addComment(key, currentUser(), "I have executed a transition.");
```

Adds a comment on the current issue, on behalf of the current user.

Example 2:

```
addComment(key, currentUser(), "you can't see me", "Administrators");
```

Adds a comment on the current issue, on behalf of the current user, viewable only by "Administrators".

Notes:

> The routine first check if exists any group with the name provided, if so, it will apply to group, else will check if exists any project role with the same name and if this exists will apply to that project role.
>
> If you have any group and project role with same name, to make the securityLevel apply to project role you should provide the project role id as number.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

addGroupToProjectRole

> **Availability**
> This routine is available since **katl-commons 3.0.8**.

Syntax:

addGroupToProjectRole(group, project, role)

Description:

Adds a single group to a project role if the group is not already in that role.

Parameters:

Parameter name	Type	Required	Description
Group	String	Yes	The group name
Project key	String	Yes	The key of the selected project
Role name	String	Yes	The name of the role to add the user to

Return type:

boolean

true if operation succeeded

Example:

Example 1:

```
addGroupToProjectRole("dev-group", "TEST", "Developers");
```

See Also:

addUserToGroup

> **Availability**
> This routine is available since **katl-commons 2.5.16** (JIRA 5.x) **/ 2.6.8** (JIRA 6).

Syntax:

addUserToGroup(user, group)

Description:

Adds a single user to a group if the user is not already in that group.

Parameters:

Parameter name	Type	Required	Description
User	String	Yes	The username or userkey
Group	String	Yes	The name of the group

Return type:

boolean

true if operation succeeded

Example:

Example 1:

```
addUserToGroup("user.3", "Senior Developers");
```

Notes:

The look-up is first made after the userkey, then after the username.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

addUserToProjectRole

Availability
This routine is available since **katl-commons 2.5.15** (JIRA 5.x) **/ 2.6.7** (JIRA 6).

Syntax:

addUserToProjectRole(user, project, role)

Description:

Adds a single user to a project role if the user is not already in that role.

Parameters:

Parameter name	Type	Required	Description
User	String	Yes	The username or userkey
Project key	String	Yes	The key of the selected project
Role name	String	Yes	The name of the role to add the user to

Return type:

~~**none**~~

~~the returned value has no meaning~~

boolean (since 2.5.16 / 2.6.8)

true if operation succeeded

Example:

Example 1:

```
addUserToProjectRole("user.3", "TEST", "Developers");
```

Notes:

The look-up is first made after the userkey, then after the username.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

allProjects

> **Availability**
> This routine is available since **katl-commons 2.0.2** (for JIRA 5.x) or **katl-commons 1.1.9** (for JIRA 4.3.x and 4.4.x) .

Syntax:

allProjects()

Description:

Returns a string array with the keys of all the projects in JIRA.

Parameters:

None

Return type:

string []

Returns a list of **keys** for all the projects in JIRA.

Example:

Example

```
string [] projects = allProjects();
print(projects);
```

Results: prints a list with all the project keys in JIRA. If we have 2 projects *PRJ* and *TST*, the logs will show *PRJ|TST*

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

attachFile

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

attachFile(path_to_file, issue)

Description:

Add an attachment to a selected issue.

Parameters:

Parameter name	Type	Required	Description
path to file	string	Yes	the absolute path to the file
issue key	string	Yes	the key of the issue to which the file will be attached

Return type:

boolean (true/false)

The return value represents the success of the attachment process. If the routine returns **true**, the file was attached successfully.

Example:

Example 1:

```
string path_to_file;
string issue;
path_to_file = "C:/jira/home/attachment/file_to_attach_1.jpg";
issue = "PRJ-239";
attachFile(path_to_file, issue);
```

Result: Returns **True** if the file is at the selected location and the issue exists, meaning the file was attached. Returns **False** if any of the conditions stated before are not met.

Notes:

1. Please use forward slashes ("/") for the path.
2. The path to the attachment must be absolute and point to a location on the server.
3. If an error occurs, the routine will return false and the error message will be visible in the log.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

attachFileFromURL

Availability
This routine is available since **katl-commons 3.0.2 .**

Syntax:

attachFileFromURL(url_to_file, issue)

Description:

Add an attachment located on an URL path to a selected issue.

Parameters:

Parameter name	Type	Required	Description
url to file	string	Yes	The URL path to the file
issue key	string	Yes	The key of the issue to which the file will be attached

Return type:

boolean (true/false)

The return value represents the success of the attachment process. If the routine returns **true**, the file was attached successfully.

Example:

```
string url_to_file;
string issueKey;
url_to_file =
"http://otherServer/generateForm.aspx?PackageName=customField_10192";
issueKey = "PRJ-239";
attachFileFromURL(url_to_file, issueKey);
```

Result: Returns **True** if the file is at the selected location and the issue exists, meaning the file was attached. Returns **False** if any of the conditions stated before are not met.

Notes:

1. If an error occurs, the routine will return false and the error message will be visible in the log.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

autotransition

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

autotransition(transition, issueKey)

New Parameter
Since katl-commons 2.5.3, the "skipStateCheck" parameter was added, which allows to force the target transition even if the issue is currently inside another transition and has been modified.

autotransition(transition, issueKey, skipStateCheck)

Description:

Executes a transition and moves to the specified step. It will execute the transition only if the transition is valid for the current status of the issue.

If the **skipStateCheck** parameter is set to true, the routine will force the target transition even if the issue is currently inside another transition and has been modified

Create Issue Transition
In order to use the routine on the **Create Issue** transition, it is very important that you place the SIL post function **after** the "Creates the issue originally" post function but **before** "Fire a Issue Created event that can be processed by the listeners.". Positioning the SIL post function otherwise will de-synchronize the issue from the workflow - will set an invalid status for the workflow step it is in.

Parameters:

Parameter name	Type	Required	Description
transition	string	Yes	The id or the name of the transition that will be executed
issuekey	string	Yes	The key of the issue for which the transition will be executed
skipStateCheck	boolean	No	If set to true, will force the target transition even if the issue is currently inside another transition and has been modified.

Return type:

boolean (true/false)

If the return value is **true**, the transition was executed successfully. A **false** return value means that the transition failed. In this case, you should check the log for additional details.

> Since version 2.5.3, autotransitions are executed after the script has finished. Since we do not know the result of the operation at the time the routine is called inside the script, it will return true by default.

Example:

Example 1:

```
autotransition(121, "PRJ-123");
```

Return: **True** if the transition with the **ID=121** for the issue **PRJ-123** was executed. **False** if the transition wasn't executed.

> Since version 2.5.3, will always return true.

Example 2:

```
autotransition("Require information", "PRJ-232");
```

Return: **True** if the transition **Require information** for the issue **PRJ-232** was executed. **False** if the transition wasn't executed.

> Since version 2.5.3, will always return true.

Notes:

1. If the transition requires a screen, **it will not be displayed** for additional input.
2. You should do **all** issue modifications **in** or **before** the post function that contains the calling of the auto transition.
3. All post functions executed by the called transitions **should not modify** the issue. These post functions could be used for sending notifications.
4. If there are validations and/or conditions in the transition called after the one that contains the auto transition, and these **validat ions fail**, the transition **will NOT be executed.**

Frequent Problems

Problem: When executing an autotransition, it fails with the message: **It seems that you have tried to perform a workflow operation (<Transition_Name>) that is not valid for the current state of this issue (<Issue_Key>). The likely cause is that somebody has changed the issue recently, please look at the issue history for details.**

Solution: Set the **skipStateCheck** parameter to true.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

changeSubtaskOrder

> **Availability**
> This routine is available since **katl-commons 3.0.2 .**

Syntax:

changeSubtaskOrder(subtaskKey,position)

Description:

Changes subtask position.

Parameters:

Parameter name	Type	Required	Description
subtaskKey	String	Yes	the key of the sub-task
position	Number	Yes	position of sub-task

Return type:

boolean (true/false)

A **true** return value means that the position of the sub-task was changed.

Example:

Example 1:

```
return changeSubtaskOrder("TSTAG-5",2);
```

Changes the position of sub-task TSTAG-5 to position 2.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

cloneIssue

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

cloneIssue(issueKey, [issueLinkTypeName [, newParentKey]])

Description:

Duplicates the issue and returns the key of the duplicated issue. If specified, it also creates a link to the duplicated issue.

Starting with katl-commons version 2.5.15 / 2.6.7 it accepts a 3rd parameter. However, **be warned** that using this parameter you can a) create subtasks in some other projects and b) create infinite subtasks. This may render **JIRA unusable**. *Make sure the parent and the subtask will be in the same project !Make sure you do not create a subtask with another subtask as parent!*

Parameters:

Parameter name	Type	Required	Description
issuekey	string	Yes	The key of the issue that has to be cloned
issueLinkTypeName	string	No	The link to the duplicated issue
newParentKey	string	No	The new parent, if subtask

Return type:

string

The return value represents the key of the duplicated issue.

Example:

Example 1:

```
cloneIssue("PRJ-343");
//"PRJ-343" represents the key of the issue that will be duplicated.
```

Result: Issue **PRJ-343** is duplicated.

Example 2:

```
cloneIssue("PRJ-267","Cloned Issue")
```

Result: Issue **PRJ-267** is duplicated and a link named **Cloned Issue** to the duplicated issue is created.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

countIssues

> **Availability**
> This routine is available since **katl-commons 2.5.15 / 2.6.7 .**

Syntax:

countIssues(jql)

Description:

Returns the number of issues that matched the search query.

Parameters:

Parameter name	Type	Required	Description
search query	string	Yes	search query containing the details of the returned issues.

Return type:

number

Returns the number of **issue keys** that match the specified jql.

Example:

Example 1:

```
string jql;
jql = "project = PRJ AND reporter in membersOf('Employees') AND status in
(Open, 'In Progress', Reopened, Resolved, 'On hold', Assigned, 'Internal
QA', 'Results rejected', 'Tested and not delivered', 'Tested and
delivred')";
countIssues(jql);
```

Result: The number of issues that are in the project **PRJ**, with the reporter included in **Employees** group and the status being one of the above.

Notes:

> For the search query, please use apostrophe(') instead of quotes(") and double backslashes(\ \ )for escaping characters (instead of a single backslash).

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

createIssue

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

createIssue(projectKey, parentIssueKey, issueType, issueSummary)

or

createIssue(projectKey, parentIssueKey, issueType, summary, priority, description, components, due date, estimate, security_level, custom_fields_mappings)

Since version 3.0.2:

createIssue(projectKey, parentIssueKey, issueType, summary, priority, description, components, due date, estimate, security_level, field_mappings)

Description:

Creates an issue based on the provided arguments.

Parameters:

Parameter name	Type	Required	Description
projectKey	string	Yes	The key of the project, in which the issue will be created, as it is saved in the Administration part
parentIssueKey	string	Yes	The key of the parent issue. Though the parameter is required, it can take an empty value to specify that the issue is not a sub-task, but a regular issue.
issueType	string	Yes	The type of the issue that will be created
summary or issueSummary	string	Yes	The summary of the issue that will be created
priority	string	No	The priority
description	string	No	The description of the issue that will be created

components	array of strings	No	The components of the issue that will be created
due date	date	No	The due date of the issue that will be created
estimate	interval	No	The original estimate of the issue that will be created
security level	string	No	The security level of the issue that will be created
custom fields mappings/ field mappings (since version 3.0.2)	array of strings	No	The mappings of the custom field of the issue that will be created. Since version 3.0.2 the mappings of the custom and system fields of the issue that will be created.

Return type:

string (the key of the created issue)

Example:

Example 1:

```
string issue_priority;//Possible values: "Major", "Critical" etc.
string issue_description;
string[] issue_components;
string issue_security_level;
string[] custom_fields_mapping;

issue_priority = "Critical";
issue_description = "Description of the issue";
issue_components = components; //an array containing all the components of
the current project
issue_security_level = "Administrator";
custom_fields_mapping = "STDUP|fmanaila|STDGP|jira-users";
string k = createIssue(
            "PROJECT",
            "PRJ-300",
            "Sub-task",
            "Summary of the sub task" ,
            issue_priority,
            issue_description,
            issue_components,
            currentDate() + "30d",
            "1h 30m",
            issue_security_level,
            custom_fields_mapping
           );
print ("On the project " + project + ", issue " + k + "is created.");
```

Result: *On the project PROJECT, issue PRJ-300 is created.* Issue details are the ones declared above.

Example 2

```
string issue_priority;//Possible values: "Major", "Critical" etc.
string issue_description;
string issue_security_level;

issue_priority = "Critical";
issue_description = "Description of the issue";
issue_security_level = "Administrator";
string k = createIssue(
            "PROJECT",
            "", // passing in empty to create a regular issue rather than
subtask
            "Bug",
            "Summary of the sub task" ,
            issue_priority,
            issue_description,
            {}, // no components
            currentDate() + "30d",
            "1h 30m",
            issue_security_level,
            {} // and no custom field mappings
           );
print ("On the project " + project + ", issue " + k + "is created.");
```

Will create a Bug with no components and no special custom field values (all defaults).

Example 3

```
string issue_priority;//Possible values: "Major", "Critical" etc.
string issue_description;
string issue_security_level;
string[] assigneeUser;

issue_priority = "Critical";
issue_description = "Description of the issue";
issue_security_level = "Administrator";
assigneeUser = "assignee|someUserName";
string k = createIssue(
            "PROJECT",
            "", // passing in empty to create a regular issue rather than
subtask
            "Improvement",
            "Summary of the sub task" ,
            issue_priority,
            "",
            {}, // no components
            currentDate() + "30d",
            "1h 30m",
            issue_security_level,
   assigneeUser
  );
print ("On the project " + project + ", issue " + k + "is created.");
```

Will create an Improvement with no components and the assignee set for user with the username "someUserName".

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

createUser

Availability
This routine is available since **katl-commons 3.1.1/katl-commons 3.0.13.**

Syntax:

createUser(username, displayName, email, password)

Description:

Creates a new user.

Creates the user and configures it with the username, displayName, email and password.

Parameters:

Parameter name	Type	Required	Description
username	string	yes	The username
displayName	string	yes	The display name

email	string	yes	The email
password	string	yes	The password

Return type:

boolean

True if the user was created, false if not

Example:

```
createUser("user", "fullname", "user@example.com", "password");
```

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

createWebLink

> **Availability**
> This routine is available since **katl-commons 2.0** (for JIRA 5.x) .

Syntax:

createWebLink(issueKey, url, title)

Description:

Creates a web link on the issue.

Parameters:

Parameter name	Type	Required	Description
issue key	String	Yes	the key of the selected issue
url	String	Yes	the URL of the link
title	String	Yes	The name/title of the link as it should appear on the issue

Return type:

none

The returned value has no meaning.

Example:

Example 1:

```
createWebLink(key, "http://www.google.com", "Super-duper search");
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

currentUser

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

currentUser()

Description:

Returns the key for the user that invoked the script containing **currentUser**.

Return type:

string

The return value represents the **key** of the user that triggered the script (usually executed a transition).

Example:

Example 1:

```
customfield = currentUser();
```

Result: customfield = <key of the current user>

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

currentUserKey

> **Availability**
> This routine is available since **katl-commons 3.0.3.**

Syntax:

currentUserKey()

Description:

Returns the key for the user that invoked the script containing **currentUserKey**.

Return type:

string

The return value represents the **key** of the user that triggered the script (usually executed a transition).

Example:

Example 1:

```
customfield = currentUserKey();
```

Result: customfield = <key of the current user>

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

currentUsername

> **Availability**
> This routine is available since **katl-commons 3.0.3 .**

Syntax:

currentUsername()

Description:

Returns the username for the user that invoked the script containing **currentUsername**.

Return type:

string

The return value represents the **username** of the user that triggered the script (usually executed a transition).

Example:

Example 1:

```
customfield = currentUsername();
```

Result: customfield = <username of the current user>

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

deleteIssue

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

deleteIssue(issuekey)

Description:

Deletes the selected issue.

Parameters:

Parameter name	Type	Required	Description
issuekey	string	Yes	The key of the issue that has to be deleted

Return type:

None. The returned value has no meaning.

Example:

Example 1:

```
deleteIssue("PRJ-323");
```

Result: If the current issue **is not** "PRJ-323" or **a child of** "PRJ-323", then the issue "PRJ-323" will be deleted

Example 2:

```
deleteIssue(key);
```

Result: Since "key" variable returns the key of the current issue, the delete operation **will fail**

Notes

> Cannot delete the current issue or the parent of the current issue.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

deleteWebLinkById

> **Availability**
> This routine is available since **katl-commons 2.5.15** (JIRA 5.x) **/ 2.6.7** (JIRA 6).

Syntax:

deleteWebLinkById(webLinkId)

Description:

Removes a web link

Parameters:

Parameter name	Type	Required	Description
webLinkId	String	Yes	the id of a web link

Return type:

none

The returned value has no meaning

Example:

Example 1:

```
number [] ids = getWebLinksForIssue(key);

for(number wlid in ids){
 string [] props = getWebLinkById(wlid);
 if(props[0] == theOtherKey) {
  deleteWebLinkById(wlid);
 }
}
```

See Also:

---

editComment

> **Availability**
> This routine is available since **katl-commons 3.0.10.**

Syntax:

editComment(commentId, comment)

or

editComment(commentId, comment, securityLevel)

Description:

Edits a comment with the specified id and text. Optional, you can edit the security level for comment. Returns the comment representation after the edit.

Parameters:

Parameter name	Type	Required	Description
commentId	number	Yes	the id of the issue comment
comment	String	Yes	the text that will be post on the selected comment
securityLevel	String	No	the security level of comment

Return type:

JComment

The comment representation with the following keys:

Key name	Description
id	the id of the issue comment
text	the comment text

author	the author of the comment
created	creation date, as string, can be assigned to a date variable
updatedBy	the updater, or empty string if there is no updater
updated	updated date, or empty string if there's no update

Example:

Example 1:

```
string newComm = "This comment was edited.";
return editComment(11801, newComm);
```

Edits the comment with id 11801 with the specified text .

Example 2:

```
JComment jcomment = getCommentById(11801);
string oldComm = jcomment["text"];
string newComm = oldComm + "\nEditing the comment";
return editComment(11801, newComm, "Developers");
```

Edits a comment by adding new text to the old one and which can be viewable only by "Developers".

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

fieldHistory

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

fieldHistory(key, history_field_name)

Description:

Returns all the pairs **date + value** for the selected field from the selected issue's history.

Parameters:

Parameter name	Type	Required	Description
issue key	String	Yes	the key of the selected issue
field name	String	Yes	the name of the selected field

Return type:

string []

The return value is an array of strings. The strings come in pairs; the first value is a date representing the time when the value was modified and the second value is the content of the requested field at that date.

Example:

Example 1:

```
//values for field Amount are 2000 at 12.02.2011, 3000 at 13.03.2011 and
4000 at 10.05.2011
string field_name;
string[] field_history;
field_name = "Amount";
field_history = fieldHistory(key, field_name);
```

Result: |12/02/2011|2000|13/03/2011|3000|10/05/2011|4000|

Notes:

1. Beside the labels of the custom fields, also the name of the standard JIRA fields (summary, assignee etc.) can be used as parameters.
2. If it returns an empty array, you must get the last value of the field from the issue.
3. The second parameter is the field name as it appears in history.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

getAllCommentIds

Availability
This routine is available since **katl-commons 2.5.10 / 2.6.2 .**

Syntax:

getAllCommentIds(key);

Parameters:

Parameter name	Type	Required	Description
key	String	Yes	the key of the issue

Description:

Gets all the comment ids which are already entered on an issue. Opposed to the getComment() routine, this one returns what is already entered on the issue.

Comments are guaranteed to be sorted from the oldest one to the newest one (i.e. the last comment is at the end of the array, first comment is the first in the list).

Return type:

number []

Ids of the comments on the issue represented by *key* parameter.

Example:

Example 1:

```
number [] arr = getAllCommentIds(argv["key"]);
for(number x in arr) {
 string [] cmt = getCommentById(x);
 runnerLog("Got comment:" + cmt["text"]);
}
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

getAttachmentPath

> **Availability**
> This routine is available since **katl-commons 2.5.13 / 2.6.5 .**

Syntax:

getAttachmentPath(issue key, attachment name)

Description:

Get the file path for the attachments with name <attachment name> attached to <issue key>.

Parameters:

Parameter name	Type	Required	Description
issue key	string	Yes	The issue key
attachment name	string	Yes	The attachment name

Return type:

string[]

The file paths.

> Note that the return type is string[], not string. You will get an array of file paths when you have multiple attachments with the same file name.

Example:

```
string [] filePaths = getAttachmentPath("PROJECT-15", "someFile.txt"); //
returns all the file paths for the "someFile.txt" attachments
```

Notes:

1. If there is no issue with that issue key, an exception will be raised.
2. The paths returned are ordered by date, starting with most recent attachment added.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getAvailableTransitions

> **Availability**
> This routine is available since **katl-commons 2.5.13 / 2.6.5.**

Syntax:

getAvailableTransitions(issueKey)

Description:

Retrieves all available transitions for the current user from the given issue state.

Parameters:

Parameter name	Type	Required	Description
issue key	String	Yes	the key of the selected issue

Return type:

string []

The return value is an array of strings, containing the names of the available transitions for the current user from the given issue state.

Example:

Issue DEMO-1 has status Open and is assigned to testuser. Currently logged in user is admin. Start Progress transition is restricted only to assignee, therefor it will be omitted.

```
return getAvailableTransitions("DEMO-1");
```

Result: Resolve Issue|Close Issue

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getComment

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

getComment();

Description:

Returns the comment entered in a transition screen, or an empty string if no comment has been entered or the transition doesn't have a screen.

Return type:

string

The returned value is the comment entered in the transition screen. If the transition doesn't have a screen or no comment was entered, an empty string is returned.

Example:

Example 1:

```
getComment();
```

Returns: The comment entered **in the transition screen**, or an empty string.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

getCommentById

> **Availability**
> This routine is available since **katl-commons 2.5.10 / 2.6.2 .**

Syntax:

getCommentById(id);

Parameters:

Parameter name	Type	Required	Description
id	number	Yes	the id of the issue comment

Description:

Gets all the comment properties for a given id.

Return type:

Comment

The comment representation with the following keys:

Key name	Description
id	the id of the issue comment
text	the comment text
author	the author of the comment
created	creation date, as string, can be assigned to a date variable
updatedBy	the updater, or empty string if there is no updater
updated	updated date, or empty string if there's no update

Example:

Example 1:

```
number [] arr = getAllCommentIds(argv["key"]);
for(number x in arr) {
 Comment cmt = getCommentById(x);
 runnerLog("Got comment:" + cmt["text"]);
}
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

getCustomFieldNameById

> **Availability**
> This routine is available since **katl-commons 3.0.11 .**

Syntax:

getCustomFieldNameById(customfieldId)

Description:

Gets the customfield name by id.

Parameters:

Parameter name	Type	Required	Description
customfieldId	number	Yes	The customfield id

Return type:

string

Returns the transition name for the workflow name and transition id provided.

Example:

```
return getCustomFieldNameById(10000);
```

See Also:

- sql
- dynamic
- id
- dbcf
- kb-how-to-article
- query
- workflow
- transition
- issue
- comment

- customfield
- edit
- status

getFieldOptions

> **Availability**
> This routine is available since **katl-commons 3.0 .**

Syntax:

getFieldOptions(projectKey, issueType, fieldName)

Description:

Returns a string array representing the list of options for a custom field.
This routine only handles CFs of the following types: single select, multi select, radio buttons and checkboxes (otherwise throws an exception).

Parameters:

Parameter name	Type	Required	Description
project key	String	Yes	the key of the selected project
issue type	String	Yes	the type of the selected issue
field name	String	Yes	the name of the selected field

Return type:

string[]

The returned string array represents the list of options for a custom field.

Example:

Example 1:

```
getFieldOptions("DEMO", "Bug", "RB - BUG");
```

Gets the options for a custom field named "RB - BUG", which exists in the project with the key "DEMO" and has its issue type "BUG".

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

getIssueFieldNames

> **Availability**
> This routine is available since **katl-commons 3.0.8 .**

Syntax:

getIssueFieldNames(issueKey)

or

getIssueFieldNames(issueKey, getNullFields)

Description:

Returns a list with the names of all standard and custom fields of an issue.

Parameters:

Parameter name	Type	Required	Description
issueKey	string	Yes	the key of the selected issue
getNullFields	boolean	No	flag for specifying whether the fields with null values should be retrieved too. If not specified, it defaults to false.

Return type:

string []

The return value is a string array containing all fields values for the selected issue. Each value can be retrieved from the array by key (the field id).

Example:

Example 1:

```
string[] fields = getIssueFieldNames("DEMO-1");
return fields;
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getIssueFields

> **Availability**
> This routine is available since **katl-commons 2.5.15 / 2.6.7 .**

Syntax:

getIssueFields(issueKey)

or

getIssueFields(issueKey, getNullFields)

Description:

Returns a map with all standard and custom fields of an issue. The map contains pairs of field name and field values.

Parameters:

Parameter name	Type	Required	Description
issueKey	string	Yes	the key of the selected issue
getNullFields	boolean	No	flag for specifying whether the fields with null values should be retrieved too. If not specified, it defaults to false.

Return type:

string []

The return value is a string array containing all fields values for the selected issue. Each value can be retrieved from the array by key (the field id).

Example:

Example 1:

```
string[] fields = getIssueFields("DEMO-1");
return fields["summary"];
```

Example 2:

We can use the routine to partially clone an issue, by copying only some of the fields from the original issue:

```
string[] fields = getIssueFields("TP-1", true);
string issue = createIssue(fields["project"], fields["parent"],
fields["issueType"], fields["summary"] + " - part 2");
%issue%.customfield_10000 = fields["customfield_10000"];
%issue%.description = "Partial clone of issue TP-1";
return issue;
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getIssueLinksDetail

Availability
This routine is available since **katl-commons 3.0.13** (for JIRA 6.x use) and **katl-commons 3.1.0** (for JIRA 7.x use).

Syntax:

getIssueLinksDetail(issueKey)

Description:

Returns all the details about the links of an issue key.

Parameters:

Parameter name	Type	Required	Description
issueKey	string	yes	The issue key

Return type:

JIssueLink[]

Example 1:

```
return getIssueLinksDetail("TEST-1");
```

Example 2:

```
JIssueLink[] jIssueLinks = getIssueLinksDetail("TEST-1");
for(JIssueLink jIssueLink in jIssueLinks){
    runnerLog("The linked issue " + jIssueLink.issue + " it has the
following properties: ");
    runnerLog("- link id : " + jIssueLink.id);
    runnerLog("- link name : " +  jIssueLink.name);
    runnerLog("- direction : " + jIssueLink.direction);
    runnerLog("- description : " + jIssueLink.description);
}
```

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getLastComment

Availability
This routine is available since **katl-commons 2.5.15 / 2.6.7 .**

Syntax:

getLastComment(issueKey);

Parameters:

Parameter name	Type	Required	Description
issueKey	string	Yes	the issue key

Description:

Gets all the comment properties for the last issue comment.

Return type:

JComment

The comment representation with the following keys

Key name	Description
id	the id of the issue comment
text	the comment text
author	the author of the comment
created	creation date, as string, can be assigned to a date variable
updatedBy	the updater, or empty string if there is no updater
updated	updated date, or empty string if there's no update
securityLevel	the security level of the comment

Example:

Example 1:

```
JComment cmt = getLastComment("DEMO-1");
runnerLog("Got comment:" + cmt["text"]);
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

getProjectComponentLead

> **Availability**
> This routine is available since **katl-commons 2.5.19 / 2.6.11.**

Syntax:

getProjectComponentLead(pkey, compname)

Description:

Returns the **leader** of the specified component from the specified project.

Parameters:

Parameter name	Type	Required	Description
pkey	String	Yes	The key of the selected project
compname	String	Yes	The name of the component

Return type:

string

Returns the **lead** of the specified component from the specified project.

Example:

```
//lead of the component comp1 from the project with key TSTPRJ is TL1
string leader = getProjectComponentLead("TSTPRJ", "comp1");
```

Result: TL1

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getProjectKeyByName

> **Availability**
> This routine is available since **katl-commons 2.5.13 / 2.6.5**.

Description:

Retrieves the key for the project with the given name.

Parameters:

Parameter name	Type	Required	Description
projectName	String	Yes	the project name

Return type:

string

The return value is a string representing the project key.

Example:

Project with key "DEMO" has the name "Demonstration Project".

```
return getProjectKeyByName("Demonstration Project");
```

Result: DEMO

getStatusNameById

> **Availability**
> This routine is available since **katl-commons 3.0.11 .**

Syntax:

getStatusNameById(statusId, issueKey)

Description:

Gets the status name by the status id and the issue key.

Parameters:

Parameter name	Type	Required	Description
status id	string	Yes	The id of the status
issue key	string	Yes	The key of the issue

Return type:

string

Returns the status name for the the status id and the issue key provided.

Example:

```
return getStatusNameById("1", "DEMO-1");
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getTeamLeaders

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

getTeamLeaders(project)

Description:

Returns the **team leaders** on the specified project (all the **component leads**).

Parameters:

Parameter name	Type	Required	Description
project key	String	Yes	The key of the selected project

Return type:

string []

Returns a list of all the **component leads** from all the components in the specified project.

Example:

Example 1:

```
//team leaders of the current project are TL1, TL2
string[] team_leaders;
team_leaders = getTeamLeaders(project);
```

Result: |TL1|TL2|

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

getTimeSpent

> **Availability**
> This routine is available since **katl-commons 1.1.2 .**

Syntax:

getTimeSpent(issueKey, user_or_group)

Description:

Gets the time spent (logged) on an issue by a certain user or group.

Parameters:

Parameter name	Type	Required	Description
issuekey	string	Yes	The key of the issue
user_or_group	string	Yes	A username or group name

Return type:

interval

Returns an interval that represents the sum of all the worklogs of the specified user or members of the specified group on the issue.

Example:

Example 1

```
getTimeSpent("PRJ-32", "jira-users");
```

Returns: The sum of the worklogs of all the members of "jira-users" on issue PRJ-32: 4d 5h

Example 2:

```
getTimeSpent("PRJ-32", "testuser");
```

Returns: The sum of the worklogs of the user "testuser" on issue PRJ-32: 2h

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

getTransitionNameById

> **Availability**
> This routine is available since **katl-commons 3.0.2 .**

Syntax:

getTransitionNameById(workflowName,transitionId)

Description:

Gets the transition name by the workflow name and transition id.

Parameters:

Parameter name	Type	Required	Description
workflow name	string	Yes	The workflow name of the transition
transition id	number	Yes	The transition id

Return type:

string

Returns the transition name for the workflow name and transition id provided.

Example:

```
return getTransitionNameById("ADMROUTINES WORKFLOW",3);
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getUser

> **Availability**
> This routine is available since **katl-commons 3.0.1**

Syntax:

getUser(user)

Description:

Returns the username

Parameters:

Parameter name	Type	Required	Description
user	String	Yes	The username or key

Return type:

JUser

Returns the user

Example:

Example 1:

```
JUser usr = getUser("john.doe");
```

Notes:

> The look-up is first made after the userkey, then after the username.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

getUserDirectoryName

> **Availability**
> This routine is available since **katl-commons 2.6.11/2.5.19 .**

Syntax:

getUserDirectoryName(user)

Description:

Returns the directory name to which the user belongs to.

Parameters:

Parameter name	Type	Required	Description
user	String	Yes	The user name or key.

Return type:

string

Returns the directory name

Example:

Example 1:

```
if(getUserDirectoryName(currentUser()) == "JIRA Internal Directory") {

 doSomethingRelatedToInternalDirectoryUsers();
}
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

getUserProperty

> **Availability**
> This routine is available since **katl-commons 2.0.7** (for JIRA 5.x) or **katl-commons 1.1.14** (for JIRA 4.3.x and 4.4.x)

Syntax:

getUserProperty(user, property)

Description:

Retrieves properties of users.

Parameters:

Parameter name	Type	Required	Description
user	String	Yes	The username or userkey to get the property for
property	String	yes	The key of the property.

Return type:

string

Returns the value of the specified property. If the user does not have the property, it will return an empty string.

Example:

```
return getUserProperty("testuser", "phone");
```

Notes:

> The look-up is first made after the userkey, then after the username.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

getWebLinkById

> **Availability**
> This routine is available since **katl-commons 2.5.15** (JIRA 5.x) **/ 2.6.7** (JIRA 6).

Syntax:

getWebLinkById(webLinkId)

Description:

Retrieves information about a web link.

Parameters:

Parameter name	Type	Required	Description
webLinkId	String	Yes	the id of a web link

Return type:

string []

Array containing information about a web link, in the following order:

Index in array	Information
0	Link Title
1	URL
2	Summary
3	Relationship
4	Icon URL
5	Icon Title
6	Status Icon Title
7	Status Icon Link
8	Status Icon URL

Example:

Example 1:

```
number [] ids = getWebLinksForIssue(key);

for(number wlid in ids){
 string [] props = getWebLinkById(wlid);
 if(props[0] == theOtherKey) {
  deleteWebLinkById(wlid);
 }
}
```

See Also:

---

getWebLinksForIssue

> **Availability**
> This routine is available since **katl-commons 2.5.15** (JIRA 5.x) **/ 2.6.7** (JIRA 6).

Syntax:

getWebLinksForIssue(issueKey)

Description:

Gets the IDs of all the web links on an issue

Parameters:

Parameter name	Type	Required	Description
issue key	String	Yes	the key of the selected issue

Return type:

number []

Array containing the IDs of all web links on an issue

Example:

Example 1:

```
number [] ids = getWebLinksForIssue(key);

for(number wlid in ids){
 string [] props = getWebLinkById(wlid);
 if(props[0] == theOtherKey) {
  deleteWebLinkById(wlid);
 }
}
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

getWorkflowStatusIds

> **Availability**
> This routine is available since **katl-commons 3.1.9.**

Syntax:

getWorkflowStatusIds(workflowName [, isDraft])

Description:

Retrieves an unique list of statuses (ids) for a given workflow

Parameters:

Parameter name	Type	Required	Description
workflow name	String	Yes	the name of the given workflow
isDraft	Boolean	No	flag for specifying whether we should consider the draft or not. If not specified, it defaults to false.

Return type:

string []

The return value is an array of strings, containing the ids for the existing statuses in the workflow.

Example:

```
string key = "TEST-19";
return getWorkflowStatusIds(key.workflow);
```

Result: 1|3|4|5|6

Example 2:

```
string key = "TEST-19";
return getWorkflowStatusIds(key.workflow, false);
```

Result: 1|3|4|5|6

Example 3:

Setting the second parameter to true, if we do have a draft in which there is another status added, the result will contain this status also.

```
string key = "TEST-19";
return getWorkflowStatusIds(key.workflow, true);
```

Result: 1|3|4|5|6|10100

groupExists

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

groupExists(grp)

Description:

Verifies if the selected group is a registered JIRA group.

Parameters:

Parameter name	Type	Required	Description
group name	string	Yes	The name of the group that will be verified

Return type:

boolean (true/false)

Returns **true** if the specified group name denotes a registered JIRA group, or **false** if the group name is unknown.

Example:

Example 1:

```
groupExists("Administrators");
```

Result: **True** if **Administrators** is a registered JIRA group. **False** if **Administrators** is not a registered JIRA group.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

hasInput

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

hasInput(customfield_id)
or
hasInput(keyword)

> Make sure that a transition screen has been attached to the current transition.

Description:

Verifies if a field **had input in the transition screen**. Should **only** be used in **transitions** that require a screen.

> This routine has a very limited working scope. Since it handles transient values, it will function correctly only if called in an interval starting from the moment the user clicks the submit button on the transition screen, up to the point when the issue is saved in the database. Consequently, we can define the valid scope for the routine to be **validators** and **some post-functions.** This means that **you will not be able to use it in conditions, SIL Services , SIL Listeners nor from the SIL Gadget.**
>
> **Usage in post-functions:** This routine will work in SIL Post-Functions that run BEFORE the ones that save the issue in the database. The latter are default JIRA post-functions which can be easily recognized by their name; e.g. "Update change history for an issue and store the issue in the database." for a regular transition, "Creates the issue originally." for the "Create Issue" transition, etc.

Parameters:

Parameter name	Type	Required	Description
customfield_id/keyword	string	Yes	The id of the custom field (customfield_id) or the specific keyword for the standard fields

Return type:

boolean (true/false)

A **true** return value means that the specified field had input in the transition screen.

Example:

Example 1:

```
hasInput("customfield_12003");
```

Returns: **True** if the **customfield_12003** contains a value, **False** if the **customfield_12003** is empty.

Example 2:

```
hasInput("comment");
```

True if the **Comment** field contains a value, **False** if the **Comment** field is empty.

Notes

> custom fields must be referenced by id (customfield_id), while standard fields can be specified by one of the following keywords: **summary**, **timetracking**, **security**, **attachment**, **reporter**, **environment**, **description**, **duedate**. You can also check the Comment field, using the keyword **comment**.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

hasPermission

> **Availability**
> This routine is available since **katl-commons 2.0.6** (for JIRA 5.x) or **katl-commons 1.1.13** (for JIRA 4.3.x and 4.4.x)

Syntax:

hasPermission(permissionId, user[, issueKey])

Description:

Checks if a user has the specified permission

Parameters:

Parameter name	Type	Required	Description
permissionId	Number	Yes	The permission ID.
user	String	Yes	The username or userkey to check the permission for
issueKey	String	No	The issue key. Only required for non-global permissions.

Permissions

Category	Permissions	Description	ID
Global Permissions	**JIRA Administrators**	Ability to perform most administration functions (excluding Import & Export, SMTP Configuration, etc.).	0
	Bulk Change	Ability to modify a collection of issues at once. For example, resolve multiple issues in one step.	33
	Create Shared Objects	Ability to share dashboards and filters with other users, groups and roles.	22
	Manage Group Filter Subscriptions	Ability to manage (create and delete) group filter subscriptions.	24
	JIRA System Administrators	Ability to perform all administration functions. There must be at least one group with this permission.	44
	JIRA Users	Ability to log in to JIRA. They are a 'user'. Any new users created will automatically join these groups, unless those groups have JIRA System Administrators or JIRA Administrators permissions.	1
	Browse Users	Ability to select a user or group from a popup window as well as the ability to use the 'share' issues feature. Users with this permission will also be able to see names of all users and groups in the system.	27
Project Permissions	**Administer Projects**	Ability to administer a project in JIRA.	23
	Browse Projects	Ability to browse projects and the issues within them.	10
	View Version Control	Ability to view Version Control commit information for issues.	29
	View Read-Only Workflow	Users with this permission may view a read-only version of a workflow.	45
	Assign Issues	Ability to assign issues to other people.	13
	Assignable User	Users with this permission may be assigned to issues.	17
	Close Issues	Ability to close issues. Often useful where your developers resolve issues, and a QA department closes them.	18
	Create Issues	Ability to create issues.	11
	Delete Issues	Ability to delete issues.	16
	Edit Issues	Ability to edit issues.	12
	Link Issues	Ability to link issues together and create linked issues. Only useful if issue linking is turned on.	21
	Modify Reporter	Ability to modify the reporter when creating or editing an issue.	30
	Move Issues	Ability to move issues between projects or between workflows of the same project (if applicable). Note the user can only move issues to a project he or she has the create permission for.	25
	Resolve Issues	Ability to resolve and reopen issues. This includes the ability to set a fix version.	14
	Schedule Issues	Ability to set or edit an issue's due date.	28
	Set Issue Security	Ability to set the level of security on an issue so that only people in that security level can see the issue.	26

Manage Watchers	Ability to manage the watchers of an issue.	32
View Voters and Watchers	Ability to view the voters and watchers of an issue	31
Delete All Comments	Ability to delete all comments made on issues.	36
Delete Own Comments	Ability to delete own comments made on issues.	37
Edit All Comments	Ability to edit all comments made on issues.	34
Edit Own Comments	Ability to edit own comments made on issues.	35
Add Comments	Ability to comment on issues.	15
Delete All Attachments	Users with this permission may delete all attachments.	38
Delete Own Attachments	Users with this permission may delete own attachments.	39
Create Attachments	Users with this permission may create attachments.	19
Work On Issues	Ability to log work done against an issue. Only useful if Time Tracking is turned on.	20
Delete All Worklogs	Ability to delete all worklogs made on issues.	43
Delete Own Worklogs	Ability to delete own worklogs made on issues.	42
Edit All Worklogs	Ability to edit all worklogs made on issues.	41
Edit Own Worklogs	Ability to edit own worklogs made on issues.	40

Return type:

boolean

Returns true if the user has the specified permission, or false otherwise.

> **Note**
> This routine also verifies the issue security level. If the specified user cannot see the issue because his security level does not allow it, the routine will return false for the project permissions even if the user does have the specified privilege. This happens because the issue security comes first and will prohibit the user from taking actions that are otherwise allowed by the permission scheme.
>
> The look-up is first made after the userkey, then after the username.

Example:

Example 1:

```
for(string user in usersInGroups({"QA-Testers"})){
 if(hasPermission(17, user, key)){
  assignee = user;
  return;
 }
}
```

Result: *The issue will be assigned to one of the testers who have the permission to be assigned issues on the current project.*

Example 2:

> **Tip**
> Attached to this page is permissions.incl which contains variable definitions for all the above permissions. You can upload this file to your JIRA instance via the SIL Manager, include it in your scripts and then use the variables defined there to make the code easier to understand.

Using permissions.incl, the above example becomes:

```
include "permissions.incl"; // assuming the file is in the default programs
folder
for(string user in usersInGroups({"QA-Testers"})){
 if(hasPermission(ASSIGNABLE_USER, user, key)){
  assignee = user;
  return;
 }
}
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

hasUserProperty

> **Availability**
> This routine is available since **katl-commons 3.0.0**

Syntax:

hasUserProperty(user, property)

Description:

Checks if the user has the given property set.

Parameters:

Parameter name	Type	Required	Description
user	String	Yes	The username or userkey to check the property for
property	String	yes	The key of the property to check

Return type:

boolean

Returns true if the user has the given property set, false otherwise.

Example:

```
if(hasUserProperty("testuser", "phone")) {
 return getUserProperty("testuser", "phone");
}
```

Notes:

> The look-up is first made after the userkey, then after the username.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

isAnyUserAuthenticated

> **Availability**
> This routine is available since **katl-commons 2.0.5** (for JIRA 5.x) or **katl-commons 1.1.12** (for JIRA 4.3.x and 4.4.x)

Syntax:

isAnyUserAuthenticated()

Description:

Verifies if there is a logged in user.

Parameters:

None

Return type:

boolean (true/false)

A **true** return value means that there is a logged in user.

Example:

Example 1:

```
if (isAnyUserAuthenticated()) {
     print(currentDate());
 }
```

Result: It will be printed the current date. If there is nobody logged in, the current date will not be printed.

2012-06-20 13:39:31,629 pool-5-thread-2 INFO admin 819x261x1 1k7wpbj 127.0.0.1 /rest/keplerrominfo/jjupin/latest/rungadget/run [commons.sil.routines.StringPrintRoutine] 2012-06-20 13:39:31

See Also:

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

isCustomFieldInContext

Availability
This routine is available since **katl-commons 2.5.10 / 2.6.2 .**

Syntax:

isCustomFieldInContext(customFieldAsString, projectKey, issueTypeName)

Parameters:

Parameter name	Type	Required	Description
customFieldAsString	String	Yes	the name, id (as "customfield_xxxxx") or alias of the custom field
projectKey	String	Yes	the project key to check
issuetypeName	String	Yes	the name of the issue type

Description:

Checks if a custom field is in the context of a specified issue type for a project.

Return type:

boolean

True if the custom field is in context, false otherwise

Example:

Example 1:

```
if(isCustomFieldInContext("My Custom Field", "PRJ", "Bug")){
 #{My Custom Field} = "I'm here";
}

if(isCustomFieldInContext("customfield_10000", "PRJ", "Improvement")){
 customfield_10000 = "I'm here";
}

if(isCustomFieldInContext("aliasOfMyCustomField", "PRJ", "Bug")){
 aliasOfMyCustomField = "I'm here";
}
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

isIssueContext

Availability
This routine is available since **katl-commons 3.0 .**

Syntax:

isIssueContext()

Description:

Verifies if the script is running in an issue context

Parameters:

none

Return type:

boolean (true/false)

A **true** return value means that the script is running is an issue context and that the standard variables and custom fields are available for use.

Example:

Example 1:

```
if(isIssueContext()){
    return usersInGroups({"jira-users"}) - reporter;
} else {
    return usersInGroups({"jira-users"}) - currentUser();
}
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

issueStateFlush

> **Availability**
> This routine is available since **katl-commons 2.0.7** (for JIRA 5.x) or **katl-commons 1.1.14** (for JIRA 4.3.x and 4.4.x)

Syntax:

issueStateFlush(issueKey)

Description:

Removes all the saved states for a given issue.

Since the states of an issue are saved in the database and can add up to quite a bit of disk space, it is a good idea to flush the states that are not needed anymore.

Parameters:

Parameter name	Type	Required	Description
issueKey	String	Yes	the key of the selected issue

Return type:

none

The returned value has no meaning

Example:

Example 1:

```
issueStatePush(key, {"assignee", "customfield_10000"});
issueStatePush(key);
issueStateFlush(key);
print(issueStatePop(key));
```

Does not print anything since there will be no more states to pop after issueStateFlush()

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

issueStatePop

> **Availability**
> This routine is available since **katl-commons 2.0.7** (for JIRA 5.x) or **katl-commons 1.1.14** (for JIRA 4.3.x and 4.4.x) .

Syntax:

issueStatePop(issueKey [, fields])

Description:

Removes and retrieves the top issue state from the stack (the latest saved state).

If the fields parameter is not specified, it will retrieve all the fields saved in the state.

If the fields parameter is specified, it will only retrieve the specified fields (if available). The specified custom field names must match the ones that were saved. If there was a *issueStatePush(key, {"My Custom Field"})* then the pop must also use "My Custom Field". Other references to it, like *customfield_10000* or an alias will not work. In other words, you can use any naming method for push (by custom field name, by id using customfield_xxxxx or by its alias), but you must use the same one for pop.

> **Note**
> Note that this removes the top state. If you specify the fields to retrieve, the values stored for all the other fields will be lost.
>
> Also, it does **NOT set the popped values on the issue.** It will only return them. To also set the popped values, use **issueStatePopAndSet**

Parameters:

Parameter name	Type	Required	Description
issueKey	String	Yes	the key of the selected issue
fields	String []	No	the fields to save. These are string values representing the **names of the fields**.

Return type:

string []

Returns the values saved in the state in pairs of field_name and field_value.

Example:

Example 1:

```
issueStatePush(key, {"assignee", "customfield_10000"});
print(issueStatePop(key));
```

Prints: *assignee|<assignee's username>|customfield_10000|<value of customfield_10000>*

Example 2:

```
issueStatePush(key, {"assignee", "customfield_10000"});
print(issueStatePop(key, {"assignee"}));
```

Prints: *assignee|<assignee's username>*

> **Note**
> Note that the field names are given as strings. Using **assignee** instead of **"assignee"** will attempt to retrieve a field which has the name of the assignee. Since the field will probably not exist, it will throw an exception and the script will result in an error.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

issueStatePopAndSet

> **Availability**
> This routine is available since **katl-commons 2.0.7** (for JIRA 5.x) or **katl-commons 1.1.14** (for JIRA 4.3.x and 4.4.x)

Syntax:

issueStatePopAndSet(issueKey [, fields])

Description:

Removes, retrieves and sets the top issue state from the stack (the latest saved state).

If the fields parameter is not specified, it will retrieve and set all the fields saved in the state.

If the fields parameter is specified, it will only retrieve and set the specified fields (if available). The specified custom field names must match the ones that were saved. If there was a *issueStatePush(key, {"My Custom Field"})* then the pop must also use "My Custom Field". Other references to it, like *customfield_10000* or an alias will not work. In other words, you can use any naming method for push (by custom field name, by id using customfield_xxxxx or by its alias), but you must use the same one for pop.

> **Note**
> Note that this removes the top state. If you specify the fields to retrieve and set, the values stored for all the other fields will be lost.
>
> Also note that this routine **SETS the popped values on the issue.** To only retrieve the popped values, use issueStatePop

Parameters:

Parameter name	Type	Required	Description
issueKey	String	Yes	the key of the selected issue
fields	String []	No	the fields to save. These are string values representing the **names of the fields**.

Return type:

string []

Returns the values saved in the state in pairs of field_name and field_value.

Example:

Example 1:

```
issueStatePush(key, {"assignee", "customfield_10000"});
print(issueStatePop(key));
```

Prints: *assignee|<assignee's username>|customfield_10000|<value of customfield_10000>*

This will also set the assignee and customfield_10000 to the popped values.

Example 2:

```
issueStatePush(key, {"assignee", "customfield_10000"});
print(issueStatePop(key, {"assignee"}));
```

Prints: *assignee|<assignee's username>*

This will also set the assignee to the popped value.

> **Note**
> Note that the field names are given as strings. Using **assignee** instead of **"assignee"** will attempt to retrieve a field which has the name of the assignee. Since the field will probably not exist, it will throw an exception and the script will result in an error.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

issueStatePush

> **Availability**
> This routine is available since **katl-commons 2.0.7** (for JIRA 5.x) or **katl-commons 1.1.14** (for JIRA 4.3.x and 4.4.x)

Syntax:

issueStatePush(issueKey [, fields])

Description:

Saves fields of an issue (standard and custom fields) into a stack. You can later pop these states from the stack to set the saved values back on the issue.

If the fields parameter is not specified, it will save all of the non-calculated custom field values and the following standard values: **assignee, created, description, dueDate, environment, estimate, originalEstimate, priorityId, priority, resolutionDate, securityLevel, securityLevelId, summary, timeSpent, votes, issueTypeId, issueType, reporter, resolution, resolutionId, components, affectedVersions, fixVersions, watchers, labels.**

If the fields parameter is specified, it will only save the specified fields.

> **Tip**
> You can use any **valid** form of referencing a custom field in the **fields** parameter. For example, we have a custom field named *Reference* with id 10000 and an alias of *ExtReference*. Therefore "Reference", "customfield_10000" or "ExtReference" are all **valid** values to be used in the **fields** parameter.

Parameters:

Parameter name	Type	Required	Description

issue key	String	Yes	the key of the selected issue
fields	String []	No	the fields to save. These are string values representing the **names of the fields**.

Return type:

none

The returned value does not have any meaning.

Example:

Example 1:

```
issueStatePush(key);
```

Creates a state with all of the current issue's values (standard fields and custom fields) and pushes it at the top of the stack.

Example 2:

```
issueStatePush(key, {"assignee", "customfield_10000"});
```

Creates a state with only two values: the assignee and the value of a custom field. On pop, the other values will not be available for restoration.

> **Note**
> Note that the field names are given as strings. Using **assignee** instead of **"assignee"** will attempt to save a field which has the name of the assignee. Since the field will probably not exist, it will throw an exception and the script will result in an error.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

issueTypesForProject

Syntax:

issueTypesForProject(projectKey)

> **Availability**
> This routine is available since **katl-commons 3.1.2/katl-commons 3.0.14.**

Description:

Retrieves the issue types for the project with the given key.

Parameters:

Parameter name	Type	Required	Description
projectKey	String	Yes	the project key

Return type:

string[]

The return value is a string array containing all the issue types available for the given project.

Example:

```
return issueTypesForProject("TST");
```

Result: Improvement|Task|Sub-task|New Feature|Bug|Epic|Story|Symptom

isTeamLeader

> **Availability**
> This routine is available since **katl-commons 1.1 .**

Syntax:

isTeamLeader(project, username)

Description:

Verifies if the specified user is a **team leader** on the project (if it is a **component lead**).

Parameters:

Parameter name	Type	Required	Description
project key	String	Yes	the key of the selected project
user name	String	Yes	the user name of user that is verified

Return type:

boolean (true/false)

A **true** return value means that the specified user is **component lead** on at least one component of the specified project.

Example:

Example 1:

```
//Team leaders of the current project are : TM1 and TM2
isTeamLeader(project, currentUser());
```

Result: **True** if the current user is TM1 or TM2, **False** if the current user is other than TM1 or TM2.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

isUserInRole

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

isUserInRole(user,project, role)

Description:

Returns true if the user has a certain role on the specified project.

This routine exists (as a convenience) starting from JJupin 2.0.6. Prior to that, one would have to get the roles of the user on the project, then check if the desired role is in the returned list

Parameters:

Parameter name	Type	Required	Description
User	String	Yes	The username or userkey of the user in question
Project key	String	Yes	The key of the selected project
Role name	String	Yes	The name of the role that is verified

Return type:

bool

True if the user has the role on a project, false otherwise

Example:

Example 1:

```
return isUserInRole("mike", "PRJ", "Developers");
```

Notes:

> The look-up is first made after the userkey, then after the username.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

lastFieldHistory

> **Availability**
> This routine is available since **katl-commons 2.5.13 / 2.6.5 .**

Syntax:

lastFieldHistory(issueKey)

Description:

Returns the last change details (**user | date | field | oldValue | newValue**) from the selected issue's history.

Parameters:

Parameter name	Type	Required	Description
issue key	String	Yes	the key of the selected issue

Return type:

string []

The return value is an array of strings, containing the following values in the specified order: the user who made the change, the date, the name of the field that has been modified, the old field value and the new field value.

Example:

```
string[] lastChange = lastFieldHistory(key);
string ret = "Issue " + key + " was last changed on " + lastChange[1] + "
by " + userFullName(lastChange[0]);
ret += ": Field " + lastChange[2] + " from >>" + lastChange[3] + "<< to >>"
+ lastChange[4] + "<<";
return ret;
```

Result: Issue DEMO-5 was last changed on 2013-08-20 16:47:57 by Admin User: Field assignee from >>Admin User<< to >>Test User<<

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

linkedIssues

> **Availability**
> This routine is available since **katl-commons 2.5** (for JIRA 5.x) .

Syntax:

linkedIssues(issueKey, [[linkTypeName], [direction]]);

Alias:

getLinkedIssue (issueKey, [[linkTypeName], [direction]])

Description:

Returns an array with the Issue names linked with the specified one.

This routine searches for all the **linked issues** with the one given as argument (parameter) and builds an array that contains them. The second parameter **[linkTypeName]** is optional and represents a searching filter that select only those issues that have a certain relation (link type) with the argument issue.

The list of **Issue types** that can be given as arguments is:

1) Blocks
2) Relates
3) Clones

But also other **Issue links** can be defined. To add new link type follow the tutorial from https://confluence.atlassian.com/display/JIRA/Configuring+Issue+Linking.

It can be used to find connections between different issues.

The third parameter [**direction**] is optional and it is used to specify whether to retrieve inward links or outward links.

> The [direction] parameter is available starting from version 2.5.8 of katl-commons.

Parameters:

Parameter name	Type	Required	Description
issueKey	string	Yes	Specifies a string representing the key of the issue.
linkTypeName	string	No	Specifies the link type.
direction	number	No	-1, 0, 1 :Negative means inward links. Positive means outward links. Use zero to get both outward and inward links

Return type:

an array of strings

The return value represents a list with all issues (issue keys) that are linked with the argument.

Example:

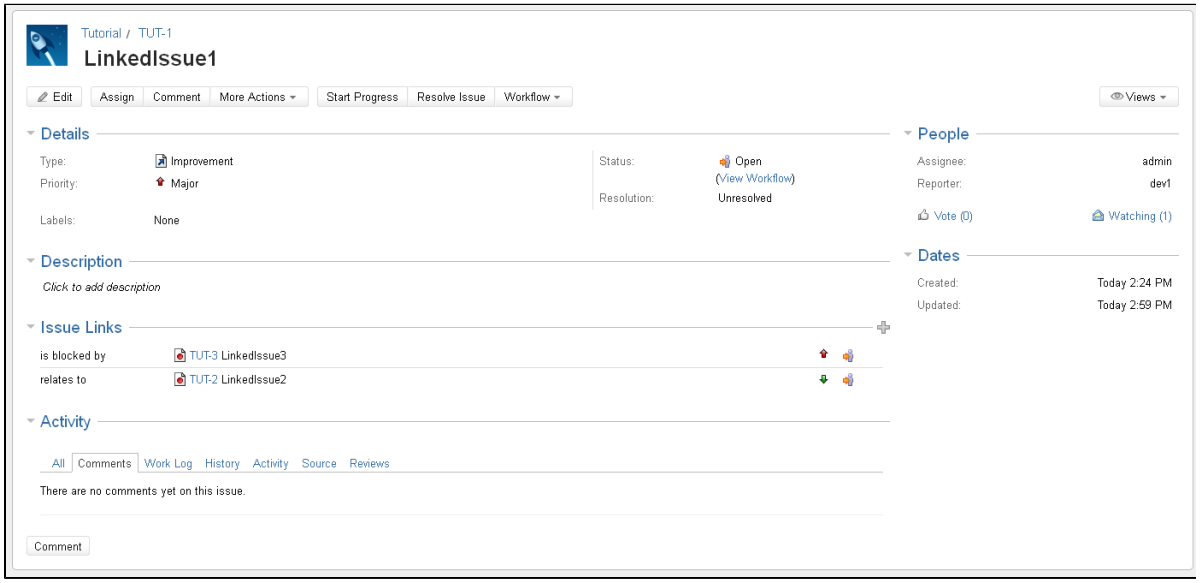
A good way to understand the idea of this routine is by using an example, so:

Let's consider the following three issues: "**TUT-1**", "**TUT-2**" and "**TUT-3**".

The issue "**TUT-1**" has the following links:

1) Is blocked by "**TUT-3 LinkedIssue3**"

2) Relates to "**TUT-2 LinkedIssue2"**



If we run the following SIL code:

```
return linkedIssues("TUT-1","Blocks");
```

The result will be an array with one element: "**TUT-3**", because **TUT-3** blocks the resolving of the "**TUT-1**" issue.

For the above example, the same result you will get by running the following code:

```
return linkedIssues("TUT-1", "Blocks", -1);
```

, as "is blocked by" represents the inward description for the "Blocks" link type.

But if we run the following code:

```
return linkedIssues("TUT-1");
```

The result will be an array with two elements: "**TUT-3|TUT-2**", because both issues are linked with "**TUT-1**".

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

linkIssue

> **Availability**
> This routine is available since **katl-commons 1.1.2 .**

Syntax

linkIssue(issueKey1, issueKey2, issueLinkTypeName)

Description:

Links two issues by a specified link type name. First issue will have the outward description of the link type, the second issue will have the inward description of the link type.

Parameters:

Parameter name	Type	Required	Description
issueKey1	string	Yes	The key of the first issue to be linked
issueKey2	string	Yes	The key of the second issue to be linked
issueLinkTypeName	string	Yes	The issue link type name

Return type:

None. The returned value has no meaning.

Example:

Example 1:

```
linkIssue(key, "TST-123", "Relates");
//key represents the key of current issue, "TST-123" the key of the issue
it will be linked to.
```

Result: current issue is linked to issue "TST-123" by **Relates** link type, such that "TST-123" **relates to** current issue and vice versa.

Example 2:

```
linkIssue("TST-123", "TST-234", "Blocks");
```

Result: "TST-123" issue is linked to issue "TST-234" by **Blocks** link type, such that "TST-123" **blocks** "TST-234" and "TST-234" **is blocked by** "TST-123".

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

projectMembers

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

projectMembers(project)

Description:

Returns a list with all the user names of the users who have a role in the specified project.

Parameters:

Parameter name	Type	Required	Description
project key	String	Yes	The key of the project that is selected

Return type:

string []

The return value is a list of **usernames** for the users who have a **role** on the specified project.

Example:

```
//The current project key = PRJ1
//The users from the current project: Admin, Dev1, Dev2, Dev3, Test1,
Test2, PM, BA1
print("The members of the project " + project + " are:");
print(projectMembers(project));
```

Result: Returns the following text: *The members of the project PRJ1 are: Admin, Dev1, Dev2, Dev3, Test1, Test2, PM, BA1.* (Check the values on the next row beginning with <StringPrintRoutine>.)

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

projectPM

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

projectPM(project)

Description:

Returns the username of the **project manager** (**project lead**) of the selected project, if exists.

Parameters:

Parameter name	Type	Required	Description
Project key	String	Yes	the key of the selected project

Return type:

string

The returned string represents the **username** of the **project lead**.

Example:

Example 1:

```
//Project key of the current project is PRJ
//User name of the project manager is JohnSmith
string PM;
PM = projectPM(project);
print("The project manager of " + project + " project is " + PM);
```

Result: *The project manager of PRJ project is JohnSmith*

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

projectsForPM

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

projectsForPM(user)

Description:

Returns all the projects that the selected user has the role of **project manager** (**project lead**).

Parameters:

Parameter name	Type	Required	Description
user name	String	Yes	The user name of the user that is verified

Return type:

string []

Returns a list of **project keys** for which the specified user is a **project lead**.

Example:

Example 1:

```
//user "Admin" has the role of project manager in the following projects:
PRJ3, PRJ5, PRJ6.
string projects = projectsForPM("Admin");
print("You have the role of project manager on the following projects: ");
print(projects);
```

Results: The following string is returned: *You have the role of project manager on the following projects: PRJ3, PRJ5, PRJ6* (Check the values on the next row beginning with <StringPrintRoutine>.)

Example 2:

```
projectsForPM(currentUser());
```

Results: Returns an array containing the keys of all the projects in which the user has the role of project manager.

Notes:

1. If there are no results to the invoking of the function the return value will be an empty array.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

projectsForUser

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

projectsForUser(user, roles)

Description:

Returns all the projects for which this user has permission to **assign** or to **be assigned** issues.

Parameters:

Parameter name	Type	Required	Description
user	String	Yes	The username or userkey of the user that is verified
roles	String[]	No	The roles the user must have

Return type:

string []

Returns a list of **project keys** on which the user denoted by the specified **username or userkey** has permission to **assign** or **to be assigned** iss ues.

If the second parameter is present, it returns a list of project keys on which the user is in one of the specified role.

Example:

Example 1:

```
//user "Admin" has permission to assign issues or has the permission to be
assigned in the following projects: PRJ1, PRJ2, PRJ8.
print("You are allowed to work on the following projects: ");
print(projectsForUser("Admin"));
```

Results: The following string is returned: *You are allowed to work on the following projects: PRJ1, PRJ2, PRJ8* (Check the values on the next row beginning with <StringPrintRoutine>.)

Example 2:

```
projectsForUser(currentUser());
```

Results: Returns an array containing the keys of all the projects in which the user has permission to assign issues or has the permission to be assigned.

Example 3:

```
return projectsForUser("admin", {"Administrators", "Developers"} );
```

Results: Returns an array containing the keys of all the projects in which the user is in role of Administrator or Developer.

Notes:

1. The look-up is first made after the userkey, then after the username.
2. If there are no results to the invoking of the function the return will be an empty array.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

projectsWithPermissionForUser

Syntax:

projectsWithPermissionForUser(user, permissionKey)

Availability
This routine is available since **katl-commons 3.1.2/katl-commons 3.0.14.**

Description:

Retrieves the project keys on which the given user has the given permission.

Parameters:

Parameter name	Type	Required	Description
user	String	Yes	the username
permissionKey	String	Yes	the key of the persmission

Return type:

string[]

The return value is a string array containing all the project keys on which the given user has the given permission.

Example:

```
return projectsWithPermissionForUser("admin", "ADMINISTER_PROJECTS"));
```

Result: DEMO|ITSD|TST

raiseEvent

> **Availability**
> This routine is available since **katl-commons 1.1 .**

Syntax:

raiseEvent(event_name, issue, user)

Description:

Triggers an event to be processed by listeners.

Parameters:

Parameter name	Type	Required	Description
event name	String	Yes	The name of the event to be triggered, as it appears in the ***Administration->Events*** section.
issue key	String	Yes	The key of the issue that will raise the event
user name	String	Yes	The user name of the user who will generate the event

Return type:

boolean (true/false)

A **true** return value means that the specified event was triggered sucessfully. A **false** value means that there was a problem and you should investigate this further by consulting the logs.

Example:

Example 1:

```
//Assuming we have already added the "Delete" event
raiseEvent("Delete", "PRJ-231", currentUser());
```

Result: **true** if the event was triggered for the issue PRJ-231 by the current user, **false** if the event wasn't triggered.

Example 2:

```
//Assuming we have already added the "Delete" event
raiseEvent("Delete", key, "Admin");
```

Result: **true** if the event was triggered for the current issue by the user "Admin", **false** if the event wasn't triggered.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

removeGroupFromProjectRole

> **Availability**
> This routine is available since **katl-commons 3.0.8 .**

Syntax:

removeGroupFromProjectRole(group, project, role)

Description:

Removes a single group from a project role if the group is in that role.

Parameters:

Parameter name	Type	Required	Description
Group	String	Yes	The group name
Project key	String	Yes	The key of the selected project
Role name	String	Yes	The name of the role to add the user to

Return type:

boolean

true if operation succeeded

Example:

Example :

```
removeGroupFromProjectRole("dev-group, "TEST", "Developers");
```

See Also:

---

removeUserFromGroup

> **Availability**
> This routine is available since **katl-commons 2.5.16** (JIRA 5.x) **/ 2.6.8** (JIRA 6).

Syntax:

removeUserFromGroup(user, group)

Description:

Removes a single user from a group if the user is in that group.

Parameters:

Parameter name	Type	Required	Description
User	String	Yes	The username or userkey
Group	String	Yes	The name of the group

Return type:

boolean

true if operation succeeded

Example:

Example 1:

```
removeUserFromGroup("user.3", "Senior Developers");
```

Notes:

> The look-up is first made after the userkey, then after the username.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

removeUserFromProjectRole

> **Availability**
> This routine is available since **katl-commons 2.5.16** (JIRA 5.x) **/ 2.6.8** (JIRA 6).

Syntax:

removeUserFromProjectRole(user, project, role)

Description:

Removes a single user from a project role if the user is in that role.

Parameters:

Parameter name	Type	Required	Description
User	String	Yes	The username or userkey
Project key	String	Yes	The key of the selected project
Role name	String	Yes	The name of the role to add the user to

Return type:

boolean

true if operation succeeded

Example:

Example 1:

```
removeUserFromProjectRole("user.3", "TEST", "Developers");
```

Notes:

> The look-up is first made after the userkey, then after the username.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

renderWiki

Syntax:

renderWiki(wiki)

Description:

Returns the html code for the wiki text.

Return type:

string

The return value represents the html code of the wiki text.

Example:

Example 1:

```
return renderWiki("h1. Biggest heading");
```

Result: <h1>Biggest heading</h1>

runAs

> **Availability**
> This routine is available since **katl-commons 2.0.5** (for JIRA 5.x) or **katl-commons 1.1.12** (for JIRA 4.3.x and 4.4.x) .

Syntax:

runAs(user)

Description:

Assumes a user when running a script.

Parameters:

Parameter name	Type	Required	Description
User	String	Yes	The username of userkey of the selected user

Return type:

None

Example:

Example 1:

```
runAs("user1");
 print(currentDate());
 print(currentUser());
 runAs("admin");
```

In the console the output will be like:

2012-06-20 13:39:31,629 pool-5-thread-2 **INFO user1** 819x261x1 1k7wpbj 127.0.0.1 /rest/keplerrominfo/jjupin/latest/rungadget/run [commons.sil.routines.StringPrintRoutine] 2012-06-20 13:39:31

2012-06-20 13:39:31,629 pool-5-thread-2 **INFO user1** 819x261x1 1k7wpbj 127.0.0.1 /rest/keplerrominfo/jjupin/latest/rungadget/run [commons.sil.routines.StringPrintRoutine] user1

Notes:

The look-up is first made after the userkey, then after the username.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

selectIssues

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

selectIssues(jql)

Description:

Returns an array with the keys of the issues that matched the search query.

Parameters:

Parameter name	Type	Required	Description
search query	string	Yes	search query containing the details of the returned issues.

Return type:

string []

Returns a list of **issue keys** that match the specified jql.

Example:

Example 1:

```
string jql;
jql = "project = PRJ AND reporter in membersOf('Employees') AND status in
(Open, 'In Progress', Reopened, Resolved, 'On hold', Assigned, 'Internal
QA', 'Results rejected', 'Tested and not delivered', 'Tested and
delivred')";
selectIssues(jql);
```

Result: An array of strings containing the keys of all the issues that are in the project **PRJ**, with the reporter included in **Employees** group and the status being one of the above.

Notes:

For the search query, please use apostrophe(') instead of quotes(") and double backslashes(\ \ )for escaping characters (instead of a single backslash).

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

selectIssuesByFilter

> **Availability**
> This routine is available since **katl-commons 2.5.15 / 2.6.7**.

Syntax:

selectIssuesByFilter(filter name, owner user name)

Description:

Returns an array with the keys of the issues obtained by running the given filter.

Parameters:

Parameter name	Type	Required	Description
filter name	string	Yes	the name of the search filter to be executed
owner user name	string	No	the username for the owner of the filter. If parameter is missing, the owner is considered to be the current logged in user

Return Type:

string []

Returns a list of **issue keys** that match the query of the specified filter.

Example:

Example 1:

```
selectIssuesByFilter("filter1", "admin");
```

Result: An array of strings containing the keys of the issues that are obtained by running the filter named "filter1" that has the owner "admin".

Example 2:

```
selectIssuesByFilter("filter1");
```

Result: An array of strings containing the keys of the issues that are obtained by running the filter named "filter1" that has the owner the current logged in user.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

setUserProperty

> **Availability**
> This routine is available since **katl-commons 2.0.7** (for JIRA 5.x) or **katl-commons 1.1.14** (for JIRA 4.3.x and 4.4.x) .

Syntax:

setUserProperty(user, propertyKey, propertyValue)

Description:

Sets properties of users.

If the property does not exist, it will be created.

Parameters:

Parameter name	Type	Required	Description
user	String	Yes	The username or userkey to get the property for
propertyKey	String	Yes	The key of the property.
propertyValue	String	Yes	The value to set for the property.

Return type:

none

The returned value has no meaning.

Example:

```
setUserProperty("testuser", "phone", "987 654 3210");
```

Notes:

> The look-up is first made after the userkey, then after the username.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

subtasks

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

subtasks(issuekey)

Description:

Get the list of sub tasks linked to the parent issue.

Parameters:

Parameter name	Type	Required	Description
issuekey	string	Yes	The key of the issue

Return type:

string []

Returns a list of **issue keys** that are subtasks of the specified issue.

Example:

Example 1:

```
//Subtasks of the issue "PRJ-32" are : PRJ-192,PRJ-193,PRJ-203.
subtasks("PRJ-32");
```

Returns: The list of the sub tasks linked to **PRJ-32**: |PRJ-192|PRJ-193|PRJ-203|

Example 2:

```
subtasks(key);
```

Returns: The list of the sub tasks linked to the **current issue** in the form similar with the example above.

See Also:

---

unlinkIssue

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax

unlinkIssue(issueKey1, issueKey2, issueLinkTypeName)

Description:

Removes the specified link between two issues.

Parameters:

Parameter name	Type	Required	Description
issueKey1	string	Yes	The key of the first issue to be unlinked
issueKey2	string	Yes	The key of the second issue to be unlinked
issueLinkTypeName	string	Yes	The issue link type name

Return type:

None. The returned value has no meaning.

Example:

Example 1:

```
unlinkIssue(key, "TST-123", "Relates");
//key represents the key of current issue, "TST-123" the key of the issue
it will be linked to.
```

Result: The **Relates** link between current issue and issue "TST-123" will be removed.

Example 2:

```
linkIssue("TST-123", "TST-234", "Blocks");
```

Result: The **Blocks** link between issue "TST-123" and issue "TST-234" will be removed.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

userEmailAddress

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

userEmailAddress(user)

Description:

Returns the email address of the selected user. The email address may be needed to supply it to various external systems.

Parameters:

Parameter name	Type	Required	Description
user	String	Yes	The user key or name of the user for which the email should be provided.

Return type:

string

Returns the e-mail address associated with the specified userkey/username.

Example:

Example 1:

```
userEmailAddress("Admin");
```

Returns: If exists, returns the email address of the user **Admin**.

Example 2:

```
userEmailAddress(currentUser());
```

Returns: If exists, returns the email address of the current user.

Notes:

> The look-up is first made after the userkey, then after the username.
> If the email of the selected user doesn't exist the function returns an empty string.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

userExists

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

userExists(user)

Description:

Verifies if the selected user is registered JIRA user.

Parameters:

Parameter name	Type	Required	Description
user	string	Yes	Username or userkey of the verified user

Return type:

boolean (true/false)

A **true** return value means that there is a registered JIRA user associated with the specified **username or userkey**.

Example:

Example 1:

```
userExists("Administrator");
```

Returns: **True** if **Administrator** is the username of a registered JIRA user, or **False** otherwise.

Notes:

> The look-up is first made after the userkey, then after the username.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

userFullName

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

userFullName(user)

Description:

Returns the full name (firstname, lastname) of the user

Parameters:

Parameter name	Type	Required	Description
user	string	Yes	The username or userkey of selected user

Return type:

string

Returns the **full name** (first name and last name) of the user associated with the specified **username or userkey**.

Example:

Example 1:

```
userFullName("Admin");
```

Returns the full name (first name and last name) of the user **Admin**.

Example 2:

```
userFullName(currentUser());
```

Returns the full name (first name and last name) of the current user.

Notes:

> The look-up is first made after the userkey, then after the username.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

userGroups

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

userGroups(user)

Description:

Returns the groups in which the selected user belongs to.

Parameters:

Parameter name	Type	Required	Description
user	string	yes	Username or userkey of the selected user

Return:

string []

The returned array represent the names of the groups this user belongs to.

Example:

Example 1:

```
userGroups("username");
```

Result: The list of groups that the user with account **username** belongs to.

Example 2:

```
userGroups(currentUser());
```

Result: The list of groups that the current user belongs to.

Notes:

> The look-up is first made after the userkey, then after the username.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

userHasAccessToComment

Syntax:

userHasAccessToComment(user, comment_id)

Description:

Verifies if a comment is visible for an user.

Parameters:

Parameter name	Type	Required	Description
user	String	Yes	the username or userkey of the user
comment_id	Number	Yes	the id of the comment

> The look-up is first made after the userkey, then after the username.

Return type:

boolean (true/false)

A **true** return value means that the user can see the comment.

Example:

Example 1:

```
userHasAccessToComment("admin","10000");
```

Returns: **True** if the user "admin" can see the comment with id "10000", **False** if the **user** cannot see the comment.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

userInGroup

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

userInGroup(group, user)

Description:

Verifies if the selected user is in the selected group(s).

Parameters:

Parameter name	Type	Required	Description
group	string / string[]	Yes	The name(s) of the group(s) in which the user must be
user	string	Yes	The name or key of the user that needs to be verified

Return type:

boolean (true/false)

A **true** return value means that the user associated with the specified username or userkey belongs to the given group(s).

Example:

Example 1:

```
userInGroup("Administrators", "Admin1");
```

Returns: **True if Admin1** is included in **"Administrators" group or False** if **Admin1** is not included in **Administrators** group.

Example 2:

```
userInGroup("Users", currentUser());
```

Returns: **True** if the current user is included in **"Users"** group or **False** if the current user is not included in **"Users"** group.

Example 3:

```
string[] groups = {"Administrators", "Users"};
userInGroup(groups, currentUser());
```

Returns: **True** if the current user is included in **"Users" or "Administrators"** group, or **False** if the current user is not included neither in **"Users"** nor in **"Administrators"** group.

Notes:

> The look-up is first made after the userkey, then after the username.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

userKeyToUsername

> **Availability**
> This routine is available since **katl-commons 3.0.1**

Syntax:

userKeyToUsername(userKey)

Description:

Returns the username

Parameters:

Parameter name	Type	Required	Description
user key	String	Yes	The user key

Return type:

string

Returns the username

Example:

Example 1:

```
string username = userKeyToUsername("someUserKey");
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

userLanguage

> **Availability**
> This routine is available since **katl-commons 3.0.2 .**

Syntax:

userLanguage(user)

Description:

Returns the language for an user.

Parameters:

Parameter name	Type	Required	Description
user	string	Yes	Username or key of the user

Return type:

string

The return value represents the language for the specified user.

Example:

Example 1:

```
userLanguage("Administrator");
```

Returns: returns the language of user "Administrator".

Notes:

> The look-up is first made after the userkey, then after the username.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

usernameToUserKey

> **Availability**
> This routine is available since **katl-commons 3.0.1**

Syntax:

usernameToUserKey(username)

Description:

Returns the user key

Parameters:

Parameter name	Type	Required	Description
username	String	Yes	The user name

Return type:

string

Returns the user key

Example:

Example 1:

```
string userKey = usernameToUserKey("someUsername");
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

userRoles

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

userRoles(project, user)

Description:

Returns the roles of the provided user in the project.

Parameters:

Parameter name	Type	Required	Description
project key	String	Yes	The key of the selected project
user name	String	Yes	The user name of the user that is verified

Return type:

string []

Returns a list of roles the user associated with the given username has on the specified project.

Example:

Example 1:

```
//dev1 has the following roles in the project PRJ: developer, tester,
business analyst.
string user;
string[] roles;
user = "dev1";
roles = userRoles(project, user);
print ("The user " + user + "has the following roles in the project" +
project + ":");
print(roles);
```

Result: *The user dev1 has the following roles in the project PRJ: developer, tester, business analyst* (Check the values in log on the next row beginning with <StringPrintRoutine>.)

Example 2:

```
//current user has the following roles in the project PRJ: developer,
tester, business analyst.
string[] roles;
roles = userRoles(project, currentUser());
print ("The current user has the following roles in the project" + project
+ ":");
print(roles);
```

Result: *The current user has the following roles in the project PRJ: developer, tester, business analyst* (Check the values in log on the next row beginning with <StringPrintRoutine>.)

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

usersInGroups

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

usersInGroups(groups)

Description:

Returns a list of users **common to all** the **specified groups**.

Parameters:

Parameter name	Type	Required	Description
groups	string[]	Yes	The array containing the groups names from which to retrieve the common users.

Return type:

string []

Returns a list with the usernames of the common users for all the specified groups.

The result of the routine is **the intersection of the sets of users for each specified group**, not the union.

Example:

Example 1:

```
//The following users belong to both groups jira-developers and
jira-administrators: user1, user2
string[] groups = {"jira-developers", "jira-administrators"};
string[] usersByGroups;
usersByGroups = usersInGroups(groups);
print("The following users belong to both groups jira-administrators and
jira-developers: ");
print(usersByGroups);
```

Result:*The following users belong to both groups jira-administrators and jira-developers:*

user1|user2

Example 2: Union of groups

```
function getUsers(string [] groups){
  string [] users;
  for(string group in groups){
    string [] currentGrp;
    currentGrp = addElement(currentGrp, group);
    for(string user in usersInGroups(currentGrp)){
      users = addElementIfNotExist(users, user);
    }
  }
  return users;
}


string [] groups = {"jira-developers", "jira-administrators"};
description = getUsers(groups);
```

This example uses the usersInGroups routine to get all users in each group individually and then adding them into a predefined array. So the usersInGroups routine will be called twice: once for jira-administrators and once for jira-developers. Since each time it will be called with a single group, it will return all users from the specified group.

Since **katl-commons 2.5.7**, the code above can be rewritten as follows:

```
string [] developers = usersInGroups({"jira-developers"});
string [] administrators = usersInGroups({"jira-administrators"});
return arrayUnion(developers, administrators);
```

usersInRole

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

usersInRole(project, role)

Description:

Returns the users that corespond to a certain role on the specified project.

Parameters:

Parameter name	Type	Required	Description
Project key	String	Yes	The key of the selected project
Role name	String	Yes	The name of the role that is verified

Return type:

string []

Returns a list with the usernames of the users who have the given role on the specified project.

Example:

Example 1:

```
//The following users have the role developer in the project PRJ: dev1,
dev2, dev3.
string role;
string[] userbyrole;
role = "developer";
userbyrole = usersInRole(project, role);
print ("The following users have the role " + role + " in the project" +
project + ":");
print(userbyrole);
```

Result:*The following users have the role developer in the project PRJ: dev1, dev2, dev3* (Check the next row beginning with <StringPrintRoutine> for the values.)

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

---

JIRA Administration

We were requested to create some routines to deal with project setup. Thus, we created the basis for the library of routines that can help you eventually to set up "template" projects in your JIRA.

There are many routines you may be missing, so please, if you need a new one, let us know.

> Some of the routines below **have the potential to ruin your JIRA installation**!
>
> You need to have administrator rights to run them and you need to take care when updating your projects.
>
> A very simple example how you can mess the JIRA install is this: you have an existing project, several issue types, and you set up a new issue type scheme which does not contain all the issue types previously in the project. The result is undefined, and most probably JIRA will complain and / or will throw exceptions because the set issue type scheme routine **does not migrate** the existing issues with non-existent types.

Routines summary

Routine	Description
admCreateProject	Creates a new project, optionally offering support for category of the project
admUpdateProject	Updates the project properties
admProjectProperties	Returns the project properties
admAddProjectVersion	Adds a version in the project
admGetProjectVersions	Returns the project versions (all)
admAddProjectComponent	Adds a component in the project
admGetProjectComponents	Returns the project components (all)
admGetProjectComponent	Returns a JComponent(id, name, description, lead) for a specified project and component name.

admGetProjectPermissionScheme	Returns the project permission scheme
admSetProjectPermissionScheme	Updates the project permission scheme
admGetProjectNotificationScheme	Returns the project notification scheme
admSetProjectNotificationScheme	Updates the project notification scheme
admGetProjectIssueSecurityScheme	Returns the project issue security scheme
admSetProjectIssueSecurityScheme	Updates the project issue security scheme
admGetIssueTypeScheme	Returns the project issue type scheme
admSetIssueTypeScheme	Updates the project issue types scheme
admGetIssueTypeScreenScheme	Returns the project issue type screen scheme
admSetIssueTypeScreenScheme	Updates the project issue type screen scheme
admGetProjectWorkflowScheme	Returns the project workflow scheme
admSetProjectWorkflowScheme	Updates the project workflow scheme
admClearCache	Clears the internal cache.
admGetProjectDefaultIssueSecurityLevel	Returns the project's default issue security level, as set by the scheme
admGetProjectIssueSecurityLevels	Returns the all the project issue security levels, as set by the scheme
admGetProjectIssueSecurityLevelsFor	Returns the all the project issue security levels, as set by the scheme, for the specified user
admReindexIssue	Triggers a re-index for the issue with the given key
admDeactivateUser	De-activates a user (sets its active flag to false or removes it from all groups associated to global login permissions).
admActivateUser	Activates a user, setting its active flag to true, if update is allowed.
admAddScreenToTransition	Sets the screen for a workflow transition.
admCreateCustomField	Creates a new custom field, offering support also for setting its context and searcher.
admAddCustomFieldAlias	Adds a custom field alias in the sil.aliases file.
admCreateScreen	Creates a new screen.
admAddFieldToScreen	Adds a field(custom field or system field) to a screen.
admUpdateProjectVersion	Updates the version of the project by the id field.

> We started to add these administration routines with **katl-commons version 2.5.8**. Prior versions do not have the routines in place.

admActivateUser

> **Availability**
> This routine is available since **katl-commons 2.6.10**.

Syntax:

admActivateUser(username)

Description:

Activates a user, setting its active flag to true, if update is allowed.

Parameters:

Parameter name	Type	Required	Description
username	String	Yes	The username or key

Return type:

boolean

true if operation succeeded, false otherwise

Example:

```
admActivateUser("testuser");
```

admAddProjectComponent

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax:

admAddProjectComponent(pkey, compname, compdesc, complead, defaultIsUnassigned)

Description:

Adds a component in the project

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key
compname	string	yes	The component name
compdesc	string	yes	The description
complead	string	yes	The component lead
defaultIsUnassigned	boolean	yes	If true, the default is unassigned. If false, if lead has a non-null value, it is assigned to the component lead by default, otherwise to the project default

Return type:

boolean

True if the component was added, false otherwise

admAddProjectVersion

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax:

admAddProjectVersion(pkey, versionName, versionDescription, releaseDate)

Description:

Adds a version in the project

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key
versionName	string	yes	The version name
versionDescription	string	yes	The description
releaseDate	date	yes	The release date

Return type:

boolean

True if the version was added, false otherwise

> **Duplicates versions**
> It doesn't check a for a the version name already created. So far, JIRA controller code allows the project version creation action.
>
> In a future release of JJUPIN&KATL-COMMONS we'll make the extra checks of the existing project version names, as it works in JIRA interface.

admAddScreenToTransition

> **Availability**
> This routine is available since **katl-commons 2.5.19 / 2.6.11.**

Syntax:

admAddScreenToTransition(workflow, transition, screen)

Description:

Sets the screen for a workflow transition.

Parameters:

Parameter name	Type	Required	Description
workflow	string	yes	The workflow name
transition	string	yes	The transition name or id
screen	string	yes	The screen name

Return type:

boolean

True if the screen was set on the workflow transition, false otherwise.

Example:

```
admAddScreenToTransition("My workflow", "Start Progress", "Default
Screen");
```

The "Default Screen" will be set on the "Start Progress" transition for the workflow "My Workflow".

> You cannot set a transition screen for the system workflow, as it is not editable.
> If the workflow is active and already has a draft, the transition screen will only be set on the draft workflow, so you will need to publish the draft for the changes to take effect.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

admArchiveProjectVersion

> **Availability**
> This routine is available since **katl-commons 2.5.16 / 2.6.8 .**

Syntax:

admArchiveProjectVersion(projectKey, versionName, archive)

Description:

Returns empty string(not relevant)

Parameters:

Parameter name	Type	Required	Description
projectKey	string	yes	The project key
versionName	string	yes	The version name
archive	boolean	yes	Flag for archive(true) or unarchive(false)

> **Notes**
> 1. If projectKey or versionName are empty, an error will be raised.
> 2. If project or version does not exist, an error will be raised.

Return type:

string

The return type doesn't have a meaning..

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

admClearCache

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax:

admClearCache()

Description:

Clears the internal cache.

Clears the cache after you have performed a modification directly into the JIRA database (something which is **not** recommended!), allowing JIRA to pick up the change.

Parameters:

none

Return type:

boolean

Always uninitialized (false). You can safely ignore the return value of this routine.

Example:

```
admClearCache();
```

admCreateCustomField

> **Availability**
> This routine is available since **katl-commons 2.5.19 / 2.6.11.**

Syntax:

admCreateCustomField(fieldName, description, fieldType, fieldSearcher, projects, issueTypes)

Description:

Creates a new custom field, offering support also for setting its context and searcher.

Parameters:

Parameter name	Type	Required	Description
fieldName	string	yes	The custom field name
description	string	yes	The custom field description (can be blank)
fieldType	string	yes	The custom field type (either key or name)
fieldSearcher	string	yes	The custom field searcher (either key or name). If blank, the default custom field searcher for the given type will be considered.
projects	string[]	yes	The projects context (project keys). If empty, global issue context will be considered.
issueTypes	string[]	yes	The issue types context (either names or ids). If empty, all issue types will be considered.

Return type:

string

Returns the string id (customfield_xxxxx) of the newly created custom field.

Example:

Example 1:

Creating a single line text field with default searcher (Free Text Searcher) and global context:

```
admCreateCustomField("Test Field", "test description", "Text Field (single
line)", "", {}, {});
```

Example 2:

Creating a multi-line text field with blank description, specified searcher name (Free Text Searcher) and specified project and issue types context:

```
admCreateCustomField("Test Field", "", "Text Field (multi-line)", "Free
Text Searcher", {"DEMO", "TEST"}, {"Bug", "Improvement"});
```

Example 3:

Creating a date picker field with blank description, specified searcher key, and specified issue types ids context:

```
admCreateCustomField("Test Field", "", "Date Picker",
"com.atlassian.jira.plugin.system.customfieldtypes:daterange", {}, {1, 2,
3});
```

If the provided custom field searcher key or name is wrong, it will be ignored and the custom field will be created with no searcher configured.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

admCreateProject

Availability
This routine is available since **katl-commons 2.5.8 .**

Syntax:

admCreateProject(pkey, pname, description, lead, url, categoryName, defaultIsUnassigned, avatarId)

Since version 3.1.0

admCreateProject(pkey, pname, description, lead, url, categoryName, defaultIsUnassigned, avatarId, projectTypeKey)

Description:

Creates a new project, optionally offering support for category of the project

Creates the project and configures it with the default security level, default workflow scheme, etc.

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key
pname	string	yes	The project name.
description	string	yes	The project description.
lead	string	yes	Represents a valid user that should be assignable on the project
url	string	yes	The URL of the project. Leave blank if no URL is available
categoryName	string	yes	The name of the category
defaultIsUnassigned	boolean	yes	Set it to true if you don't want to assign issues to the project lead above
avatarId	numeric	yes	Set it to a negative number or zero for the default avatar id. Otherwise, you need to choose a valid avatar id.
projectTypeKey	string	no	Set the project type key. If it is not set, the default value is "business".

Return type:

boolean

True if the project was created, false if not

Example1:

```
admCreateProject("T3", "Thunderbolt3", "Three thunderbolts in a single
hole", "zeus", "http://thunderbolts.olimpus.gr", "PUBLIC", false, 0);
```

Example2:

```
admCreateProject("T3", "Thunderbolt3", "Three thunderbolts in a single
hole", "zeus", "http://thunderbolts.olimpus.gr", "PUBLIC", false, 0,
"business");
```

Duplicates projects
It doesn't check a for a the project key existence or the project name already created. So far, JIRA controller code allows the project version creation action.

In a future release of JJUPIN&KATL-COMMONS we'll make the extra checks of the existing project keys and names, as it works in JIRA interface.

You will need permissions to create a project.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

admDeactivateUser

Availability
This routine is available since **katl-commons 2.5.18** (JIRA 5.x) **/ 2.6.10** (JIRA 6.x).

Syntax:

admDeactivateUser(username)

Description:

De-activates a user (sets its active flag to false or removes it from all groups associated to global login permissions).

Parameters:

Parameter name	Type	Required	Description
username	String	Yes	The username or key

Return type:

boolean

true if operation succeeded, false otherwise

Example:

```
admDeactivateUser("testuser");
```

admGetFieldConfigurationScheme

> **Availability**
> This routine is available since **katl-commons 2.5.13 / 2.6.5 .**

Syntax:

admGetFieldConfigurationScheme(pkey)

Description:

Returns the project field configuration scheme

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key

Return type:

string

The requested scheme name or an empty string if the default field configuration scheme is used.

admGetIssueTypeScheme

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax:

admGetIssueTypeScheme(pkey)

Description:

Returns the project issue type scheme

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key

Return type:

string

The requested scheme name.

admGetIssueTypeScreenScheme

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax:

admGetIssueTypeScreenScheme(pkey)

Description:

Returns the project issue type screen scheme

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key

Return type:

string

The requested scheme name.

admGetProjectComponents

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax:

admGetProjectComponents(pkey)

Description:

Returns the project components (all)

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key

Return type:

string [] (array of strings)

All the project components names, as returned by the underlying JIRA layer.

admGetProjectDefaultIssueSecurityLevel

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax:

admGetProjectDefaultIssueSecurityLevel(pkey)

Description:

Returns the project's default issue security level, as set by the scheme

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key

Return type:

number

The requested default security level.

admGetProjectIssueSecurityLevels

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax:

admGetProjectIssueSecurityLevels(pkey)

Description:

Returns the all the project issue security levels, as set by the scheme

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key

Return type:

number [] (array of numbers)

The requested security levels ids.

admGetProjectIssueSecurityLevelsFor

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax:

admGetProjectIssueSecurityLevelsFor(pkey, user)

Description:

Returns the all the project issue security levels, as set by the scheme, for the specified user

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key
user	string	yes	The user

Return type:

number [] (array of numbers)

The requested security levels ids for that user on that project.

admGetProjectIssueSecurityScheme

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax:

admGetProjectIssueSecurityScheme(pkey)

Description:

Returns the project issue security scheme

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key

Return type:

string

The requested scheme name.

admGetProjectNotificationScheme

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax:

admGetProjectNotificationScheme(pkey)

Description:

Returns the project notification scheme

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key

Return type:

string

The requested scheme name.

admGetProjectPermissionScheme

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax:

admGetProjectPermissionScheme(pkey)

Description:

Returns the project permission scheme

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key

Return type:

string

The requested scheme name.

admGetProjectVersion

> **Availability**
> This routine is available since **katl-commons 2.5.16 / 2.6.8 .**

Syntax:

admGetProjectVersion(projectKey, versionName)

Description:

Returns the project version

Parameters:

Parameter name	Type	Required	Description
projectKey	string	yes	The project key
versionName	string	yes	The version name

Return type:

Version

All the properties of a version(id, name, description, projectKey, startDate, releaseDate, archived, released).

> **Notes**
> 1. If projectKey or versionName are empty, an error will be raised.
> 2. If project or version does not exist, an error will be raised.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

admGetProjectVersions

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax:

admGetProjectVersions(pkey)

Description:

Returns the project versions (all)

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key

Return type:

string [] (array of strings)

All the project versions, unsorted, as returned by the underlying JIRA layer. The meaning that v2 comes after v1.3 is to be addressed in the script, not in this routine.

admGetProjectWorkflowScheme

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax:

admGetProjectWorkflowScheme(pkey)

Description:

Returns the project workflow scheme

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key

Return type:

string

The requested scheme name.

Remarks:

Because JIRA "**Default Workflow Scheme"** has no name in the database, if the routine returns the empty string that means the default scheme is currently attached to the specified project.

admProjectExists

> **Availability**
> This routine is available since **katl-commons 2.5.14 / 2.6.6 .**

Syntax:

admProjectExists(project key)

Description:

Returns true if project with provided key exists, false otherwise

Parameters:

Parameter name	Type	Required	Description
project key	string	yes	The project key

Return type:

boolean

True if project exists, false otherwise.

Example:

```
boolean prjExist = admProjectExists("TEST");
```

admProjectProperties

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax:

admProjectProperties(pkey)

Description:

Returns the project properties

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key

Return type:

Project

The properties of the project.

The returned properties keys are:

Key	Description
key	Project key (for completitude, it's the same as the one given as parameter)
name	Name of the project
description	The description of the project
lead	The lead
url	The URL of the project
unassignedByDefault	true/false, depending of the assignment option
avatarId	The avatar id, zero if no avatar
category	The project category name

Example:

```
JProject prj = admProjectProperties("T3");

string msg1 = "Project description is:"  + prj["description"] + "; category
is:" + prj["category"];
string msg2 = "This project belongs to:" + prj["lead"];
number avatarId = prj["avatarId"];
```

admReindex

> **Availability**
> This routine is available since **katl-commons 2.5.13 / 2.6.5 .**

Syntax:

admReindex()

Description:

Triggers a reindex for all issues.

Return type:

number

Returns a number representing the duration of the reindex in milliseconds.

> The routine is synchronous. Even though the task will be run on multiple threads, the current thread will be stuck inside the routine until

the reindex is done.

admReindexIssue

Availability
This routine is available since **katl-commons 2.5.17 / 2.6.9 .**

Syntax:

admReindexIssue(issueKey)

Description:

Triggers a re-index for the issue with the given key

Parameters:

Parameter name	Type	Required	Description
issueKey	string	Yes	the key of the selected issue

Return type:

number

Returns true if the re-index succeeds, false if some exception appears.

admReleaseProjectVersion

Availability
This routine is available since **katl-commons 2.5.16 / 2.6.8 .**

Syntax:

admReleaseProjectVersion(projectKey, versionName, release)

Description:

Returns empty string(not relevant)

Parameters:

Parameter name	Type	Required	Description
projectKey	string	yes	The project key
versionName	string	yes	The version name
release	boolean	yes	Flag for release(true) or unrelease(false)

Notes
1. If projectKey or versionName are empty, an error will be raised.
2. If project or version does not exist, an error will be raised.

Return type:

string

The return type doesn't have a meaning..

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

admSetFieldConfigurationScheme

> **Availability**
> This routine is available since **katl-commons 2.5.13 / 2.6.5 .**

Syntax:

admSetFieldConfigurationScheme(pkey, schemeName)

Description:

Updates the project field configuration scheme

Both the scheme and the project must exist. If the scheme name is an empty string, then the default field configuration scheme will be set.

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key. Must exist
schemeName	string	yes	The scheme name. The scheme must exist or must be an empty string. If passed an empty string, the default field configuration scheme will be set. Otherwise the setter won't change the previously scheme set.

Return type:

boolean

True if the project was updated, false if not

admSetIssueTypeScheme

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax:

admSetIssueTypeScheme(pkey, schemeName)

Description:

Updates the project issue types scheme

Both the scheme and the project must exist.

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key. Must exist
schemeName	string	yes	The scheme name. The scheme must exist

Return type:

boolean

True if the project was updated, false if not

admSetIssueTypeScreenScheme

Availability
This routine is available since **katl-commons 2.5.8 .**

Syntax:

admSetIssueTypeScreenScheme(pkey, schemeName)

Description:

Updates the project issue type screen scheme

Both the scheme and the project must exist.

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key. Must exist
schemeName	string	yes	The scheme name. The scheme must exist

Return type:

boolean

True if the project was updated, false if not

admSetProjectIssueSecurityScheme

Availability
This routine is available since **katl-commons 2.5.8 .**

Syntax:

admSetProjectIssueSecurityScheme(pkey, schemeName)

Description:

Updates the project issue security scheme

Both the scheme and the project must exist.

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key. Must exist
schemeName	string	yes	The scheme name. The scheme must exist. Otherwise the setter won't change the previously scheme set.

Return type:

boolean

True if the project was updated, false if not

admSetProjectNotificationScheme

Availability
This routine is available since **katl-commons 2.5.8 .**

Syntax:

admSetProjectNotificationScheme(pkey, schemeName)

Description:

Updates the project notification scheme

Both the scheme and the project must exist.

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key. Must exist
schemeName	string	yes	The scheme name. The scheme must exist

Return type:

boolean

True if the project was updated, false if not

admSetProjectPermissionScheme

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax:

admSetProjectPermissionScheme(pkey, schemeName)

Description:

Updates the project permission scheme

Both the scheme and the project must exist.

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key. Must exist
schemeName	string	yes	The scheme name. The scheme must exist. Otherwise the setter won't change the previously scheme set.

Return type:

boolean

True if the project was updated, false if not

admSetProjectVersionReleaseDate

> **Availability**
> This routine is available since **katl-commons 2.5.16 / 2.6.8 .**

Syntax:

admSetProjectVersionReleaseDate(projectKey, versionName, releaseDate)

Description:

Returns empty string(not relevant)

Parameters:

Parameter name	Type	Required	Description
projectKey	string	yes	The project key
versionName	string	yes	The version name

releaseDate	date	yes	The release date

Notes
1. If projectKey or versionName are empty, an error will be raised.
2. If project or version does not exist, an error will be raised.
3. releaseDate must be a date after the start date, otherwise an error will be raised.

Return type:

string

The return type doesn't have a meaning..

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

admSetProjectVersionStartDate

Availability
This routine is available since **katl-commons 2.6.8 .**

Syntax:

admSetProjectVersionStartDate(projectKey, versionName, startDate)

Description:

Returns empty string(not relevant)

Parameters:

Parameter name	Type	Required	Description
projectKey	string	yes	The project key
versionName	string	yes	The version name
startDate	date	yes	The start date

Notes
1. If projectKey or versionName are empty, an error will be raised.
2. If project or version does not exist, an error will be raised.
3. startDate must be a date before the release date, otherwise an error will be raised.

Return type:

string

The return type doesn't have a meaning..

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

admSetProjectWorkflowScheme

Availability
This routine is available since **katl-commons 2.5.8 .**

Syntax:

admSetProjectWorkflowScheme(pkey, schemeName)

Description:

Updates the project workflow scheme

Both the scheme and the project must exist.

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key. Must exist
schemeName	string	yes	The scheme name. The scheme must exist

Return type:

boolean

True if the project was updated, false if not

Remarks

When using an empty or null scheme name, the routine will set the default workflow scheme on the specified project.

In other words

```
string pkey = "TEST";
string schemeName;
return admSetProjectWorkflowScheme(pkey, schemeName);
```

equivalent to

```
string pkey = "TEST";
string schemeName = "";
return admSetProjectWorkflowScheme(pkey, schemeName);
```

will return true and will set the default JIRA workflow scheme (**Default Workflow Scheme**).

Otherwise, when using a nonexistent workflow scheme name, the routine will return false and will not make any change.

admUpdateProject

> **Availability**
> This routine is available since **katl-commons 2.5.8 .**

Syntax:

admUpdateProject(pkey, pname, description, lead, url, categoryName, defaultIsUnassigned, avatarId)

Description:

Updates the project properties

The project key cannot be changed, it's the only thing that identifies the project

Parameters:

Parameter name	Type	Required	Description
pkey	string	yes	The project key. Must exist

pname	string	yes	The project name.
description	string	yes	The project description.
lead	string	yes	Represents a valid user that should be assignable on the project
url	string	yes	The URL of the project. Leave blank if no URL is available
categoryName	string	yes	The name of the category
defaultIsUnassigned	boolean	yes	Set it to true if you don't want to assign issues to the project lead above
avatarId	numeric	yes	Set it to a negative number or zero for the default avatar id. Otherwise, you need to choose a valid avatar id.

Return type:

boolean

True if the project was updated, false if not

Example:

```
admUpdateProject("T3", "Thunderbolt3", "Three thunderbolts in a single
hole", "zeus", "http://thunderbolts.olimpus.gr", "PUBLIC", false, 12010);
```

1. You will need permissions to update the project.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

admCreateScreen

Availability
This routine is available since **katl-commons 2.5.19 / 2.6.11.**

Syntax:

admCreateScreen(name, description)

Description:

Creates a new screen.

Parameters:

Parameter name	Type	Required	Description
name	string	yes	The screen name
description	string	yes	The screen description (can be blank)

Return type:

string

Returns the id (as a number) of the newly created screen.

Example:

Example 1:

Creating a screen:

```
admCreateScreen("Test Screen", "Test screen description");
```

Example 2:

Creating a screen with empty description:

```
admCreateScreen("Test Screen", "");
```

> The method throws an exception if the name parameter is empty or a screen with the same name already exists.

See also:

admAddCustomFieldAlias

> **Availability**
> This routine is available since **katl-commons 2.5.19 / 2.6.11.**

Syntax:

admAddCustomFieldAlias(customField, alias)

Description:

Adds a custom field alias in the sil.aliases file.

Parameters:

Parameter name	Type	Required	Description
customField	string	yes	The custom field string id, name or existing alias
alias	string	yes	The new custom field alias to set

Return type:

boolean

True if the custom field alias was added successfully in the sil.aliases file or already exists, false otherwise.

Example:

Example 1:

Setting a custom field alias using custom field id:

```
admAddCustomFieldAlias("customfield_10000", "TestAlias");
```

Example 2:

Setting a custom field alias using custom field name:

```
admAddCustomFieldAlias("Text Field", "TestAlias");
```

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

admAddFieldToScreen

> **Availability**
> This routine is available since **katl-commons 2.5.19 / 2.6.11.**

Syntax:

admAddFieldToScreen(scrName, fieldAsStr)

or
admAddFieldToScreen(scrName, fieldAsStr, scrTabName)

or
admAddFieldToScreen(scrName, fieldAsStr, scrTabName, indexInTab)

Description:

Adds a field(custom field or system field) to a screen.

Parameters:

Parameter name	Type	Required	Description
scrName	string	yes	The screen name
fieldAsStr	string	yes	The field as a string(the field id for system fields, the field id as string, the field name or alias for custom fields)
scrTabName	string	no	The tab name(if not provided, the first tab will be used to add the field)
indexInTab	number	no	The position in tab(if not provided, or -1, the field will be the last in the tab)

Return type:

string

Returns true if the field is successfully added to the screen, false otherwise.

> The method throws an exception if the screen name or field parameters are empty or don't refer to a valid screen/field. Also an exception is thrown if the field is already present in the screen.
>
> If the tab name is provided and a tab with this name does't exist in the screen, it will be created and added as the last tab in the screen.

Example:

Example 1:

Adding a system field to a screen, in the first tab, default(last position) in tab:

```
admAddFieldToScreen("TestScreen", "priority");
```

Example 2:

Adding a system field to a screen, in the tab "TestTab", first position in tab:

```
admAddFieldToScreen("TestScreen", "priority", "TestTab", 0);
```

Example 3:

Adding a custom field to a screen, in the tab "TestTab", default(last) position in tab:

```
admAddFieldToScreen("TestScreen", "customfield_10000", "TestTab");
```

See also:

admUpdateProjectVersion

> **Availability**
> This routine is available since **katl-commons 3.0**

Syntax:

admUpdateProjectVersion(versionStruct)

Description:

Updates the version of the project by the id field.

The version name of the project is unique and it has an id version that corresponds with it.

1. This routine updates only the **name**, **description**, **startDate** and **releaseDate** fields. The archived and released fields can be updated using admArchiveProjectVersion and admReleaseProjectVersion routines.
2. **Name** must be unique

Parameter:

Parameter name	Type	Required	Description
versionStruct	JVersion	yes	Version id and the version name should have values assigned.

The parameter versionStruct should have the type JVersion which it is described here.

Return type:

string
You can safely ignore the return value of this routine.

Example:

```
JVersion jVersion;
jVersion.id = 10201;
jVersion.name = "Version2";
jVersion.description = "This is the second version.";
jVersion.startDate = "2014-10-27";
jVersion.releaseDate = "2014-10-30";
return admUpdateProjectVersion(jVersion);
```

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

admGetProjectComponent

Availability
This routine is available since **katl-commons 3.0.2.**

Syntax:

admGetProjectComponent(projectKey, componentName)

Description:

Returns a JComponent(id, name, description, lead) for a specified project and component name.

Parameters:

Parameter name	Type	Required	Description
projectKey	string	yes	The project key
componentName	string	yes	The name of the component.

Return type:

JComponent

Example:

```
admGetProjectComponent("TEST", "sil");
```

Returns a JComponent structure for the component named "sil" from the project "TEST".

See also:

admClearLinksCache

Availability
This routine is available since **katl-commons 3.0.2.**

Syntax:

admCreateLinksCache()

Description:

Clears the issue links cache.

Parameters:

none

Return type:

string

Returned value has no meaning.

Example:

```
admClearCache();
```

See also:

File Manipulation Routines

Introduction

This section contains a collection of routines for handling directories and files.

Routines summary

Routine	Description
createDirectory	Returns true if the directory was created successfully, false otherwise. If returned false check the log for a detailed reason on why it failed.
createFile	Creates an empty file. It also returns true if the file was created successfully, false otherwise. If returned false check the log for a detailed reason on why it failed.
deleteFile	Deletes a file. It also returns true if the file was deleted successfully, false otherwise. If returned false check the log for a detailed reason on why it failed.
directoryExists	Returns true if the directory exists, false otherwise.
fileContains	Returns true if the file contains any string that matches the specified regex. If the regex contains a backslash, please replace it with two backslashes, or else you will get a syntax error.
fileCopy	Copy a file from one location to another. Returns true if the file was copied successfully, false otherwise. If false is returned check the log for a detailed reason on why it failed.
fileExists	Returns true if the file exists and false otherwise.
findFiles	Searches for **files** that match the given regex, in the specified folder. The routine only searches for files and NOT directories. Returns a list with **absolute** paths for files that match the given regex. The regex match is done on the file name, not the full path. If the regex parameter is empty string, the routine returns an empty result
printInFile	Prints in the specified file the provided value. It appends the value to the file. For best results, use absolute paths, since relative paths are resolved starting with the current working directory.
readFromTextFile	Read the text of the file

createDirectory

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

createDirectory(path_to_directory)

Description:

Returns true if the directory was created successfully, false otherwise. If returned false check the log for a detailed reason on why it failed.

Parameters:

Parameter name	Type	Required	Description
path_to_directory	string	Yes	Specifies a directory or a path.

Return type:

boolean (true/false)

Example:

```
string dir = "C:/myDirectory";
if(directoryExists(dir)){
    print("The directory exists!");
} else {
    if(!createDirectory(dir)){
        print("Failed to create directory " + dir);
    }
}
```

Notes:

1. It is recommended that you use forward slashes (/) for file paths.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

createFile

Availability
This routine is available since **katl-commons 3.0.2.**

Syntax:

createFile(path_to_file)

Description:

Creates an empty file. It also returns true if the file was created successfully, false otherwise. If returned false check the log for a detailed reason on why it failed.

Parameters:

Parameter name	Type	Required	Description
path_to_file	string	Yes	Specifies the file name to create.

Return type:

boolean (true/false)

Example:

```
createFile("C:/fileToCreate.txt");
```

Notes:

1. It is recommended that you use forward slashes (/) for file paths.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

deleteFile

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

deleteFile(path_to_file)

Description:

Deletes a file. It also returns true if the file was deleted successfully, false otherwise. If returned false check the log for a detailed reason on why it failed.

Parameters:

Parameter name	Type	Required	Description
path_to_file	string	Yes	Specifies the file name to delete.

Return type:

boolean (true/false)

Example:

```
deleteFile("C:/fileToDelete.txt");
```

Notes:

1. It is recommended that you use forward slashes (/) for file paths.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

directoryExists

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

directoryExists(path_to_directory)

Description:

Returns true if the directory exists, false otherwise.

Parameters:

Parameter name	Type	Required	Description
path_to_directory	string	Yes	Specifies the name of the directory to locate.

Return type:

boolean (true/false)

Example:

```
if(directoryExists("C:/myDirectory")){
    print("The directory exists!");
} else {
    print("The directory does not exist!");
}
```

Notes:

1. It is recommended that you use forward slashes (/) for file paths.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

fileContains

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

fileContains(path_to_file, regex)

Description:

Returns true if the file contains any string that matches the specified regex. If the regex contains a backslash, please replace it with two backslashes, or else you will get a syntax error.

Parameters:

Parameter name	Type	Required	Description
path_to_file	string	Yes	Specifies the file name.
regex	string	Yes	Specifies search string into the file path_to_file.

Return type:

boolean (true/false)

Example:

```
//This will match any "com.keplerrominfo.jira" occurence in myfile.txt.
fileContains("C:/myfile.txt","com\\.keplerrominfo.jira");
```

Notes:

1. It is recommended that you use absolute file paths withforward slashes (/).

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

fileCopy

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

fileCopy(path_to_source, path_to_destination)

Description:

Copy a file from one location to another. Returns true if the file was copied successfully, false otherwise. If false is returned check the log for a detailed reason on why it failed.

Parameters:

Parameter name	Type	Required	Description
path_to_source	string	Yes	Specifies the source file name.
path_to_destination	string	Yes	Specifies the destination file name.

Return type:

boolean (true/false)

Example:

```
fileCopy("C:/source.txt", "C:/destination.txt");
```

If the file ***—source.txt*** was copied to the file ***—destination.txt*** , return ***—true*** .

Notes:

It is recommended that you use forward slashes (/) for file paths.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

fileExists

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

fileExists(path_to_file)

Description:

Returns true if the file exists and false otherwise.

Parameters:

Parameter name	Type	Required	Description
path_to_file	string	Yes	Specifies the file name you want to search for.

Return type:

boolean (true/false)

Example:

```
if(fileExists("C:/someFile.txt")){
    print("The file exists!");
} else {
    print("The file does not exist!");
}
```

Notes:

1. It is recommended that you use forward slashes (/) for file paths.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

findDirectories

Availability
This routine is available since **katl-commons 2.5.15 / 2.6.7 .**

Syntax:

findDirectories(directory, regex)

Description:

Searches for **directories** that match the given regex, in the specified folder. The routine only searches for directories and NOT files.

Returns a list with **absolute** paths for directories that match the given regex.

The regex match is done on the file name, not the full path. If the regex parameter is empty string, the routine returns an empty result.

Parameters:

Parameter name	Type	Required	Description
directory	string	Yes	Specifies the directory to search into.
regex	string	Yes	Specifies the file pattern to search for.

Return type:

string []

Returns an array containing the absolute paths of all directories from the given directory whose name match the regex.

Example:

```
findDirectories("C:/JIRA/plugins", "kepler.*");
```

Results: An array containing all absolute paths for folders starting with 'kepler' from the folder C:\JIRA\plugins.

Notes:

1. It is recommended that you use forward slashes (/) for file paths.
2. If the regex contains a backslash, please replace it with two backslashes, or else you will get a syntax error.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

findFiles

Availability
This routine is available since **katl-commons 1.1.2 .**

Syntax:

findFiles(directory, regex)

Description:

Searches for **files** that match the given regex, in the specified folder. The routine only searches for files and NOT directories.

Returns a list with **absolute** paths for files that match the given regex.

The regex match is done on the file name, not the full path. If the regex parameter is empty string, the routine returns an empty result

Parameters:

Parameter name	Type	Required	Description
directory	string	Yes	Specifies the directory to search into.
regex	string	Yes	Specifies the file pattern to search for.

Return type:

string []

Returns an array containing the absolute paths of all files from the given directory whose name match the regex.

Example:

```
findFiles("C:/JIRA/plugins", ".*\\.jar");
```

Results: An array containing all absolute paths for jar files from the folder C:\JIRA\plugins.

Notes:

1. It is recommended that you use forward slashes (/) for file paths.
2. If the regex contains a backslash, please replace it with two backslashes, or else you will get a syntax error.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

printInFile

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

printInFile(filepath, var)

Description:

Prints in the specified file the provided value. It appends the value to the file. For best results, use absolute paths, since relative paths are resolved starting with the current working directory.

Parameters:

Parameter name	Type	Required	Description
filepath	string	Yes	Specifies the file name to write in.
var	string	Yes	Specifies the character expression to write into the file filepath.

Return type:

Empty value (The returned value has no meaning.)

Example:

```
printInFile("C:/story.txt", "Once upon a time...");
```

Prints to the file ***—story.txt*** the record ***—Once upon a time...***

Notes:

1. It is recommended that you use forward slashes (/) for file paths. As a general observation, please use the silEnv() routine to create an absolute path.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

readFromTextFile

> **Availability**
> This routine is available since **katl-commons 2.5.13 / 2.6.5 .**

Syntax:

readFromTextFile(path)

Description:

Read the text of the file

Parameters:

Parameter name	Type	Required	Description
path	string	Yes	Specifies the file name to read from.

Return type:

string

The text of the file

Example:

```
string fileContent = readFromTextFile("C:/story.txt");
```

Notes:

1. You can use absolute paths and relative paths to "sil.home".
2. If the file is not found, an error will be raised.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

String Routines

Introduction

This section contains a collection of routines for handling strings.

Routines summary

Routine	Description
chop	Returns a string, chopped at ***—nmb*** characters. Cuts down the string (if length is greater than ***—nmb*** characters) to a string of max ***—nmb*** chars.
contains	Returns ***—true*** if character expression ***—str2*** is in character expression ***—str1***.
endsWith	Returns ***—true*** if ***—str1*** ends with ***—str2***.

indexOf	Returns the index of the first match of the ***—str2*** in ***—str1*** or -1 if ***—str2*** is nowhere to be found in ***—str1***.
lastIndexOf	Returns the index of the last match of the ***—str2*** in ***—str1*** or -1 if ***—str2*** is nowhere to be found in ***—str1***.
isAlpha	Returns ***—true*** if the provided argument ***—str*** is a string containing only letters.
isAlphaNumeric	Returns ***—true*** if the provided argument ***—str*** is a string containing only letters and digits.
isLower	Returns ***—true*** if the provided argument ***—str*** is a string containing only lowercase letters.
isNumeric	Returns ***—true*** if the provided argument ***—str*** is a string containing only digits.
isUpper	Returns ***—true*** if the provided argument ***—str*** is a string containing only upper letters.
length	Returns the length of the provided string, 0 if the string is null or has no chars.
replace	Replaces the ***—search_str*** string with ***—replacement_str*** in ***—str*** and returns the resulting string. The string ***—str*** is not modified.
split	Returns the array of strings computed by splitting this string around matches of the given regular expression.
startsWith	Returns ***—true*** if ***—str1*** starts with ***—str2***.
substring	Returns substring is starting at index ***—start*** and stop at index ***—stop***.
toLower	Returns the string only with **lower case letters**.
toUpper	Returns the string only with **upper case letters**.
trim	Returns a trimmed copy of the string passed as parameter with the leading and trailing whitespaces removed.
matches	Returns ***—true*** if character expression ***—string*** matches the regular expression ***—regex***.

See Also:

Syntax
Variable Resolution

chop

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

chop(str, nmb)

Description:

Returns a string, chopped at ***—nmb*** characters. Cuts down the string (if length is greater than ***—nmb*** characters) to a string of max ***—nmb*** chars.

Returns ***—nmb*** characters from ***—str***, starting with the leftmost character.

Parameters:

Parameter name	Type	Required	Description
str	string	Yes	Specifies a character expression.
nmb	number	Yes	Specifies the number of characters returned from the character expression str.

Return type:

string

Example:

Example 1:

```
wret = chop("Once upon a time", 6);
print(wret);
```

Print ***—Once u***

Example 2:

```
wret = chop("Once upon a time", 30);
print(wret);
```

Print ***—Once upon a time***

Example 3:

```
wret = chop("Once upon a time", -1);
print(wret);
```

Print ***—Once upon a time***

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

contains

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

contains(str1, str2)

Description:

Returns ***—true*** if character expression ***—str2*** is in character expression ***—str1***.

Parameters:

Parameter name	Type	Required	Description
str1	string	Yes	Specifies a character expression to search for str2.
str2	string	Yes	Specifies a character expression to search for in str1.

Return type:

boolean (true/false)

Example:

Example 1:

```
wret = contains("This will return ?", "will");
print("Return " + wret);
```

Print ***—Return true***

Example 2:

```
wret = contains("This will return ?", "Will");
print("Return " + wret);
```

Print ***—Return false***

Example 3:

```
wret = contains("This will return ?", "This");
print("Return " + wret);
```

Print ***—Return true***

Notes:

If you need to find ***—str2*** multiple times, use *—substring( )* routine.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

endsWith

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

endsWith(str1, str2)

Description:

Returns ***—true*** if ***—str1*** ends with ***—str2***.

Parameters:

Parameter name	Type	Required	Description
str1	string	Yes	Specifies a character expression to search for str2.
str2	string	Yes	Specifies a character expression to search for in str1.

Return type:

boolean (true/false)

Example:

Example 1:

```
wret = endsWith("This will return ?", "will");
print("Return " + wret);
```

Print ***—Return false***

Example 2:

```
wret = endsWith("This will return ?", "n ?");
print("Return " + wret);
```

Print ***—Return true***

Example 3:

```
wret = endsWith("This will return ?", "N ?");
print("Return " + wret);
```

Print ***—Return false***

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

indexOf

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

indexOf(str1, str2)

Description:

Returns the index of the first match of the ***—str2*** in ***—str1*** or -1 if ***—str2*** is nowhere to be found in ***—str1***.

Returns an integer indicating the position of the first character for a character expression within another character expression, beginning from the leftmost character.

Parameters:

Parameter name	Type	Required	Description
str1	string	Yes	Specifies a character expression to search for str2.
str2	string	Yes	Specifies a character expression to search for in str1.

Return type:

number

Example:

Example 1:

```
wret = indexOf("This will return ?", "will");
print("Return " + wret);
```

Print ***—Return 5***

Example 2:

```
wret = indexOf("This will return ?", "Will");
print("Return " + wret);
```

Print ***—Return -1***

Example 3:

```
wret = indexOf("This will return ?", "This");
print("Return " + wret);
```

Print ***—Return 0***

Notes:

If the first occurrence of *—str2* is the first caracter of *—str1*, indexOf( ) returns *—0*.
If you need to find ***—str2*** multiple times, use *—substring( )* routine.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

isAlpha

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

isAlpha(str)

Description:

Returns ***—true*** if the provided argument ***—str*** is a string containing only letters.

Parameters:

Parameter name	Type	Required	Description
str	string	Yes	Specifies a character expression.

Return type:

boolean (true/false)

Example:

Example 1:

```
wret = isAlpha("foobar");
print(wret);
```

Print ***—true***

Example 2:

```
wret = isAlpha("aaa2345f.ff");
print(wret);
```

Print ***—false***

Example 3:

```
wret = isAlpha("Once upon a time ... ! ");
print(wret);
```

Print ***—false***

Notes:

IsAlpha returns ***—false*** if the character string ***—str*** contains blanks or special characters.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

isAlphaNumeric

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

isAlphaNumeric(str)

Description:

Returns ***—true*** if the provided argument ***—str*** is a string containing only letters and digits.

Parameters:

Parameter name	Type	Required	Description
str	string	Yes	Specifies a character expression.

Return type:

boolean (true/false)

Example:

Example 1:

```
wret = isAlphaNumeric("foobar");
print(wret);
```

Print ***—true***

Example 2:

```
wret = isAlphaNumeric("aaa2345G");
print(wret);
```

Print ***—true***

Example 3:

```
wret = isAlphaNumeric("23asd.*;45");
print(wret);
```

Print ***—false***

Notes:

isAlphaNumeric returns ***—false*** if the character string ***—str*** contains blanks or special characters or is a null string.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

isDigit

Availability
This routine is available since **katl-commons 3.0.5 .**

Syntax:

isDigit(str)

Description:

Returns ***—true*** if the provided argument ***—str*** is a string containing only digits.

Parameters:

Parameter name	Type	Required	Description
str	string	Yes	Specifies a character expression.

Return type:

boolean (true/false)

Example:

Example 1:

```
wret = isDigit("2345");
print(wret);
```

Print ***—true***

Example 2:

```
wret = isDigit("aaa2345f.ff");
print(wret);
```

Print ***—false***

Example 3:

```
wret = isDigit("2.34");
print(wret);
```

Print ***—false***

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

isLower

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

isLower(str)

Description:

Returns ***—true*** if the provided argument ***—str*** is a string containing only lowercase letters.

Parameters:

Parameter name	Type	Required	Description
str	string	Yes	Specifies a character expression.

Return type:

boolean (true/false)

Example:

Example 1:

```
wret = isLower("foobar");
print(wret);
```

Print ***—true***

Example 2:

```
wret = isLower("aaa2345f.ff");
print(wret);
```

Print ***—false***

Example 3:

```
wret = isLower("");
print(wret);
```

Print ***—false***

Notes:

isLower returns ***—false*** if the character string ***—str*** contains blanks or special characters or is a null string.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

isNumeric

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

isNumeric(str)

Description:

Returns ***—true*** if the provided argument ***—str*** is actually a number.

Parameters:

Parameter name	Type	Required	Description
str	string	Yes	Specifies a character expression.

Return type:

boolean (true/false)

Example:

Example 1:

```
wret = isNumeric("2345");
print(wret);
```

Print ***—true***

Example 2:

```
wret = isNumeric("-2345.678");
print(wret);
```

Print ***—true***

Example 3:

```
wret = isNumeric("2345.678abcd");
print(wret);
```

Print ***—false***

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

isUpper

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

isUpper(str)

Description:

Returns ***—true*** if the provided argument ***—str*** is a string containing only upper letters.

Parameters:

Parameter name	Type	Required	Description
str	string	Yes	Specifies a character expression.

Return type:

boolean (true/false)

Example:

Example 1:

```
wret = isUpper("FOOBAR");
print(wret);
```

Print ***—true***

Example 2:

```
wret = isUpper("aAA2345f.FF");
print(wret);
```

Print ***—false***

Example 3:

```
wret = isUpper("");
print(wret);
```

Print ***—false***

Notes:

isUpper returns ***—false*** if the character string ***—str*** contains blanks or special characters or is a null string.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

lastIndexOf

Availability
This routine is available since **katl-commons 2.5.17 / 2.6.9.**

Syntax:

lastIndexOf(str1, str2)

Description:

Returns the index of the last match of the ***—str2*** in ***—str1*** or -1 if ***—str2*** is nowhere to be found in ***—str1***.

Returns the index within a string of the last occurrence of the specified substring.

Parameters:

Parameter name	Type	Required	Description
str1	string	Yes	Specifies a character expression to search for str2.
str2	string	Yes	Specifies a character expression to search for in str1.

Return type:

number

Example:

Example 1:

```
wret = lastIndexOf("f1/f2/f3", "/f");
print("Return " + wret);
```

Print ***—Return 5***

Example 2:

```
wret = lastIndexOf("f1/f2/f3", "/fi");
print("Return " + wret);
```

Print ***—Return -1***

Example 3:

```
wret = lastIndexOf("f1/f2", "");
print("Return " + wret);
```

Print ***—Return 5***

Notes:

The last occurrence of the empty string "" is considered to occur at the index value `str1.length()`
If you need to find ***—str2*** multiple times, use *—substring( )* routine.

See also:

length

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

length(str)

Description:

Returns the length of the provided string, 0 if the string is null or has no chars.

Parameters:

Parameter name	Type	Required	Description
str	string	Yes	Specifies a character expression for which LENGHT( ) returns the number of characters.

Return type:

number

Example:

```
wlen = length("This text is <wlen> characters long");
print("Lenght is " + wlen);
```

Print ***—Lenght is 35***

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

matchEnd

> **Availability**
> This routine is available since **katl-commons 2.5.15 / 2.6.7**

Syntax:

matchEnd(input, regex)

Description:

Returns *the position where the match ends or -1 if it doesn't match.*

Parameters:

Parameter name	Type	Required	Description
input	string	Yes	Specifies a character expression to match the regex against.
regex	string	Yes	Specifies a regular expression to match the specified string.

Return type:

number

Example:

Example 1:

```
wret = matchEnd("This will return ?", ".*will");
print("Return " + wret);
```

*—Matches the string ending with **will** and will return 9(the position of l character).*

Notes:

1. For more information on regular expressions, see http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

matches

> **Availability**
> This routine is available since **katl-commons 2.0.3** (for JIRA 5.x) or **katl-commons 1.1.10** (for JIRA 4.3.x and 4.4.x) **.**

Syntax:

matches(string, regex)

Description:

Returns ***—true*** if character expression ***—string*** matches the regular expression ***—regex***.

Parameters:

Parameter name	Type	Required	Description

string	string	Yes	Specifies a character expression to match the regex against.
regex	string	Yes	Specifies a regular expression to match the specified string.

Return type:

boolean (true/false)

Example:

Example 1:

```
wret = matches("This will return ?", ".*will.*");
print("Return " + wret);
```

Matches any string containing "will". Prints ***—Return true—***

Example 2:

```
wret = matches("This will return ?", "will");
print("Return " + wret);
```

Matches only the string "will". Prints ***—Return false***

Example 3:

```
wret = matches("This will return ?", ".*will[^\\?]*\\?");
print("Return " + wret);
```

Matches any string containing "will" and ending with a question mark. Prints ***—Return true***

Notes:

1. As shown in Example 3, use double backslash (\\) instead of a single backslash where needed.
2. For more information on regular expressions, see http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

matchStart

Availability
This routine is available since **katl-commons 2.5.15 / 2.6.7**

Syntax:

matchStart(input, regex)

Description:

Returns *the position where the match starts or -1 if it doesn't match.*

Parameters:

Parameter name	Type	Required	Description
input	string	Yes	Specifies a character expression to match the regex against.
regex	string	Yes	Specifies a regular expression to match the specified string.

Return type:

number

Example:

Example 1:

```
wret = matchStart("This will return ?", "will.*");
print("Return " + wret);
```

*—Matches the string starting with **will** and will return 5(the position of w character).*

Notes:

1. For more information on regular expressions, see http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

matchText

> **Availability**
> This routine is available since **katl-commons 2.5.15 / 2.6.7**

Syntax:

matchText(input, regex)

Description:

Returns *the text matched or empty string if it doesn't match.*

Parameters:

Parameter name	Type	Required	Description
input	string	Yes	Specifies a character expression to match the regex against.
regex	string	Yes	Specifies a regular expression to match the specified string.

Return type:

string

Example:

Example 1:

```
wret = matchText("This will return ?", ".*will");
print("Return " + wret);
```

*—Returns the string matched(the beginning of the string until **will** word.*

Notes:

1. For more information on regular expressions, see http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

replace

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

replace(str, search_str, replacement_str)

Description:

Replaces the ***—search_str*** string with ***—replacement_str*** in ***—str*** and returns the resulting string. The string ***—str*** is not modified.

Parameters:

Parameter name	Type	Required	Description
str	string	Yes	Specifies the string for which to replace characters.
search_str	string	Yes	Specifies a character expression to search for in str.
replacement_str	string	Yes	Specifies the string that replaces search_str.

Return type:

string

Example:

Example 1:

```
wret = replace("foobar", "foo", "bar");
print(wret);
```

Print ***—barbar***

Example 2:

```
wret = replace("aaa", "aa", "b");
print(wret);
```

Print ***—ba***

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

split

> **Availability**
> This routine is available since **katl-commons 3.0 .**

Syntax:

split(str, regex)

Description:

Returns the array of strings computed by splitting this string around matches of the given regular expression.

Parameters:

Parameter name	Type	Required	Description
str	string	Yes	Specifies the string which will be split
regex	string	Yes	The delimiting regular expression

Return type:

string[]

Example:

Example 1:

```
return split("boo:and:foo", ":");
```

Returns ***— { "boo", "and", "foo" }.***

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

substring

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

substring(str, start, stop)

Description:

Returns substring is starting at index ***—start*** and stop at index ***—stop***.

The substring returned has length ***—stop - start***.
First position of character expression ***—str*** is 0.
If index ***—start*** is not initialized or -1, it will be set on 0. If index ***—stop*** is not initialized or -1, it will be set on end of the string.

Parameters:

Parameter name	Type	Required	Description
str	string	Yes	Specifies a character expression from which the character string is returned.
start	number	Yes	Specifies the position in the character expression —*str* from where the character string is returned.
stop	number	Yes	Specifies the position in the character expression —*str* to where the character string is returned.

Return type:

string

Example:

Example 1:

```
number start;
number stop;
substring("FooBar", start, stop);
// start will default to 0
// stop will default to 6
// substring() will return FooBar
// Result are the same if we initialize start and/or stop with -1
```

Example 2:

```
number start;
number stop = 3;
substring("FooBar", start, stop);
// start will default to 0
// will return Foo
```

Example 3:

```
string v = "ABCDEFGHIJKLMNOP...";
print(substring(v, 0, 3)); //"ABC"
print(substring(v, 10, 100)); //"KLMNOP..."
print(substring(v, 10, -1)); //"KLMNOP..." (same call)
```

Notes:

If ***—stop*** is equal ***—start*** or ***—start*** is greater than the maximum length of character expression ***—str***, ***—substring*** returns an empty string.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

toLower

Availability
This routine is available since **katl-commons 2.5** (for JIRA 5.x) **.**

Syntax:

toLower(string)

Description:

Returns the string only with **lower case letters**.

Returns the given string that has it's letters converted to lower case. If the string contains other characters than letters the chars that are not letters will not be converted.

Parameters:

Parameter name	Type	Required	Description
string	string	Yes	Specifies a character expression that will be converted to lower case.

Return type:

string

Example:

Example 1:

```
wret = toLower("FOoBaR");
print(wret);
```

Prints "—***foobar".***

Example 2:

```
wret = toLower("ABC2345f.ff");
print(wret);
```

Prints —***"abc2345f.ff—".***

toUpper

> **Availability**
> This routine is available since **katl-commons 2.5** (for JIRA 5.x) .

Syntax:

toUpper(string)

Description:

Returns the string only with **upper case letters**.

Returns the given string that has it's letters converted to **upper case**. If the string contains other characters than letters the chars that are not letters will not be converted.

Parameters:

Parameter name	Type	Required	Description

string	string	Yes	Specifies a character expression that will be converted to upper case.

Return type:

string

Example:

Example 1:

```
wret = toUpper("fOoBaR");
print(wret);
```

Prints "—***FOOBAR".***

Example 2:

```
wret = toUpper("AbC2w&345f.ff");
print(wret);
```

Prints —***"ABC2W&345F.FF—".***

trim

> **Availability**
> This routine is available since **katl-commons 2.0.2** (for JIRA 5.x) or **katl-commons 1.1.9** (for JIRA 1.1.9) .

Syntax:

trim(str)

Description:

Returns a trimmed copy of the string passed as parameter with the leading and trailing whitespaces removed.

Parameters:

Parameter name	Type	Required	Description
str	string	Yes	The string to trim.

Return type:

string

Example:

```
string original = "   foobar   ";
string trimmed = trim(original);
print(">>" + original + "<<"); // will print ">>   foobar   <<"
print(">>" + trimmed + "<<");  // will print ">>foobar<<"
```

Notes:

This also works if the string passed in as parameter only has leading or trailing whitespaces. If the string has no leading or trailing whitespaces, a copy of the original string will be returned.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

Date routines

Introduction

This section contains a collection of routines for handling date.

Routines summary

Routine	Description
currentDate	Returns the current date.
day	Returns a number representing the day of month. (1-31)
dayOfWeek	Returns the day of week, English only. One of the following string values: "Sun","Mon","Tue","Wed","Thu","Fri","Sat".
formatDate	Formats the given date into a date/time string accordingly to the given format expression.
hour	Returns the hour of the provided date (0-23).
minute	Returns the minutes of the provided date (0-59).
month	Returns a number representing the month of the provided date. (1-12)
monthName	Returns the month name of the provided date, English only. One of the following string values: "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov","Dec".
second	Returns the seconds of the provided date (0-59).
startOfDay	Returns a date, but strips off the hours, minutes and seconds.
toRawWorkingInterval	Returns a date interval converted to the specified number of hours per day.
week	Returns the week number in the year of the provided date
year	Returns the year of the provided date.
startOfMonth	Returns a date set on the first day of the month for the given date. Also sets hours/minutes/seconds to 0.
endOfMonth	Returns a date set on the last day of the month for the given date. Also sets hours/minutes/seconds/milliseconds to their respective maximum value.
addMonths	Adds a number of months to a specified date, preserving the day of month where possible.
toDate	Creates a date value from the specified parameters.
toTimeZone	Converts a date to the specified time zone.
parseDate	Returns the parsed date, according to the format you provided; if parse fails, it will return a null date

See Also:

Routines
Syntax
Variable Resolution

addMonths

> **Availability**
> This routine is available since **katl-commons 2.5.5**

Syntax:

addMonths(date, noMonths)

Description:

Adds a number of months to a specified date, preserving the day of month where possible.

Parameters:

Parameter name	Type	Required	Description
date	date	Yes	Specifies a Date expression
noMonths	number	Yes	Number of months to add. If given a negative number, will do substract.

Return type:

date

Example:

```
function addMonthsInDesc(date d, int noMonths){
    desc += "Date " + d + " + " + noMonths + " months = " + addMonths(d,
noMonths) + "\n";
}

desc = "";
addMonthsInDesc("2012-01-31", 1);
addMonthsInDesc("2012-04-30", -2);
addMonthsInDesc("2013-01-31", 1);
addMonthsInDesc("2013-04-30", -2);
addMonthsInDesc("2012-01-31", 12);
addMonthsInDesc("2012-01-31", -1);
```

Outputs to description:

Date 2012-01-31 00:00:00 + 1 months = 2012-02-29 00:00:00

Date 2012-04-30 00:00:00 + -2 months = 2012-02-29 00:00:00

Date 2013-01-31 00:00:00 + 1 months = 2013-02-28 00:00:00

Date 2013-04-30 00:00:00 + -2 months = 2013-02-28 00:00:00

Date 2012-01-31 00:00:00 + 12 months = 2013-01-31 00:00:00

Date 2012-01-31 00:00:00 + -1 months = 2011-12-31 00:00:00

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

currentDate

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

currentDate()

Description:

Returns the current date.

Return type:

date

Example:

```
dueDate = currentDate() + "3d";
```

Returns the 3rd day after the current day.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

day

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

day(date)

Description:

Returns a number representing the day of month. (1-31)

Parameters:

Parameter name	Type	Required	Description
date	date	Yes	Specifies a Date or DateTime expression

Return type:

number

Example:

Example 1:

```
print("Today is " + day(currentDate()));
```

Example 2:

```
date varDateTime = "2011-08-17T18:30:55";
print("Day is " + day(varDateTime));
```

Print ***—Day is 17***

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

dayOfWeek

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

dayOfWeek(date)

Description:

Returns the day of week, English only. One of the following string values: "Sun","Mon","Tue","Wed","Thu","Fri","Sat".

Parameters:

Parameter name	Type	Required	Description
date	date	Yes	Specifies a Date or DateTime expression

Return type:

string

Example:

Example 1:

```
print("Today is " + dayOfWeek(currentDate()));
```

Example 2:

```
date varDateTime = "2011-08-17T18:30:55";
print("Day of week is " + dayOfWeek(varDateTime));
```

Print ***—Day of week is Wed***

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

endOfMonth

Availability
This routine is available since **katl-commons 2.5.5**

Syntax:

endOfMonth(date)

Description:

Returns a date set on the last day of the month for the given date. Also sets hours/minutes/seconds/milliseconds to their respective maximum value.

Parameters:

Parameter name	Type	Required	Description
date	date	Yes	Specifies a Date expression

Return type:

date

Example:

```
function endOfMonthToDesc(date d){
    desc += "End of month for date " + d + " is " + endOfMonth(d) + "\n";
}

desc = "";
endOfMonthToDesc("2013-01-26 20:19:18");
endOfMonthToDesc("2013-02-01 20:19:18");
endOfMonthToDesc("2012-02-01 20:19:18");
endOfMonthToDesc("2013-02-28 20:19:18");
endOfMonthToDesc("2012-02-29 20:19:18");
endOfMonthToDesc("2012-12-31 23:59:59");
endOfMonthToDesc("2012-12-31 00:00:00");
endOfMonthToDesc("2012-12-01 23:59:59");
endOfMonthToDesc("2012-12-01 00:00:00");
```

Outputs to description:

End of month for date 2013-01-26 20:19:18 is 2013-01-31 23:59:59

End of month for date 2013-02-01 20:19:18 is 2013-02-28 23:59:59

End of month for date 2012-02-01 20:19:18 is 2012-02-29 23:59:59

End of month for date 2013-02-28 20:19:18 is 2013-02-28 23:59:59

End of month for date 2012-02-29 20:19:18 is 2012-02-29 23:59:59

End of month for date 2012-12-31 23:59:59 is 2012-12-31 23:59:59

End of month for date 2012-12-31 00:00:00 is 2012-12-31 23:59:59

End of month for date 2012-12-01 23:59:59 is 2012-12-31 23:59:59

End of month for date 2012-12-01 00:00:00 is 2012-12-31 23:59:59

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

formatDate

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

formatDate(date, format)

Description:

Formats the given date into a date/time string accordingly to the given format expression.

Parameters:

Parameter name	Type	Required	Description
date	date	Yes	Specifies a Date or DateTime expression
format	string	Yes	Specifies a pattern expression representing the desired date format

Return type:

string

The return value represents the formatted string representation for the given date.

Example:

Example 1:

```
string format = "yyyy.MM.dd G 'at' HH:mm:ss z";
print("Current time is " + formatDate(currentDate(), format));
```

Assuming that current date is 30.10.2011 and time 12:08, prints —*Today is `2011.10.30 AD at 12:08:00 PDT`*

Example 2:

```
date varDateTime = "2011-08-17T18:30:55";
string format = "EEE, d MMM yyyy HH:mm:ss Z";
print("Formatted date is " + formatDate(varDateTime, format));
```

Print —*Formatted date is `Wed, 17 Aug 2011 18:30:55 +0300`*

For a full set of date formats that can be used, check out the SimpleDateFormat documentation from Oracle.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

hour

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

hour(date)

Description:

Returns the hour of the provided date (0-23).

Parameters:

Parameter name	Type	Required	Description
date	date	Yes	Specifies a DateTime expression

Return type:

number

Example:

Example 1:

```
print("Hour is " + hour(currentDate()));
```

Example 2:

```
date varDateTime = "2011-08-17T18:30:55";
print("Hour is " + hour(varDateTime));
```

Print ***—Hour is 18***

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

millisToDate

> **Availability**
> This routine is available since **katl-commons 2.5.9**

Syntax:

millisToDate(millis)

Description:

Converts milliseconds to a date.

Parameters:

Parameter name	Type	Required	Description
millis	number	Yes	Specifies the milliseconds to convert.

Return type:

date

Example:

```
return millisToDate(1370616623112);
```

Outputs: *2013-06-07 17:50:23*

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

millisToInterval

Availability
This routine is available since **katl-commons 2.5.9**

Syntax:

millisToInterval(millis)

Description:

Converts milliseconds to a time interval.

Parameters:

Parameter name	Type	Required	Description
millis	number	Yes	Specifies the milliseconds to convert.

Return type:

interval

Example:

```
return millisToInterval(616623112);
```

Outputs: 1w 3h 17m 3s

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

minute

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

minute(date)

Description:

Returns the minutes of the provided date (0-59).

Parameters:

Parameter name	Type	Required	Description

date	date	Yes	Specifies a Date or a DateTime expression

Return type:

number

Example:

Example 1:

```
print("Minute is " + minute(currentDate()));
```

Example 2:

```
date varDateTime = "2011-08-17T18:30:55";
print("Minute is " + minute(varDateTime));
```

Print ***—Minute is 30***

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

month

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

month(date)

Description:

Returns a number representing the month of the provided date. (1-12)

Parameters:

Parameter name	Type	Required	Description
date	date	Yes	Specifies a Date or a DateTime expression

Return type:

number

Example:

Example 1:

```
print("Month is " + month(currentDate()));
```

Example 2:

```
date varDateTime = "2011-08-17T18:30:55";
print("Month is " + month(varDateTime));
```

Print ***—Month is 8***

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

monthName

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

monthName(date)

Description:

Returns the month name of the provided date, English only. One of the following string values: "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov","Dec".

Parameters:

Parameter name	Type	Required	Description
date	date	Yes	Specifies a Date or a DateTime expression

Return type:

string

Example:

Example 1:

```
print("Month is " + monthName(currentDate()));
```

Example 2:

```
date varDateTime = "2011-08-17T18:30:55";
print("Month is " + monthName(varDateTime));
```

Print ***—Month is Aug***

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

parseDate

Availability
This routine is available since **katl-commons 2.5.10 / 2.6.2 .**

Syntax:

parseDate(format, date_as_string)

Description:

Returns the parsed date, according to the format you provided; if parse fails, it will return a null date

Parameters:

Parameter name	Type	Required	Description
format	string	Yes	A valid Java format, as defined here: http://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html
date_as_string	string	Yes	The string which represents the date

Return type:

date

Example:

```
print("(1) Parsed date is:" + parseDate("yyyy.MM.dd", "2000.01.01"));
//returns a valid date
print("(2) Parsed date is:" + parseDate("yyyy.MM.dd", "2000/01/01"));
//returns an empty date (null)
```

second

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

second(date)

Description:

Returns the seconds of the provided date (0-59).

Parameters:

Parameter name	Type	Required	Description
date	date	Yes	Specifies a Date or a DateTime expression

Return type:

number

Example:

Example 1:

```
print("Second is " + second(currentDate()));
```

Example 2:

```
date varDateTime = "2011-08-17T18:30:55";
print("Second is " + second(varDateTime));
```

Print ***—Second is 55***

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

startOfDay

Availability
This routine is available since **katl-commons 1.0 .**

Syntax:

startOfDay(date)

Description:

Returns a date, but strips off the hours, minutes and seconds.

Parameters:

Parameter name	Type	Required	Description
date	date	Yes	Specifies a Date expression

Return type:

date

Example:

Example 1:

```
if(datepicker < startOfDay(currentDate())){
    return false;
}
```

Example 2:

```
date varDate = "2011-08-17";
print("Date is " + startOfDay(varDate));
```

Print ***—Date is 17/Aug/11***

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

startOfMonth

Availability

This routine is available since **katl-commons 2.5.5**

Syntax:

startOfMonth(date)

Description:

Returns a date set on the first day of the month for the given date. Also sets hours/minutes/seconds to 0.

Parameters:

Parameter name	Type	Required	Description
date	date	Yes	Specifies a Date expression

Return type:

date

Example:

```
function startOfMonthToDesc(date d){
    desc += "Start of month for date " + d + " is " + startOfMonth(d) +
"\n";
}

desc = "";
startOfMonthToDesc("2013-01-26 20:19:18");
startOfMonthToDesc("2013-02-01 20:19:18");
startOfMonthToDesc("2013-02-28 20:19:18");
startOfMonthToDesc("2012-02-29 20:19:18");
startOfMonthToDesc("2012-12-31 23:59:59");
startOfMonthToDesc("2012-12-01 23:59:59");
startOfMonthToDesc("2012-12-01 00:00:00");
```

Outputs to description:

Start of month for date 2013-01-26 20:19:18 is 2013-01-01 00:00:00

Start of month for date 2013-02-01 20:19:18 is 2013-02-01 00:00:00

Start of month for date 2013-02-28 20:19:18 is 2013-02-01 00:00:00

Start of month for date 2012-02-29 20:19:18 is 2012-02-01 00:00:00

Start of month for date 2012-12-31 23:59:59 is 2012-12-01 00:00:00

Start of month for date 2012-12-01 23:59:59 is 2012-12-01 00:00:00

Start of month for date 2012-12-01 00:00:00 is 2012-12-01 00:00:00

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

toDate

Availability
This routine is available since **katl-commons 2.5.5**

Syntax:

toDate(year, month, day[, hour, minute, second[, millis]])

Since katl-commons 2.5.6:

toDate(year, month, day[, hour, minute, second[, millis[, timeZone]]])

Description:

Creates a date value from the specified parameters.

Parameters:

Parameter name	Type	Required	Restrictions	Description
year	number	Yes	-	Specifies the year value
month	number	Yes	[1-12]	Specifies the month value. This value is 1-based (e.g 1 = January, 12 = December).
day	number	Yes	[1-28-29-30-31]	Specifies the day of month value. Providing an invalid day value for the specified month/year (e.g. 30 Feb 2013, 29 Feb 2013) will throw an exception **at runtime**.
hour	number	No	[0-23]	Specifies the hour of day value
minute	number	No	[0-59]	Specifies the minutes value
second	number	No	[0-59]	Specifies the number of seconds
millis	number	No	[0-999]	Specifies the number of milliseconds
timeZone	string	No	-	Specifies the time zone using a format compatible with TimeZone. See TimeZone for more details.

Wrong parameters
If any of the parameters are outside of the specified interval (e.g attempting to create 32 January), the routine will return an empty date, such that isNull(date) is true.

Note
The created date will use the locale of the JVM.

Return type:

date

Example:

```
function toDate7Descr(number year, number month, number day, number hour,
number minute, number second, string timeZone) {
  string format = "dd-MMM-yyyy HH:mm:ss Z";
  desc += "Converted year " + year + ", month " + month + ", day " + day +
", hour " + hour + ", minute " + minute + ", second " + second + ",
timeZone " + timeZone;
  desc += " to date " + formatDate(toDate(year, month, day, hour, minute,
second, 0, timeZone), format) + "\n";
}

function toDate6Descr(number year, number month, number day, number hour,
number minute, number second){
 desc += "Converted year " + year + ", month " + month + ", day " + day +
", hour " + hour + ", minute " + minute + ", second " + second;
 desc += " to date " + toDate(year, month, day, hour, minute, second) +
"\n";
}

function toDate3Descr(number year, number month, number day){
 desc += "Converted year " + year + ", month " + month + ", day " + day;
 desc += " to date " + toDate(year, month, day) + "\n";
}

desc = "";
toDate3Descr(2013, 12, 31);
toDate3Descr(2013, 2, 28);
toDate3Descr(2012, 2, 29);
toDate3Descr(2013, 2, 29);

toDate6Descr(2013, 12, 31, 23, 59, 59);
toDate6Descr(2013, 2, 28, 0, 0, 0);
toDate6Descr(2012, 2, 29, 14, 15, 16);
toDate6Descr(2013, 1, 1, 24, 1, 1);


toDate7Descr(2012, 01, 20, 0, 0, 0, "GMT+2");
```

Outputs to description:

Converted year 2013, month 12, day 31 to date 2013-12-31 00:00:00

Converted year 2013, month 2, day 28 to date 2013-02-28 00:00:00

Converted year 2012, month 2, day 29 to date 2012-02-29 00:00:00

Converted year 2013, month 2, day 29 to date

Converted year 2013, month 12, day 31, hour 23, minute 59, second 59 to date 2013-12-31 23:59:59

Converted year 2013, month 2, day 28, hour 0, minute 0, second 0 to date 2013-02-28 00:00:00

Converted year 2012, month 2, day 29, hour 14, minute 15, second 16 to date 2012-02-29 14:15:16

Converted year 2013, month 1, day 1, hour 24, minute 1, second 1 to date

Converted year 2012, month 1, day 20, hour 0, minute 0, second 0, timeZone GMT+2 to date 20-Jan-2012 00:00:00 +0200

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

toRawWorkingInterval

> **Availability**
> This routine is available since **katl-commons 2.5.**

Syntax:

toRawWorkingInterval (interval, [workingHours/day]);

Description:

Returns a date interval converted to the specified number of hours per day.

This routine re-writes a 24h day interval in an interval which is based on 8h (for instance). It is raw, in the sense that it does NOT care for weekends, holidays, leave without pay etc.

The result interval is converted using the workingHours/day parameter that is optional and is an integer between 1 and 24. This routine can be used to view how much hours can a user work during a week (for instance).

Parameters:

Parameter name	Type	Required	Description
interval	interval	Yes	Specifies a date interval that will be converted from 24 hours/day to working hours/day.
format	number	No	Specifies the number of working hours per day the interval will be converted to. The default number is 8, if another number isn't specified.

Return type:

interval

The return value represents the converted interval from the interval given as argument.

Example:

Example 1:.

Let us consider the following SIL code:

```
date begin = "2013-01-17T12:30:00";
date end = "2013-01-28T16:30:00";
return toRawWorkingInterval(end - begin, 6);
```

This will return: "**2d 22h**" because:

1) The interval between begin date and end date is 11d 4h.

2) If we convert a day from **24 hours** to a **6 hour** per day we will make the following computation:

11d * 6h + 4h = 66h + 4h = 70h that will be viewed as **2d 22h**.

Example 2:

By running this code:

```
date begin = "2013-01-17T02:30:00";
date end = "2013-01-28T16:30:00";
return toRawWorkingInterval(end - begin);
```

The result of it will be: "**4d**" because:

1) The interval is "**11d 12h**" long.

2) The conversion is based on this computation:

12d * 8h(default hours) = 96h that means "**4d**"

3) This is done so because the number of hours in the 12th day of the interval exceeds the number of working hours per day (by default 8) and a new day is added.

toTimeZone

> **Availability**
> This routine is available since **katl-commons 2.5.6**

Syntax:

toTimeZone(date, timeZone)

Description:

Converts a date to the specified time zone.

Parameters:

Parameter name	Type	Required	Description
date	date	Yes	Specifies the date to convert.
timeZone	string	No	Specifies the time zone using a format compatible with TimeZone. See TimeZone for more details.

Return type:

date

Example:

```
desc = "";
date d = toDate(2012, 01, 20, 0, 0, 0, 0, "GMT-8");
string format = "dd-MMM-yyyy HH:mm:ss Z";

desc += formatDate(d, format) + " converted to " + formatDate(toTimeZone(d,
"GMT+2"), format);
```

Outputs to description:

20-Jan-2012 00:00:00 -0800 converted to 20-Jan-2012 10:00:00 +0200

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

week

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

week(date)

Description:

Returns the week number in the year of the provided date

Parameters:

Parameter name	Type	Required	Description
date	date	Yes	Specifies a Date or a DateTime expression

Return type:

number

Example:

Example 1:

```
print("Week is " + week(currentDate()));
```

Example 2:

```
date varDateTime = "2011-08-17T18:30:55";
print("Week is " + week(varDateTime));
```

Print ***—Week is 33***

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

year

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

year(date)

Description:

Returns the year of the provided date.

Parameters:

Parameter name	Type	Required	Description

date	date	Yes	Specifies a Date or a DateTime expression

Return type:

number

Example:

Example 1:

```
print("Year is " + year(currentDate()));
```

Example 2:

```
date varDateTime = "2011-08-17T18:30:55";
print("Year is " + year(varDateTime));
```

Print ***—Year is 2011***

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

Worklog Routines

Introduction

This sections contains a collection of worklog related routines.

> When upgrading to JJupin 2.5 and katl-commons 2.5 you have to remove the worklog plugin from your JIRA plugins directory. The routines from this plugins were added to katl-commons 2.5 .

Routines summary

Routine	Description
addWorklogAdjustEstimate	Adds worklog and the Remaining will be reduced by the amount of work done, but never below 0
addWorklogExistingEstimate	Adds worklog and keeps the same Remaining
addWorklogSetEstimateTo	Adds worklog and sets Remaining to the value of setTo
addWorklogReduceEstimateBy	Adds worklog and the Remaining will be reduced by the value of reduceBY, but never below 0
updateWorklogAdjustEstimate	Updates the worklog and the Remaining will be adjusted with the value of interval, but never below 0
updateWorklogExistingEstimate	Updates the worklog and keeps the same Remaining
updateWorklogSetEstimateTo	Updates the worklog and sets Remaining to the value of setTo
removeWorklogAdjustEstimate	Removes the worklog and the Remaining will be increased by the time from the worklog
removeWorklogExistingEstimate	Removes the worklog and keeps the same Remaining
removeWorklogIncreaseEstimateBy	Removes the worklog and the Remaining will be increased by the value of increaseBy
removeWorklogSetEstimateTo	Removes the worklog and sets Remaining to the value of setTo
getWorkingInterval	Returns the number of working hours from a time interval

getWorklogLoggedHours	Returns the logged hours for a worklog
getWorklogStartDate	Returns the start date of a worklog
getWorklogCreatedDate	Returns the created date of a worklog
getWorklogAuthor	Returns the author of a worklog
getWorklogUpdateAuthor	Returns the author who last updated the worklog
getWorklogUpdatedDate	Returns the last date when the worklog was updated
getWorklogComment	Returns the comment associated with the worklog
getWorklogIds	Returns an array with the ids of the worklogs for the specified date and issue
getWorklogIdsForUser	Returns an array with the ids of the worklogs for the specified user and issue.
getWorklogsForIssues	Returns an array with the worklogs for the specified list of issues, and if exists for given user also.
getWorkingDaysPerWeek	Returns the number of working days per week.
getWorkingHoursPerDay	Returns the number of working hours per day.

See also

See also
Routines
Syntax
Variable Resolution

addWorklogAdjustEstimate

Availability
This routine is available since **katl-commons 2.5 .**

Syntax:

addWorklogAdjustEstimate(issue, user, interval, startDate, comment)

Description:

Adds worklog and the Remaining will be reduced by the amount of work done, but never below 0.

Parameters:

Parameter	Type	Required	Description
issue	String	Yes	the key of the selected issue
user	String	Yes	the user name of the selected user
interval	Interval	Yes	the interval of time worked
startDate	Date	Yes	start working date
comment	String	Yes	the comment that will be post on the worklog

Return type:

None

Example:

```
addWorklogAdjustEstimate(key, currentUser(), "2h", currentDate(), "test worklog");
```

Adds an worklog of 2 hours for the current user on the current issue, recalculating the remaining time automatically.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

addWorklogExistingEstimate

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

addWorklogExistingEstimate(issue, user, interval, startDate, comment)

Description:

Adds worklog and keeps the same Remaining.

Parameters:

Parameter	Type	Required	Description
issue	String	Yes	the key of the selected issue
user	String	Yes	the user name of the selected user
interval	Interval	Yes	the interval of time worked
startDate	Date	Yes	start working date
comment	String	Yes	the comment that will be post on the worklog

Return type:

None

Example:

```
addWorklogExistingEstimate(key, currentUser(), "2h", currentDate(), "test worklog");
```

Adds an worklog of 2 hours for the current user on the current issue, keeping the old remaining time.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

addWorklogReduceEstimateBy

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

addWorklogReduceEstimateBy(issue, user, interval, startDate, comment, reduceBY)

Description:

Adds worklog and the Remaining will be reduced by the value of reduceBY, but never below 0.

Parameters:

Parameter	Type	Required	Description
issue	String	Yes	the key of the selected issue
user	String	Yes	the user name of the selected user
interval	Interval	Yes	the interval of time worked
startDate	Date	Yes	start working date
comment	String	Yes	the comment that will be post on the worklog
reduceBY	Interval	Yes	the interval to reduce Remaining by

Return type:

None

Example:

```
addWorklogReduceEstimateBy(key, currentUser(), "2h", currentDate(), "test worklog", "4h");
```

Adds an worklog of 2 hours for the current user on the current issue, reducing the remaining time by 4 hours.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

addWorklogSetEstimateTo

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

addWorklogSetEstimateTo(issue, user, interval, startDate, comment, setTo)

Description:

Adds worklog and sets Remaining to the value of setTo.

Parameters:

Parameter	Type	Required	Description
issue	String	Yes	the key of the selected issue
user	String	Yes	the user name of the selected user
interval	Interval	Yes	the interval of time worked
startDate	Date	Yes	start working date
comment	String	Yes	the comment that will be post on the worklog

setTo	Interval	Yes	the interval to set the Remaining with

Return type:

None

Example:

```
addWorklogSetEstimateTo(key, currentUser(), "2h", currentDate(), "test
worklog", "8h");
```

Adds an worklog of 2 hours for the current user on the current issue, setting the remaining time to 8 hours.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

updateWorklogAdjustEstimate

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

updateWorklogAdjustEstimate(worklog, interval, startDate, comment)

Description:

Updates the worklog and the Remaining will be adjusted with the value of interval, but never below 0.

Parameters:

Parameter	Type	Required	Description
worklog	Number	Yes	the id of the selected worklog
interval	Interval	Yes	the interval of time worked
startDate	Date	Yes	start working date
comment	String	Yes	the comment that will be post on the worklog

Return type:

None

Example:

```
updateWorklogAdjustEstimate(10000, "2h", currentDate(), "test worklog");
```

Updates the worklog with id = 10000, setting the worked interval to 2 hours and recalculating the remaining time automatically.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

updateWorklogExistingEstimate

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

updateWorklogExistingEstimate(worklog, interval, startDate, comment)

Description:

Updates the worklog and keeps the same Remaining.

Parameters:

Parameter	Type	Required	Description
worklog	Number	Yes	the id of the selected worklog
interval	Interval	Yes	the interval of time worked
startDate	Date	Yes	start working date
comment	String	Yes	the comment that will be post on the worklog

Return type:

None

Example:

```
updateWorklogExistingEstimate(10000, "2h", currentDate(), "test worklog");
```

Updates the worklog with id = 10000, setting the worked interval to 2 hours and keeping the old remaining time.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

updateWorklogSetEstimateTo

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

updateWorklogSetEstimateTo(worklog, interval, startDate, comment, setTo)

Description:

Updates the worklog and keeps the same Remaining.

Parameters:

Parameter	Type	Required	Description
worklog	Number	Yes	the id of the selected worklog

interval	Interval	Yes	the interval of time worked
startDate	Date	Yes	start working date
comment	String	Yes	the comment that will be post on the worklog
setTo	Interval	Yes	the interval to set the Remaining with

Return type:

None

Example:

```
updateWorklogSetEstimateTo(10000, "2h", currentDate(), "test worklog",
"8h");
```

Updates the worklog with id = 10000, setting the worked interval to 2 hours and setting the remaining time to 8 hours.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

removeWorklogAdjustEstimate

Availability
This routine is available since **katl-commons 2.5 .**

Syntax:

removeWorklogAdjustEstimate(worklog)

Description:

Removes the worklog and the Remaining will be increased by the time from the worklog.

Parameters:

Parameter	Type	Required	Description
worklog	Number	Yes	the id of the selected worklog

Return type:

None

Example:

```
removeWorklogAdjustEstimate(10000);
```

Removes the worklog with id = 10000, recalculating the remaining time automatically.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

removeWorklogExistingEstimate

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

removeWorklogExistingEstimate(worklog)

Description:

Removes the worklog and keeps the same Remaining.

Parameters:

Parameter	Type	Required	Description
worklog	Number	Yes	the id of the selected worklog

Return type:

None

Example:

```
removeWorklogExistingEstimate(10000);
```

Removes the worklog with id = 10000, keeping the old remaining time.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

removeWorklogIncreaseEstimateBy

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

removeWorklogIncreaseEstimateBy(worklog, increaseBy)

Description:

Removes the worklog and the Remaining will be increased by the value of increaseBy.

Parameters:

Parameter	Type	Required	Description
worklog	Number	Yes	the id of the selected worklog

increaseBy	Interval	Yes	the interval to increase Remaining by

Return type:

None

Example:

```
removeWorklogIncreaseEstimateBy(10000, "4h");
```

Removes the worklog with id = 10000, increasing the remaining time by 4 hours.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

removeWorklogSetEstimateTo

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

removeWorklogSetEstimateTo(worklog, increaseBy)

Description:

Removes the worklog and sets Remaining to the value of setTo.

Parameters:

Parameter	Type	Required	Description
worklog	Number	Yes	the id of the selected worklog
setTo	Interval	Yes	the interval to set the Remaining with

Return type:

None

Example:

```
removeWorklogSetEstimateTo(10000, "8h");
```

Removes the worklog with id = 10000, setting the remaining time to 8 hours.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getWorkingInterval

Availability
This routine is available since **katl-commons 2.5 .**

Syntax:

getWorkingInterval(startDate, endDate, startWorkingHour, endWorkingHour, weekendDays, holidays)

Description:

Returns the number of working hours from a time interval.

Parameters:

Parameter	Type	Required	Description
startDate	Date	Yes	the start working date
endDate	Date	Yes	the end working date
startWorkingHour	String	Yes	the daily start working hour (HH:mm)
endWorkingHour	String	Yes	the daily end working hour (HH:mm)
weekendDays	Number array	Yes	the weekend days (day of week)
holidays	Date array	Yes	the dates of the free days

Return type:

number

The returned number represents the working hours from the given time interval.

Example:

```
number hours;
date startDate = "2012-01-10 12:00:00";
date endDate = "2012-01-20 10:30:00";
string startHour = "09:00";
string endHour = "17:00";
number[] weekend = {7, 1};
date[] holidays = {"2012-01-18", "2012-01-16"};

hours = getWorkingInterval(startDate, endDate, startHour, endHour, weekend,
holidays);
```

Returns the number of working hours from 2012-01-10 12:00 to 2012-01-20 10:30, excluding the weekends (7 is Saturday and 1 is Sunday) and the holidays (2012-01-18 and 2012-01-16).

getWorklogAuthor

Availability
This routine is available since **katl-commons 2.5 .**

Syntax:

getWorklogAuthor(worklog)

Description:

Returns the author of a worklog.

Parameters:

Parameter	Type	Required	Description
worklog	Number	Yes	the id of the selected worklog

Return type:

String

Example 1:

```
print(getWorklogAuthor(11201))
```

Prints author's username of the worklog with id = 11201.

Example 2:

```
for(number id in getWorklogIds("2012-05-23 16:01:00",key)){
 print(getWorklogAuthor(id))
}
```

Prints the author for the worklogs of the current issue who have the start date = "2012-05-23 16:01:00".

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getWorklogComment

Availability
This routine is available since **katl-commons 2.5 .**

Syntax:

getWorklogComment(worklog)

Description:

Returns the comment associated with the worklog.

Parameters:

Parameter	Type	Required	Description
worklog	Number	Yes	the id of the selected worklog

Return type:

String

Example 1:

```
print(getWorklogComment(11201))
```

Prints the comment associated with the specified worklog (the worklog with id = 11201).

Example 2:

```
for(number id in getWorklogIdsForUser("admin", key)){
 print(getWorklogComment(id))
}
```

Prints the comment associated with the admin's worklogs of the current issue.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getWorklogCreatedDate

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

getWorklogCreatedDate(worklog)

Description:

Returns the created date of a worklog.

Parameters:

Parameter	Type	Required	Description
worklog	Number	Yes	the id of the selected worklog

Return type:

Date

Example 1:

```
print(getWorklogCreatedDate(11201))
```

Prints the created date of the worklog with id = 11201.

Example 2:

```
for(number id in getWorklogIdsForUser("admin", "TP-9")){
 print(getWorklogCreatedDate(id))
}
```

Prints the created date for all the admin's worklogs of the issue with key = "TP-9".

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getWorklogIds

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

getWorklogIds(startDate, issueKey)

Description:

Returns an array with the ids of the worklogs for the specified date and issue.

Parameters:

Parameter	Type	Required	Description
startDate	Date	Yes	the start working date
issueKey	String	Yes	the key of the selected issue

Return type:

Number []

Example 1:

```
getWorklogIds("2012-05-23 16:01:00",key)
```

Returns an array containing the ids of all the worklogs of the current issue who have the start date = "2012-05-23 16:01:00".

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getWorklogIdsForUser

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

getWorklogIdsForUser(author, issueKey)

Description:

Returns an array with the ids of the worklogs for the specified user and issue.

Parameters:

Parameter	Type	Required	Description
author	String	Yes	the username of the selected user
issueKey	String	Yes	the key of the selected issue

Return type:

Number []

Example 1:

```
getWorklogIdsForUser("admin",key)
```

Returns an array containing the ids of all the worklogs of the current issue who were created by admin.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getWorklogLoggedHours

Availability
This routine is available since **katl-commons 2.5 .**

Syntax:

getWorklogLoggedHours(worklog)

Description:

Returns the logged hours for a worklog.

Parameters:

Parameter	Type	Required	Description
worklog	Number	Yes	the id of the selected worklog

Return type:

Interval

Example 1:

```
print(getWorklogLoggedHours(11201))
```

Prints the logged hours for the worklog with id = 11201.

Example 2:

```
for(number id in getWorklogIdsForUser("admin", "TP-9")){
 print(getWorklogLoggedHours(id))
}
```

Prints the logged hours for all the admin's worklogs of the issue with key = "TP-9".

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getWorklogStartDate

Availability
This routine is available since **katl-commons 2.5 .**

Syntax:

getWorklogStartDate(worklog)

Description:

Returns the start date of a worklog.

Parameters:

Parameter	Type	Required	Description
worklog	Number	Yes	the id of the selected worklog

Return type:

Date

Example 1:

```
print(getWorklogStartDate(11201))
```

Prints the start date of the worklog with id = 11201.

Example 2:

```
for(number id in getWorklogIdsForUser("admin", "TP-9")){
 print(getWorklogStartDate(id))
}
```

Prints the start date for all the admin's worklogs of the issue with key = "TP-9".

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getWorklogUpdateAuthor

Availability
This routine is available since **katl-commons 2.5 .**

Syntax

getWorklogUpdateAuthor(worklog)

Description:

Returns the author who last updated the worklog.

Parameters:

Parameter	Type	Required	Description
worklog	Number	Yes	the id of the selected worklog

Return type:

String

Example 1:

```
print(getWorklogUpdateAuthor(11201))
```

Prints the username of the author who last updated the worklog with id = 11201.

Example 2:

```
for(number id in getWorklogIds("2012-05-23 16:01:00",key)){
 print(getWorklogUpdateAuthor(id))
}
```

Prints the author who last updated the worklogs of the current issue who have the start date = "2012-05-23 16:01:00".

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getWorklogUpdatedDate

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

getWorklogUpdatedDate(worklog)

Description:

Returns the last date when the worklog was updated.

Parameters:

Parameter	Type	Required	Description
worklog	Number	Yes	the id of the selected worklog

Return type:

Date

Example 1:

```
print(getWorklogUpdatedDate(11201))
```

Prints the last date when the worklog with id = 11201 was updated.

Example 2:

```
for(number id in getWorklogIds("admin",key)){
 print(getWorklogUpdatedDate(id))
}
```

Prints the last date when the admin's worklogs of the current issue were updated.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getWorkingHoursPerDay

> **Availability**
> This routine is available since **katl-commons 3.0.0 .**

Syntax:

getWorkingHoursPerDay()

Description:

Gets the number of working hours per day as configured in JIRA.

Parameters:

This routine has no parameters.

Return type:

Number

Returns the number of working hours per day.

Example:

```
return getWorkingHoursPerDay();
```

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getWorkingDaysPerWeek

> **Availability**
> This routine is available since **katl-commons 3.0.0.**

Syntax:

getWorkingDaysPerWeek()

Description:

Gets the number of working days per week as configured in JIRA.

Parameters:

This routine has no parameters.

Return type:

Number

Returns the number of working days per week.

Example:

```
return getWorkingDaysPerWeek();
```

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getWorklogsForIssues

> **Availability**
> This routine is available since **katl-commons 3.0.2 .**

Syntax:

getWorklogsForIssues(startDate, endDate, issues[, user])

Description:

Returns an array with the worklogs for the specified list of issues, and if exists for given user also.

Parameters:

Parameter	Type	Required	Description
startDate	Date	Yes	The start date of the interval where the startDate of the worklog can be.
endDate	Date	Yes	The end date of the interval where the startDate of the worklog can be.
issues	String[]	Yes	The list of the issues from which can be made the filter
user	String	No	The user from whom can be made the filter.

Return type:

JWorklog []

Example 1:

```
return getWorklogsForIssues("2014-11-26", "2014-12-03",
{"DEMO-6","LFTP-1","TSTAG-4"},"user");
```

Returns an array containing the JWorklog strucures of the list of the issues which have the startDate between start date = "2014-11-26"(inclusive) and end date = "2014-12-03"(exclusive), for given user.

Example 2:

```
return getWorklogsForIssues("2014-11-26", "2014-12-03",
{"DEMO-6","LFTP-1","TSTAG-4"});
```

Returns an array containing the JWorklog strucures of the list of the issues which have the startDate between start date = "2014-11-26"(inclusive) and end date = "2014-12-03"(exclusive).

The filter of the worklogs is made only by the issues.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

Math Routines

Availability
Routines from this section are available since **katl-commons 2.5 .**

Math Library

Mathlib plugin is a library that contains math routines for SIL.

Compatibility

Product	JJupin
Version 2.5	Version 2.5.x

Routines

Here is the list of available routines:

Routine	Description	Syntax
abs	Returns the absolute value of a number.	abs(number)

ceiling	Returns number rounded up, away from zero to the nearest multiple of significance.	ceiling(number, significance)
cos	Returns the cosine of the given angle.	cos(number)
degrees	Converts radians into degrees.	degrees(radians)
e	Returns the value of e.	e()
exp	Returns e raised to the power of a given number.	exp(number)
fact	Returns the factorial of a number.	fact(number)
floor	Rounds number down, toward zero, to the nearest multiple of significance.	floor(number, significance)
formatNumber	Formats the number according to a format.	formatNumber(number, format)
ln	Returns the natural logarithm of a number.	ln(number)
log	Returns the logarithm of a number to a specified base.	log(number, base)
pi	Returns the value of Pi.	pi()
power	Returns the result of a number raised to a power.	power(base, exponent)
radians	Converts degrees to radians.	radians(degrees)
rand	Return a random number between 0.00 and 1.00.	rand()
random	Return a random int value between 0 (inclusive) and the specified value (exclusive).	random(number)
roman	Converts an arabic numeral to roman, as text.	roman(arabic number)
round	Rounds a number to a specified number of digits.	round(number, digits)
sign	Returns the sign of a number.	sign(number)
sin	Returns the sine of a given angle.	sin(number)
sqrt	Returns a positive square root.	sqrt(number)
tan	Returns the tangent of a number.	tan(number)
trunc	Truncates a number to a specified precision.	trunc(number, digits)

abs

Availability

This routine is available since **katl-commons 2.5 .**

Syntax:

abs(number)

Description:

Returns the absolute value of a number. The absolute value of a number is the number without its sign.

Parameters:

Parameter	Type	Required	Description
number	Number	Yes	the number to get the absolute value for

Return type:

number

Example:

```
number a = abs(4);
number b = abs(-4);
print("a= " + a);
print("b= " + b);
```

Print: a= 4;

b= 4;

ceiling

Availability
This routine is available since **katl-commons 2.5 .**

Syntax:

ceiling(number, significance)

Description:

Returns number rounded up, away from zero, to the nearest multiple of significance.

Parameters:

Parameter	Type	Required	Description
number	Number	Yes	The numeric value you want to round.
significance	Number	Yes	The multiple to which you want to round.

Return type:

number

Example:

```
number a = floor(2.5, 1);
print("a= " + a);
number b = floor(-2.5, -2);
print("b= " + b);
```

Print: a= 3;

b= -4;

Notes:

- If number is positive and significance is negative, ceiling returns the NaN value.

cos

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

cos(number)

Description:

Returns the cosine of the given angle. If the angle is in degrees, either multiply the angle by PI()/180 or use the **radians** function to convert the angle to radians.

Parameters:

Parameter	Type	Required	Description
number	Number	Yes	The angle in radians for which you want the cosine.

Return type:

number

Example:

```
number a = cos(1.047);
print("a= " + a);

number b = cos(60*pi()/180);
print("b= " + b);

number c = cos(radians(30));
print("c= " + c);
```

Print: a= 0.5001710745970701 ;

b= 0.5000000000000001;

c= 0.5000000000000001;

degrees

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

degrees(radians)

Description:

Converts radians into degrees.

Parameters:

Parameter	Type	Required	Description
radians	Number	Yes	The angle in radians that you want to convert.

Return type:

number

Example:

```
number a = degrees(pi());
print("a= " + a);
```

Print: a= 180;

e number

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

e()

Description:

Returns the value of e.

Return type:

number

Example:

```
number a = e();
print("a= " + a);
```

Print: a= 2.718281828459045;

exp

Availability
This routine is available since **katl-commons 2.5 .**

Syntax:

exp(number)

Description:

Returns e raised to the power of number. The constant e equals 2.71828182845904, the base of the natural logarithm.

Parameters:

Parameter	Type	Required	Description
number	Number	Yes	The exponent applied to the base e.

Return type:

number

Example:

```
number a = exp(1);
print("a= " + a);

number b = exp(2);
print("b= " + b);

number c = exp(-2);
print("c= " + c);

number d = exp(0);
print("d= " + d);
```

Print: a= 2.71828182845904 ;

b= 7.38905609893065 ;

c= 0.1353352832366127;

d= 1;

fact

Availability
This routine is available since **katl-commons 2.5 .**

Syntax:

fact(number)

Description:

Returns the factorial of a number. The factorial of a number is equal to 1*2*3*...* number.

Parameters:

Parameter	Type	Required	Description
number	Number	Yes	The non-negative number for which you want the factorial. If number is not an integer, it is truncated.

Return type:

number

Example:

```
number a = fact(5);
print("a= " + a);

number b = fact(3.7);
print("b= " + b);

number c = fact(-3);
print("c= " + c);
```

Print: a= 120 ;

b= 6;

c= NaN;

floor

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

floor(number, significance)

Description:

Rounds number down, toward zero, to the nearest multiple of significance.

Parameters:

Parameter	Type	Required	Description
number	Number	Yes	The numeric value you want to round.
significance	Number	Yes	The multiple to which you want to round.

Return type:

number

Example:

```
number a = floor(4.5, 2);
print("a= " + a);
number b = floor(-2.5, -2);
print("b= " + b);
```

Print: a= 4;

b= -2;

Notes:

- If number is positive and significance is negative, FLOOR returns the NaN value.

formatNumber

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

formatNumber(number, format)

Description:

Formats a number according to a format.

Parameters:

Parameter	Type	Required	Description
number	Number	Yes	The number to be formatted.
format	String	Yes	The string that specifies the format.

Return type:

String, represents the formatted number.

Example:

```
string a = formatNumber(123456.789 ,"###,###.###");
//The pound sign (#) denotes a digit, the comma is a placeholder for the
grouping separator, and the period is a placeholder for the decimal
separator.
print("a= " + a);

string b = formatNumber(12345.67, "$###,###.###");
print("b= " + b);

string c = formatNumber(123.78, "000000.000");
//The format specifies leading and trailing zeros, because the 0 character
is used instead of the pound sign (#).
print("c= " + c);

string d = formatNumber(123456.789, "###.##");
// The number has three digits to the right of the decimal point, but the
format has only two. The format method handles this by rounding up.
print("d=" + d);
```

Print: a= 123,456.789

b = $12,345.67

c= 000123.780

d=123456.79

The symbols are described in the following table:

Symbol	Description
0	a digit
#	a digit, zero shows as absent
.	placeholder for decimal separator
,	placeholder for grouping separator
E	separates mantissa and exponent for exponential formats
;	separates formats
-	default negative prefix
%	multiply by 100 and show as percentage
?	multiply by 1000 and show as per mille
$	currency sign; replaced by currency symbol; if doubled, replaced by international currency symbol; if present in a pattern, the monetary decimal separator is used instead of the decimal separator
X	any other characters can be used in the prefix or suffix
'	used to quote special characters in a prefix or suffix

ln

Availability
This routine is available since **katl-commons 2.5 .**

Syntax:

ln(number)

Description:

Returns the natural logarithm of a number. Natural logarithms are based on the constant e (2.71828182845904).

Parameters:

Parameter	Type	Required	Description
number	Number	Yes	The positive real number for which you want the natural logarithm.

Return type:

number

Example:

```
number a = ln(86);
print("a= " + a);
number b = ln(exp(4));
print("b= " + b);
```

Print: a= 4.454347;

b= 4;

log

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

log(number, base)

Description:

Returns the logarithm of a number to the base you specify.

Parameters:

Parameter	Type	Required	Description
number	Number	Yes	The positive real number for which you want the logarithm.
base	Number	Yes	The base of the logarithm.

Return type:

number

Example:

```
number a = log(100, 10);
print("a= " + a);
number b = log(8, 2);
print("b= " + b);
```

Print: a= 2;

b= 3;

pi

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

pi()

Description:

Returns the value of Pi.

Return type:

number

Example:

```
number a = pi();
print("a= " + a);
```

Print: a= 3.141592653589793;

power

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

power(base, exponent)

Description:

Returns the result of a number raised to a power.

Parameters:

Parameter	Type	Required	Description
number	Number	Yes	The base number.
base	Number	Yes	The exponent to which the base number is raised.

Return type:

number

Example:

```
number a = power(2, 3);
print("a= " + a);
```

Print: a= 8;

radians

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

radians(degrees)

Description:

Converts degrees into radians.

Parameters:

Parameter	Type	Required	Description
degrees	Number	Yes	An angle in degrees that you want to convert.

Return type:

number

Example:

```
number a = radians(180);
print("a= " + a);
```

Print: a= 3.14159;

rand

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

rand()

Description:

Return a random number between 0.00 and 1.00.

Return type:

number

Example:

```
number a = rand();
print("a= " + a);
```

Print: a= 0.2655790724320535;

random

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

random(number)

Description:

Return a random int value between 0 (inclusive) and the specified value (exclusive).

Parameters:

Parameter	Type	Required	Description
number	Number	Yes	the number to get the random int value for

Return type:

number

Example:

```
number a = random(4);
print("a= " + a);
```

Print: a= 3;

roman

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

roman(arabic number)

Description:

Converts an arabic numeral to roman, as text.

Parameters:

Parameter	Type	Required	Description
arabic number	Number	Yes	The arabic numeral you want converted

Return type:

string

Example:

```
string a = roman(7);
print("a= " + a);
string b = roman(2034);
print("b= " + b);
```

Print: a= VII;

b= MMXXXIV;

Notes:

- If number is less than 0 and greater than 3999 is returned an empty string.

round

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

round(number, digits)

Description:

Rounds a number to a specified number of digits.

Parameters:

Parameter	Type	Required	Description
number	Number	Yes	The number that you want to round.
digits	Number	Yes	The number of digits to which you want to round the number argument.

Return type:

number

Example:

```
number a = round(2.36547, 2);
print("a= " + a);
```

Print: a= 2.37;

sign

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

sign(number)

Description:

Determines the sign of a number. Returns 1 if the number is positive, zero (0) if the number is 0, and -1 if the number is negative.

Parameters:

Parameter	Type	Required	Description
number	Number	Yes	Any real number.

Return type:

number

Example:

```
number a = sign(3);
print("a= " + a);
number b = sign(-4);
print("b= " + b);
number c = sign(3-3);
print("b= " + c);
```

Print: a= 1;

b= -1;

c= 0;

sin

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

sin(number)

Description:

Returns the sine of the given angle. If the angle is in degrees, either multiply the angle by PI()/180 or use the **radians** function to convert the angle to radians.

Parameters:

Parameter	Type	Required	Description
number	Number	Yes	The angle in radians for which you want the sine.

Return type:

number

Example:

```
number a = sin(1.047);
print("a= " + a);

number b = sin(60*pi()/180);
print("b= " + b);
```

Print: a= 0.8659266112878228 ;

b= 0.8660254037844386;

sqrt

> **Availability**

This routine is available since **katl-commons 2.5 .**

Syntax:

sqrt(number)

Description:

Returns a positive square root.

Parameters:

Parameter	Type	Required	Description
number	Number	Yes	The number for which you want the square root.

Return type:

number

Example:

```
number a =sqrt(16);
print("a= " + a);
number b = sqrt(100);
print("b= " + b);
```

Print: a= 4;

b= 10;

tan

Availability
This routine is available since **katl-commons 2.5 .**

Syntax:

tan(number)

Description:

Returns the tangent of a number.

Parameters:

Parameter	Type	Required	Description
number	Number	Yes	the number to get the tangent for

Return type:

number

Example:

```
number a = tan(4);
print("a= " + a);
```

Print: a= 1.1578212823495777;

trunc

> **Availability**
> This routine is available since **katl-commons 2.5 .**

Syntax:

trunc(number, digits)

Description:

Truncates a number to a specified precision.

Parameters:

Parameter	Type	Required	Description
number	Number	Yes	the number to truncate
digits	Number	Yes	the precision

Return type:

number

Example:

```
number a = trunc(1.1578212823495777, 3);
print("a= " + a);
```

Print: a= 1.157;

System Integration

Introduction

This section includes routines for System integration.

Routines summary

Routine	Description
call	Executes another SIL script. If sysname is equal with "local" (or is empty "") it will execute the script on the current JIRA server. The execution will automatically define an variable named 'argv' which will contain the parameters, as a string array. You can use return to return values back to the caller; again, a string array only. If the call is local, current issue is available. If the call is a remote call, the current context is lost (issue variables will have no meaning).
silEnv	Returns the variable as presented in the environment. A special variable is set '**sil.home**', which will point to the actual location of the SIL programs. You can configure this folder from the Administration Page.

system	Executes the command command of the operating system. Returns the exit code of the program and the output and error streams as strings. Through this you may integrate outside scripts (sh, ksh, perl, ...) with JIRA. Streams are limited to 4Kb (only the first 4Kb are taken into account).
sql	Executes the SQL phrase over the defined JNDI datasource. For selects returning multiple rows, it concatenates the values (i.e. you select 2 values and the select returns 4 rows, you will have 2*4 = 8 values). For updates, it returns the update count.
sqlCallStoredProcedure	Executes the stored procedure over the defined JNDI datasource.
sqlCallStoredProcedureWithOutParams	Executes the stored procedure over the defined JNDI datasource.
sendEmail	Sends an email. to and cc are string arrays with any of email addresses, users or groups, in which case an email will be sent to all the users of the groups.from is optional; if missing the email will be sent from the default email address configured in JIRA. The last parameter is the language and it is available in the full form of the routine. To pass the language, you must specify the sender, to, cc, subject, body and finally the language, as a string (e.g. "en", "fr", "en_US", "ro", etc.). However, this is relevant only if you use templates, since these support internationalization. For example, an email sent with language "en" will look for templates in the folder called "en", inside the default template directory. If no such template is found, it will use the one in the default directory. Also, since katl-commons-1.1.1, a default template placed in the default directory is mandatory for each template name used in your SIL programs. For example, if you want to use a template "t.tpl" using language "en_US",it is not only necessary to have "t.tpl" in the "en_US" folder, but you must also have a file "t.tpl" in the default directory. If you don't specify the language parameter and you use templates, by default the messages are sent in the sender defined language. For the users that are JIRA users(to or cc are user names and not email addresses) it can be used the language defined in the user profile(for each user) for the sending of the email, by changing the value of the parameter 'Send Email Language'(Kepler General Parameters->Main admin page->Select plugin katl-commons) to "receiver_language" (no quotes) instead of the default value "sender_language".For the rest of the users the email will be sent using the sender defined language.
httpPost	Executes a HTTP POST over the given URL. You can specify the headers (header,value, header, value, ...) and any parameters for that post (param, value, param, value, ...). Returns the HTTP code (i.e. 200 - OK).
httpGet	Executes a HTTP GET over the given URL. You can specify the headers (header,value, header, value, ...). Returns the HTTP code (i.e. 200 - OK).
ldapUserAttr	Returns an array of the requested attribute. This is a LDAP search routine. It gets the LDAP record and shows the attribute of that user (returned user must be unique, otherwise exception occurs). You can retrieve any attribute defined in the schema.
ldapUserList	Returns an array of the requested attributes for all users matching the query. This is a LDAP search routine.
ldapUserRecord	Returns an array of the requested attribute. This is a LDAP search routine. The second form is a convenience routine. Both get the LDAP record and shows the attribute of that user (returned user must be unique, otherwise exception occurs).
sendSMS	Sends an SMS using the ENMS service (an additional Kepler product). JJupin-integration-provider.jar is required. See the Configuration manual for more details. Returns false if sending failed.

call

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

call(sysname, silprogrampath, arguments)

Description

Executes another SIL script. If sysname is equal with "local" (or is empty "") it will execute the script on the current JIRA server.
The execution will automatically define an variable named 'argv' which will contain the parameters, as a string array.
You can use return to return values back to the caller; again, a string array only. If the call is local, current issue is available. If the call is a remote

call, the current context is lost (issue variables will have no meaning).

Parameters:

Parameter name	Type	Required	Description
sysname	string	yes	The system name. The local system (JIRA server machine) is identified by an empty string "" or by the string "local". For remote systems, it should contain the system name, as configured in the configuration page
silprogrampath	string	yes	Full path to the program being run (absolute path, i.e. "/opt/jira/home/silprograms/myprogram.sil" or "C:/Atlassian/Jira/Home/silprograms/myprogram.sil").
arguments	string []	yes	The arguments, as an array of strings

Returns:

string []

The return from the script, as an array of strings (values that are returned using the **return** keyword)

Example:

```
//Local script, placed in a postfunction (for instance):
string [] arrp = "param1|param2";
string [] rec;
call("", "/tmp/printme.sil", arrp); //local call
rec = call("system_remote", "c:/testme.sil", arrp); //remote call
if(isNotNull(rec)) {
  //The following code prints: "Hello", "from", "remote", "jjupin"
  for(string s in rec) {
    print(s);
  }
}
```

.....

```
//This is the remote SIl script placed in the c:/testme.sil:
string p = "Hello world!";
print("Remote called P is " + p + " Parameter at index 1 is=" +
arrayGetElement(argv, 1));
return "Hello", "from", "remote", "jjupin";
```

Notes:

> For remote calls, you need additional steps (see the configuration manual).
>
> Windows: We recommend you to use forward slashes "/" in paths instead of "\" since it will simplify your life.

Resolution of the remote system goes as follows: (as you may define the same name for a remote system in multiple places):

1. Try to see if the name of the system is empty ('') or the string 'local'. If yes, it will call a local script
2. Next, try to find the name of the system as defined by REST. If it is defined, it calls the REST remote system
3. If it is not defined, fallback on SOAP
4. If it is still not defined, error

Please see the configuration page on how you should configure the remote systems.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

databaseAvailable

Availability
This routine is available since **katl-commons 2.6.5**

Syntax:

databaseAvailable(dbstring [, sqlstring])

Description:

Checks the database available over the defined JNDI datasource. Optionally, it can check alto the availability of the SQL passed in as the second parameter.

Parameters:

Parameter name	Type	Required	Description
dbstring	string	yes	The datasource JNDI name. For JIRA database, this is set to "jdbc/JiraDS" by default
sqlstring	string	no	The SQL string

Return type:

boolean (true/false)

Example:

Let's suupose in our examples `TextField` is a custom field text already configured for the current Jira server instance and `TestDB` represents the database resource name configured in the server context. For more details see SQL Configuration.

Example 1:

```
TextField=databaseAvailable("TestDB"); //will set `TextField` value to
`true`
```

Example 2:

```
TextField=databaseAvailable("TestDB", "select pname from project"); //will
set `TextField` value to `true` if the SQL passed in can be executed
```

Example 3:

```
TextField=databaseAvailable("TestDB", "select pname from
inexistentTableName");//will set `TextField` value to `false`, because the
SQL passed in cannot be executed
```

Example 4:

```
TextField=databaseAvailable("InexistentTestDB", "select pname from
project");//will set `TextField` value to `false`, because the database
resource does not exist
```

Notes:

To see how you should configure the data source, check the corresponding configuration chapter: SQL Configuration

getIssueURL

Availability
This routine is available since **katl-commons 3.0.2.**

Syntax:

getIssueURL(issueKey)

Description:

Returns the url of the issue.

Parameters:

Parameter	Type	Required	Description
issueKey	String	Yes	the key of the selected issue

.

Return type:

The returned value represents the url of the issue.

Example:

```
return getIssueURL("TEST-1");
```

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

getJIRABaseUrl

> **Availability**
> This routine is available since **katl-commons 3.0.2 .**

Syntax:

getJIRABaseUrl()

Description:

This routine returns the base url of the JIRA from where you call the SIL script.

Parameters:

This routine has no parameters.

Returns:

The routine returns the base url of the JIRA.

Example:

Simple usage

```
return getJIRABaseUrl();
```

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

httpGet

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

httpGet(url, headers_array)

Description:

Executes a HTTP GET over the given URL. You can specify the headers (header,value, header, value, ...). Returns the HTTP code (i.e. 200 - OK).

Parameters:

Parameter name	Type	Required	Description
url	string	yes	The URL of the remote machine
headers_array	string []	yes	The headers to be sent over HTTP

Returns:

number

The returned value represents the HTTP code returned by the remote HTTP server.

Example:

```
string [] headers = "Accept|text/plain|Accept-Language|ro_RO";
httpGet("http://www.somerandomsite.com", headers);
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

httpPost

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

httpPost(url, headers_array, params_array)

Description:

Executes a HTTP POST over the given URL. You can specify the headers (header,value, header, value, ...) and any parameters for that post (param, value, param, value, ...). Returns the HTTP code (i.e. 200 - OK).

Parameters:

Parameter name	Type	Required	Description
url	string	yes	The URL to post to
headers_array	string []	yes	The headers array, as (name, value, name, value ...)
params_array	string []	yes	The parameters to be sent in the post (name, value, name, value, ...)

Returns:

number

Example:

```
string [] headers = "Accept|text/plain|Accept-Language|ro_RO";
string [] params = "param1|value1|param2|value2";
httpPost("http://www.somerandomsite.com", headers, params);
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

ldapUserAttr

> **Availability**
> This routine is available since **katl-commons 3.0**.

Syntax:

ldapUserAttr(attrib, ldapQuery)

Description:

Returns an array of the requested attribute. This is a LDAP search routine. It gets the LDAP record and shows the attribute of that user (returned user must be unique, otherwise exception occurs). You can retrieve any attribute defined in the schema.

Alias:

ldapUserRecord(attrib, ldapQuery) (deprecated)

Parameters:

Parameter name	Type	Required	Description
attrib	string	yes	the attribute to be returned
ldapQuery	string	yes	the query, must return exactly one result

Returns:

string []

The values of the specified attribute. If the attribute only has one value, the array will contain only one element, but will still be an array and not a single string.

OpenDS Example:

```
string email = ldapUserAttr("mail",
"(&(uid=user.1)(objectClass=inetOrgPerson))");
string address = ldapUserAttr("postalAddress",
"(&(uid=user.1)(objectClass=inetOrgPerson))");
```

Notes:

LDAP must be configured (see configuration manual).

Only Microsoft Active Directory is supported at this time, but it might work with others as well (tested with OpenDS). To provide support for other LDAP types, please contact us.

For more information about LDAP filters, see Active Directory LDAP filter syntax.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

ldapUserList

Availability
This routine is available since **katl-commons 2.5.15** (for JIRA 5.x) and **2.6.7** (for JIRA 6.x) .

Syntax:

ldapUserList(attributes, ldapQuery)

Description:

Returns an array of the requested attributes for all users matching the query. This is a LDAP search routine.

Parameters:

Parameter name	Type	Required	Description
attributes	string []	yes	the attributes to be returned
ldapQuery	string	yes	the query

Returns:

string []

The values of the attributes, for all users, in multiples of N, where N is the number of requested attributes. The length of the returned array will be N x M (N = number of attributes requested, M = number of users matching the query), such that element at index i is the value of the attribute at position i%N from the attributes array, for the $(i/N)^{th}$ user matching the query.

Example:

```
return ldapUserList({"cn", "uid"}, "objectClass=inetOrgPerson");
//example return value: Aaron Atrc|user.3|Aarika Atpco|user.2|Aaren
Atp|user.1|Aartjan Aalders|user.4|Aaccf Amar|user.0

// contains cn,uid for the 5 users matching the filter:
cn1,uid1,cn2,uid2,cn3,uid3 ...
```

You can use this routine with the User Group Picker PRO.

Example script for SIL User Picker:

```
return ldapUserList({"uid"},
"(&(objectClass=inetOrgPerson)(isMemberOf=cn=group-one,dc=example,dc
=com))");
```

Notes:

LDAP must be configured (see LDAP Configuration).

Only Microsoft Active Directory is supported at this time, but it might work with others as well (tested with OpenDS). To provide support for other LDAP types, please contact us.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

ldapUserRecord

Since **katl-commons 3.0**, this syntax is deprecated and you should use the new alias name: **ldapUserAttr** instead.

Availability
This routine is available since **katl-commons 1.0,** changed in **1.1.15** (for JIRA 4.3.x and JIRA 4.4.x) and **2.0.8** (for JIRA 5.x) .

Syntax:

ldapUserRecord(attrib, ldapQuery)

or (deprecated form)

ldapUserRecord(attrib, cn, dn, memberof, email, ldapObjectClass)

Description:

Returns an array of the requested attribute. This is a LDAP search routine. The second form is a convenience routine. Both get the LDAP record and shows the attribute of that user (returned user must be unique, otherwise exception occurs).

Parameters:

Parameter name	Type	Required	Description
attrib	string	yes	the attribute to be returned
ldapQuery	string	yes	the query, must return exactly one result

Allowed attributes are (these are **case sensitive**):

•CN
•DN
•firstName
•lastName
•displayName
•title
•department
•division
•officeName (mapped on physicalDeliveryOfficeName attribute)
•company
•empID
•manager
•mail
•otherMailbox
•mobile
•homePhone
•workPhone
•userPrincipalName
•winPrincipalName

> **Tip**
> Since katl-commons 1.1.15 and 2.0.8, you can retrieve any attribute defined in the schema.

Returns:

string []

The values of the specified attribute. If the attribute only has one value, the array will contain only one element, but will still be an array and not a single string.

Example:

```
ldapUserRecord("mobile", user, "", "", "", "user");
//gets the mobile attribute from LDAP user with specified CN
```

OpenDS example:

```
string email = ldapUserRecord("mail",
"(&(uid=user.1)(objectClass=inetOrgPerson))");
string address = ldapUserRecord("postalAddress",
"(&(uid=user.1)(objectClass=inetOrgPerson))");
```

Notes:

LDAP must be configured (see configuration manual).

Only Microsoft Active Directory is supported at this time, but it might work with others as well (tested with OpenDS). To provide support for other LDAP types, please contact us.

For more information about LDAP filters, see Active Directory LDAP filter syntax.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

ldapUserStruct

Availability
This routine is available since **katl-commons 3.0**.

Syntax:

ldapUserStruct(ldapQuery)

Description:

Returns an array of JLdapUserStruct representing all users matched by the query.

Parameters:

Parameter name	Type	Required	Description
ldapQuery	string	yes	the query, must return exactly one result

Returns:

JLdapUserStruct []

Each element in the array represents an user. The attributes field of the JLdapUserStruct is also keyed by the attribute name for easy access of attributes. Each attribute is a JLdapUserAttribute. The "value" field of the attribute is a string array. If the attribute only has one value, the array will contain only one element, but will still be an array and not a single string.

OpenDS Example:

```
JLdapUserStruct [] users = ldapUserStruct("objectClass=inetOrgPerson");
for(JLdapUserStruct u in users) {
    print(u.DN);
    for(JLdapUserAttribute attr in u.attributes) {
        print(attr.name + " = " + attr.value);
    }
    print("ID is : " + u.attributes["uid"].value);
}
```

Notes:

LDAP must be configured (see configuration manual).

Only Microsoft Active Directory is supported at this time, but it might work with others as well (tested with OpenDS). To provide support for other LDAP types, please contact us.

For more information about LDAP filters, see Active Directory LDAP filter syntax.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

runSILInline

Availability
This routine is available since **katl-commons 2.5.13 / 2.6.5 .**

Syntax:

runSILInline(script, args)

Description:

Executes the script with the given arguments

Parameters:

Parameter name	Type	Required	Description
script	string	Yes	The script to be executed
args	string[]	Yes	The arguments for the script

Return type:

string[]

The output of the script

Example:

```
string [] result = runSILInline("return \"SIL is \" + argv[0];",
"awesome"); // returns "SIL is awesome"

string script = "number sum = 0; for (string s in argv) { sum += (number)s;
} return sum;";
string [] argsArr = "1|2|3";
string [] result = runSILInline(script, argsArr); // returns "6"
```

Notes:

1. If you don't need/have any arguments, pass an empty string for the args parameter.

See also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

saveURLToFile

Availability
This routine is available since **katl-commons 3.0.8 .**

Syntax:

saveURLToFile(url, file)

Description:

Executes the command command of the operating system. Returns the exit code of the program and the output and error streams as strings. Through this you may integrate outside scripts (sh, ksh, perl, ...) with JIRA. Streams are limited to 4Kb (only the first 4Kb are taken into account).

Parameters:

Parameter name	Type	Required	Description
url	string	yes	Valid URL. No auth is performed
file	string	yes	Path to the (new) file that will keep the content of the URL

Returns:

The routine returns a boolean indicating if the call succeeded or not.

Example:

Simple usage

```
string testfile="c:/tests/mytempfile.html"; //assuming that folder `tests`
has been already created in c:\ path

if(saveURLToFile("https://kepler-rominfo.com/pages/solutions/jira-plugins",
testfile)) {
  //do something with that html
}
```

sendEmail

Availability
This routine is available since **katl-commons 1.0**, changed in **1.1.14** (for JIRA 4.3.x and 4.4.x) and **2.0.7** (for JIRA 5.x) .

Syntax:

sendEmail([from], to, [cc], subject, body_or_template, [language])

Since katl-commons 2.0.7, we also have support for attachments using one of two forms:

sendEmail(from, to, cc, subject, body_or_template, language, issue_key, regex_array)

sendEmail(from, to, cc, subject, body_or_template, language, wildcard_path_array)

Description:

Sends an email. to and cc are string arrays with any of email addresses, users or groups, in which case an email will be sent to all the users of the groups.from is optional; if missing the email will be sent from the default email address configured in JIRA.

The last parameter is the language and it is available in the full form of the routine. To pass the language, you must specify the sender, to, cc, subject, body and finally the language, as a string (e.g. "en", "fr", "en_US", "ro", etc.). However, this is relevant only if you use templates, since these support internationalization. For example, an email sent with language "en" will look for templates in the folder called "en", inside the default template directory. If no such template is found, it will use the one in the default directory. Also, since katl-commons-1.1.1, a default template placed in the default directory is mandatory for each template name used in your SIL programs. For example, if you want to use a template "t.tpl" using language "en_US",it is not only necessary to have "t.tpl" in the "en_US" folder, but you must also have a file "t.tpl" in the default directory.

If you don't specify the language parameter and you use templates, by default the messages are sent in the sender defined language. For the users that are JIRA users(to or cc are user names and not email addresses) it can be used the language defined in the user profile(for each user) for the sending of the email, by changing the value of the parameter 'Send Email Language'(Kepler General Parameters->Main admin page->Select plugin katl-commons) to "receiver_language" (no quotes) instead of the default value "sender_language".For the rest of the users the email will be sent using the sender defined language.

Parameters:

Parameter name	Type	Required	Description
from	string	no	the from address
to	string []	yes	recipient list
cc	string []	no	CC'ed recipient list
subject	string	yes	subject
body_or_template	string	yes	message body, either direct or a template
language	string	no	the language used to send the email(relevant only if you use templates)
issue_key	string	no	the issue to extract attachments from
regex_array	string []	if issue_key is present	name patterns to match the attachments from the issue
wildcard_path_array	string []	no	absolute paths containing wildcards for attaching files from disk

Alias:

For historical reasons, this routine may be named '**sendEmailFrom**'

Example:

Example 1:

```
sendEmail("projectmanager", "teamleader1", "Transition executed",
currentUser() + " has executed a transition");
// here we have to, cc, subject and body.
// The from and language parameters were omitted.
```

Example 2:

If SendEmailLanguage has the value receiver_language, testJiraUser1 is a JIRA user having the defined language French and testJiraUser2 is a JIRA user having the defined language German

```
string [] to = {"testJiraUser1", "testEmail@kepler.ro", "testJiraUser2"};
string [] cc = {"testEmail2@kepler.ro"};
sendEmail("testFrom@kepler.ro", to, cc, "testSubject.tpl", "testBody.tpl");
```

Result: It will be sent one email in French for testJiraUser1, one email in German for testJiraUser2 and one email in the sender defined language for testEmail@kepler.ro(as to) and testEmail2@kepler.ro(as cc).

Example 3:

Is similar with example 2, but here the language parameter is used.

If SendEmailLanguage has the value receiver_language, testJiraUser1 is a JIRA user having the defined language French and testJiraUser2 is a JIRA user having the defined language German

```
string [] to = {"testJiraUser1", "testEmail@kepler.ro", "testJiraUser2"};
string [] cc = {"testEmail2@kepler.ro"};
sendEmail("testFrom@kepler.ro", to, cc, "testSubject.tpl", "testBody.tpl",
"en_US");
```

Result: It will be sent one email in English(because of the en_US language parameter).

Example 4:

This example will demonstrate the ability to attach files (to the email) selected from the attachments of the issue using regex patterns.

We will assume that the issue has three attachments: attachment1.txt, attachment2.txt and attachment3atxt (note that this last one does not have a dot to separate the extension).

Now let's see a few examples of regex patterns that will match some of the attachment. Note that we will use key to specify the current issue, but feel free to use any other issue key.

```
sendEmail("santa@kepler.ro", {"jira-users"}, {}, "santa_subject.tpl",
"santa_letter.tpl", key, {"attachment.*"});
```

This will match all of the attachments. Since we are using regex patterns, **attachment.*** will match anything that starts with attachment.

```
sendEmail("santa@kepler.ro", {"jira-users"}, {}, "santa_subject.tpl",
"santa_letter.tpl", key, {"attachment.\\.txt"});
```

This will match attachment1.txt and attachment2.txt. The first dot will match any character (the 1 and 2). Note that the second dot is escaped using double backslashes and will not match attachment3atxt.

```
sendEmail("santa@kepler.ro", {"jira-users"}, {}, "santa_subject.tpl",
"santa_letter.tpl", key, {"attachment1\\.txt", "attachment3atxt"});
```

This will match attachment1.txt and attachment3atxt.

> Don't forget to use double backslashes when escaping special characters in regex patterns.

Example 5:

You can also attach files directly from disk by specifying absolute paths. Note that you can also use * (anything) and ? (any single char) as wildcards.

```
sendEmail("santa@kepler.ro", {"jira-users"}, {}, "santa_subject.tpl",
"santa_letter.tpl", {"C:/gifts/jira-users*.gift"});
```

Notes:

> If you would like to use templates for emails, see the Mail Configuration page.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

sendHtmlEmail

> **Availability**
> This routine is available since **katl-commons 2.5.10/ 2.6.2**

Syntax:

sendHtmlEmail([from], to, [cc], subject, body_or_template, [language])

sendHtmlEmail(from, to, cc, subject, body_or_template, language, issue_key, regex_array)

sendHtmlEmail(from, to, cc, subject, body_or_template, language, wildcard_path_array)

Description:

Sends an email, HTML formatted. to and cc are string arrays with any of email addresses, users or groups, in which case an email will be sent to all the users of the groups.from is optional; if missing the email will be sent from the default email address configured in JIRA.

The last parameter is the language and it is available in the full form of the routine. To pass the language, you must specify the sender, to, cc, subject, body and finally the language, as a string (e.g. "en", "fr", "en_US", "ro", etc.). However, this is relevant only if you use templates, since these support internationalization. For example, an email sent with language "en" will look for templates in the folder called "en", inside the default template directory. If no such template is found, it will use the one in the default directory. Also, a default template placed in the default directory is mandatory for each template name used in your SIL programs. For example, if you want to use a template "t.tpl" using language "en_US",it is not only necessary to have "t.tpl" in the "en_US" folder, but you must also have a file "t.tpl" in the default directory.

If you don't specify the language parameter and you use templates, by default the messages are sent in the sender defined language. For the

users that are JIRA users(to or cc are user names and not email addresses) it can be used the language defined in the user profile(for each user) for the sending of the email, by changing the value of the parameter 'Send Email Language'(Kepler General Parameters->Main admin page->Select plugin katl-commons) to "receiver_language" (no quotes) instead of the default value "sender_language".For the rest of the users the email will be sent using the sender defined language.

Parameters:

Parameter name	Type	Required	Description
from	string	no	the from address
to	string []	yes	recipient list
cc	string []	no	CC'ed recipient list
subject	string	yes	subject
body_or_template	string	yes	message body, either direct or a template
language	string	no	the language used to send the email(relevant only if you use templates)
issue_key	string	no	the issue to extract attachments from
regex_array	string []	if issue_key is present	name patterns to match the attachments from the issue
wildcard_path_array	string []	no	absolute paths containing wildcards for attaching files from disk

This routine shares the same semantics as the sendEmail routine. Check that routine for examples.

Notes:

> If you would like to use templates for emails, see the Mail Configuration page.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

sendSMS

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

sendSMS(senderName, phone, text)

Description:

Sends an SMS using the ENMS service (an additional Kepler product).

JJupin-integration-provider.jar is required.

See the Configuration manual for more details. Returns false if sending failed.

> More about the configuration can be found at SMS Provider Configuration section.

Parameters:

Parameter name	Type	Required	Description
senderName	string	yes	the sender name
phone	string	yes	the MSISDN of the destination

text	string	yes	the text

> The sender name can be either a sequence of maximum 11 characters or a phone number and must not contain spaces.

Returns:

boolean

Returns true if the message was sent successfully or false if an error occurred.

Example:

```
sendSMS("KEPLER","+40123456789","Hello world!");
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

silEnv

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

silEnv(variable_name)

Description:

Returns the variable as presented in the environment. A special variable is set '**sil.home**', which will point to the actual location of the SIL programs. You can configure this folder from the Administration Page.

Valid variables:

- **sil.home** - points to the folder keeping the SIL programs
- Any Java environment variable (such as '*java.home*', '*os.arch*', etc)
- Any environment variable, exported in the operating system (XX=aa; export XX)

Additionally, the routine looks into the '**sil.home**' directory after a file called '**sil.properties**'.
You can put all your host-dependent variables in here, and create easy-transferable scripts.
This file contains key-value pairs (name=value).

> Every time the **sil.home** is changed, the contents of the old *sil.properties* file will not be taken into account by this routine. You must copy the contents of the old file in the new location manually.

> See Environment Variables for more details.

Parameters:

Parameter name	Type	Required	Description
variable_name	string	yes	the env variable name

Returns:

string

Example:

```
print("Your SIL programs are in " + silEnv("sil.home"));
```

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

sql

> **Availability**
> This routine is available since **katl-commons 1.0**, changed in **2.5.8**.

Syntax:

sql(JNDIstring, sqlstring, [...])

Description:

Executes the SQL phrase over the defined JNDI datasource. For selects returning multiple rows, it concatenates the values (i.e. you select 2 values and the select returns 4 rows, you will have 2*4 = 8 values). For updates, it returns the update count.

Parameters:

Parameter name	Type	Required	Description
JNDIstring	string	yes	The datasource JNDI name. For JIRA database, this is set to "jdbc/JiraDS" by default
sqlstring	string	yes	the SQL string

Since **katl-commons 2.5.8**, the routine accepts multiple parameters, in this case the sql statement being pushed as prepared into the database (check the second example below for right syntax). The old syntax is still functional too.

Returns:

string []

Example:

Example 1:

```
string [] results = sql("TestDB_JNDI_Name", "select project_id from
project_lookaside where project_code='" + project + "'");
```

Example 2:

```
string [] results = sql("TestDB_JNDI_Name", "select project_id from
project_lookaside where project_code=?", project);
```

Notes:

> To see how you should configure the data source, check the corresponding configuration chapter: SQL Configuration.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

sqlCallStoredProcedure

> **Availability**
> This routine is available since **katl-commons 3.0.3**

Syntax:

sqlCallStoredProcedure(JNDIstring, procedureName, [...])

Description:

Executes the stored procedure over the defined JNDI datasource.

Parameters:

Parameter name	Type	Required	Description
JNDIstring	string	yes	The datasource JNDI name. For JIRA database, this is set to "jdbc/JiraDS" by default
procedureName	string	yes	the stored procedure name

The routine accepts multiple parameters, in this case the sql statement being pushed as prepared into the database (check the second example below for right syntax).

Returns:

string []

Example:

Example 1:

```
string [] results = sqlCallStoredProcedure("myDB", "showMessage");
```

Where showMessage() is a stored procedure existing in myDB database.

Example 2:

```
string [] results = sqlCallStoredProcedure("myDB", "addComponent",
"componentId", "componentName");
```

Where addComponent(String id, String name) is a stored procedure existing in myDB database.

Notes:

> To see how you should configure the data source, check the corresponding configuration chapter: SQL Configuration.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

sqlCallStoredProcedureWithOutParams

> **Availability**
> This routine is available since **katl-commons 3.0.6**

Syntax:

sqlCallStoredProcedureWithOutParams(JNDIstring, procedureName, [params])

Description:

Executes the stored procedure over the defined JNDI datasource.

Parameters:

Parameter name	Type	Required	Description
JNDIstring	string	yes	The datasource JNDI name. For JIRA database, this is set to "jdbc/JiraDS" by default
procedureName	string	yes	the stored procedure name

The routine accepts multiple parameters, in this case the sql statement being pushed as prepared into the database (check the second example below for right syntax).

Returns:

string []

Notes:

> To see how you should configure the data source, check the corresponding configuration chapter: SQL Configuration.
>
> Please notice that this routine work only with Oracle stored procedures.
>
> The parameters with type "OUT" must have three properties (corresponding parameter name from the procedure, data type, type), while the other ones must have four (corresponding parameter name from the procedure, data type, type, value).

Example:

Example 1:

```
string [] results = sqlCallStoredProcedureWithOutParams("myDB",
"showMessage");
```

Where showMessage() is a stored procedure existing in myDB database.

Example 2:

```
string [] results = sqlCallStoredProcedureWithOutParams("myDB",
"INSERTTEST", {"p_userid", "FLOAT", "IN", "1.22"}, {"p_username",
"VARCHAR2", "IN", "username"}, {"p_createdby", "VARCHAR2", "IN", "admin"},
{"p_date", "TIMESTAMP", "IN", "28-APR-2015"});
```

Where INSERTTEST is a stored procedure existing in myDB database, you can see it below:

```
create or replace PROCEDURE INSERTTEST (
       p_userid IN TEST.USER_ID%TYPE,
    p_username IN TEST.USERNAME%TYPE,
    p_createdby IN TEST.CREATED_BY%TYPE,
    p_date IN TEST.CREATED_DATE%TYPE )
IS
BEGIN

  INSERT INTO TEST ("USER_ID", "USERNAME", "CREATED_BY", "CREATED_DATE")
  VALUES (p_userid, p_username,p_createdby, p_date);

COMMIT;

END INSERTTEST;
```

Example 3:

```
string [] results = sqlCallStoredProcedureWithOutParams("myDB",
"TESTPROCEDURE", {"pObjName", "VARCHAR2", "IN", ""}, {"p_cursor", "REF
CURSOR", "OUT"}, {"param", "VARCHAR2", "OUT"});
return results;
```

Where TESTPROCEDURE(pObjName IN varchar2, p_cursor OUT SYS_REFCURSOR, param OUT VARCHAR2) is a stored procedure existing in myDB database.

You can also return the result this way:

```
string [] results = sqlCallStoredProcedureWithOutParams("myDB",
"TESTPROCEDURE", {"pObjName", "VARCHAR2", "IN", ""}, {"p_cursor", "REF
CURSOR", "OUT"}, {"param", "VARCHAR2", "OUT"});
return results["p_cursor"];
```

This will return the entry for "p_cursor"; same result will be returned for: return results[0];

```
string [] results = sqlCallStoredProcedureWithOutParams("myDB",
"TESTPROCEDURE", {"pObjName", "VARCHAR2", "IN", ""}, {"p_cursor", "REF
CURSOR", "OUT"}, {"param", "VARCHAR2", "OUT"});
string[] t = results["p_cursor"];
return t[3];
```

This will return the third entry from "p_cursor".

Example 4:

```
string[] results = sqlCallStoredProcedureWithOutParams("myDB",
"TESTPROCEDUREINOUTPARAMETER", {"param", "VARCHAR2", "IN OUT",
"paramValue"});
return results;
```

Where TESTPROCEDUREINOUTPARAMETER (param IN OUT VARCHAR2) is a stored procedure existing in myDB database.

In the next table you can see the mapping between SQL Types and SIL Types:

SQL Type	SIL Type
User-defined collection	ARRAY
CHAR	CHAR
NUMBER	DECIMAL
NUMBER	FLOAT
NUMBER	INTEGER
LONG	LONG
REF CURSOR	REF CURSOR
DATE	TIMESTAMP
VARCHAR2	VARCHAR2

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

system

> **Availability**
> This routine is available since **katl-commons 1.0 .**

Syntax:

system(command)

Description:

Executes the command command of the operating system. Returns the exit code of the program and the output and error streams as strings. Through this you may integrate outside scripts (sh, ksh, perl, ...) with JIRA. Streams are limited to 4Kb (only the first 4Kb are taken into account).

Parameters:

Parameter name	Type	Required	Description
command	string	yes	Valid Operating System command

Returns:

string []

The routine returns

1. The operating system exit code of the process being spawned
2. The output stream (limited to 4Kb) as a string
3. The error stream (limited to 4Kb) as a string

in this order.

Example:

Let's create a file using system routine and windows command prompt:

Creating a file using command promt

```
string testfolder="c:/tests"; //assuming that foler `tests` has been
already created in c:\ path

system("C:/WINDOWS/system32/cmd.exe /c echo return true; > " + testfolder +
"qqq.sil"); // you can run here any other program or custom script
```

Let's invoke a windows .bat script file using system routine:

Invoking a windows .bat file

```
string testfolder="c:/tests/"; //assuming this path exists.
//we assume in this path a file called `myexec.bat` has been previously
created and contains one line `echo Hello;`

//now the following call will return `0.0|[THE_PATH_OF_EXECUTION]>echo
Hello; Hello;|`
return system("C:/WINDOWS/system32/cmd.exe /c " + testfolder +
"myexec.bat");
```

Note:

> For windows operating system, you should put the full (absolute) path of the command.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

User-Defined Routines (UDR)

On this page:

Introduction

User-defined routines (UDR) are functions that perform specific actions, which you can define in your SIL programs for a later use. These can considerably improve the readability and maintainability of your code.

Syntax

```
function <name>(<type> param1, <type> param2, ...) {
    Instruction1;
    ...
    InstructionN;
    return <value>;
}
```

> The name of the UDR cannot contain spaces.

Example:

```
function zero(){
    return 0;
}
number a = zero();
```

> Definition of UDRs must be done before the code, even though it is not used anywhere up to that point. Therefore, the following code is **invalid**.
>
> ```
> number a;
> function zero(){
>     return 0;
> }
> ```

Parameters

The list of parameters in the definition of a UDR can be of any length (including 0) and their respective types can be any valid SIL type.

Example:

```
function zero(){
    return 0;
}

function doSomething(string s, number n1, number [] n2, boolean flag,
string [] oneMore){
    ....
}
```

UDRs use a "pass-by-value" policy. This means that even though you modify the value of a parameter in your function, **on exit** the value **will be lost**.

Example:

```
function increment(number a){
    a = a + 1; // the value of a is only modified locally
    return a;
}
number b = 0;
number c = increment(b); // the value of b does not change
print(b);  // this prints 0
print(c);  // this prints 1
```

Constant Parameters

Parameters of user-defined routines can be made read-only in the scope of the routine by adding the keyword "**const**" before the parameter definition in the signature of the routine.

```
function f(const string s) {
 ...
}
```

Variable visibility

There are three categories of variables that can be used in a UDR:

1. Local variables

These are the variables you define in the body of the UDR. These can be used throughout the body of the UDR. On exit, the values of these variables are lost.

```
function example(){
    number a = 3;
    number b = a + 10;
    // use here variables a and b
}
```

2. Parameter variables

These are the values passed to the UDR in the list of parameters. Because SIL uses a "pass-by-value" policy, even though you modify the value of these variables in the body of the function, on exit, their original values will be restored.

```
function increment(number a){
    a = a + 1; // the value of a is only modified locally
    return a;
}
number b = 0;
number c = increment(b); // the value of b does not change
print(b);  // this prints 0
print(c);  // this prints 1
```

3. Global variables

These are the variables that are already defined and can be used right away (issue fields, customfields and any variables defined before the routine). You can use issue fields and customfields anywhere in your code (including in the UDR body) without having to declare them.

```
function printKey(){
    print(key);
}
```

Notice that the **key** variable is a standard issue field that you could otherwise use anywhere in your SIL program without having to declare it.

Return value

Return values can be used to communicate with the context that called the UDR or to halt its execution.

Examples:

```
function isEven(number a){
    if(a % 2 == 0){
        return true;
    }
    return false;
}
```

```
function increment(number a){
    return a + 1;
}

number b = increment(2);
```

Notice that there is no need to declare the type of the return value; this will be evaluated at runtime. Therefore, even though the check on the following program will be ok, at runtime the value of **d** will **NOT** be modified because of the incompatibility between date (on the right-hand-side) and number (on the left-hand-side).

```
function increment(number a){
    return a + 1;
}

date d = increment(2);
```

There can be only one return value (at most). If you would like to return more values of the same type, consider using an array.

You can return simply from a routine without specifying a value. However, you should always remember that by design routines return a value, even if it is undefined. The following code is therefore valid:

```
function f(number a) {
    if(a > 0) {
        print("positive");
        return;
    }
    if(a == 0) { print("ZERO"); }
}

//[...................]

string s =f(4); //s is still undefined, no value was returned

if(isNull(s)) {
  print("S IS NULL!"); //this will be printed
} else {
  print("S IS NOT NULL!");
}
```

Of course, the above code will print the text '*S IS NULL'* in the log.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

JJUPIN Routines

Introduction

In this section you can find a collection of routines that are available by using our JJupin plugin.

Info
For more information, see our JJupin documentation.

Routines summary

Routine	Description

lfHide	Hides a field.
lfShow	Shows a field.
lfDisable	Disables a field.
lfEnable	Enables a field.
lfShowFieldMessage	Displays a message for the given field.
lfHideFieldMessage	Hides a message for the given field.
lfGlobalMessage	Displays a global message.
lfDialogMessage	Displays a global message in a dialog.
lfSet	Sets a field with the given values.
lfWatch	Attach listeners for the given events.
lfExecuteJS	Gives you the possibility to run your own javascript code.

DBCF Routines

Introduction

If you need the information from an external database brought into JIRA, use data table custom field to see the whole data from an entire table.

This section contains a collection of routines which retrieves data from a data table custom field.

These routines are available in Database Custom Field plugin.

Routines summary

Routine	Description
getColumn	Retrieves the data in the column at the specified index from a data table custom field.
getHeaders	Retrieves the headers from a data table custom field.
getRow	Retrieves the data in the row at the specified index from a data table custom field.
getTableElement	Retrieves the data in the cell at the specified location from a data table custom field.

UGP Routines

Introduction

This section contains a collection of routines that are available by using our User Group Picker plugin.

Routines Summary

Routine	Description
getGroupsOf	Retrieves the groups mapped to the given user group picker custom field.

Parameter Routines

Introduction

This section includes the routines meant to help the user define and configure the starting parameters for the SIL scripts used in this plugin and to retrieve values from the parameters defined by the input routines. A similar set of routines may be found in another one of our plugins, namely the BA plugin. Aliases for these routines can be found in Sil Excel Reporting plugin..

See Also:

Routines
Syntax
Variable Resolution

Gadget Input Routines

Introduction

This section includes the routines meant to help the user define and configure the starting parameters for the SIL scripts used in this plugin. A similar set of routines may be found in another one of our plugins, namely the BA plugin.

Routines summary

Routine	Description	Syntax
gadget_createSingleCheckbox	Creates a single checkbox.	gadget_CheckboxGroup(label, isChecked[, isRequired, fieldDescription])
gadget_createCheckboxGroup	Creates a checkbox group(multiple select).	gadget_createCheckboxGroup(label, options, defaultValues[, isRequired, fieldDescription])
gadget_createRadioGroup	Creates a radio button group(single select).	gadget_createRadioGroup(label, options, defaultValue[, isRequired, fieldDescription])
gadget_createSelectList	Creates a single select list.	gadget_createSelectList(label, options, defaultValue[, isRequired, fieldDescription])
gadget_createMultiSelectList	Creates a multi select list.	gadget_createMultiSelectList(label, options, defaultValues[, isRequired, fieldDescription])
gadget_createInput	Creates a single line text input.	gadget_createInput(label, defaultValue[, isRequired, fieldDescription])
gadget_createTextArea	Creates a multiple line text area.	gadget_createTextArea(label, defaultValue[, rows[, isRequired, fieldDescription]])
gadget_createDatePicker	Creates a standard Date picker for selecting a date.	gadget_createDatePicker(label, defaultValue, isRequired, fieldDescription)
gadget_createDateTimePicker	Creates a DateTime picker for selecting a date and a time.	gadget_createDateTimePicker(label, defaultValue, isRequired, fieldDescription)
gadget_createUserPicker	Creates a user picker with a single selectable value.	gadget_createUserPicker(label, defaultValue, isRequired, fieldDescription)
gadget_createMultiUserPicker	Creates a user picker with multiple selectable values.	gadget_createMultiUserPicker(label, defaultValue, isRequired, fieldDescription)

gadget_createCheckboxGroup

Syntax:

gadget_createCheckboxGroup(label, options, defaultValue[, isRequired, fieldDescription])

Description:

Creates a checkbox group.

Parameters:

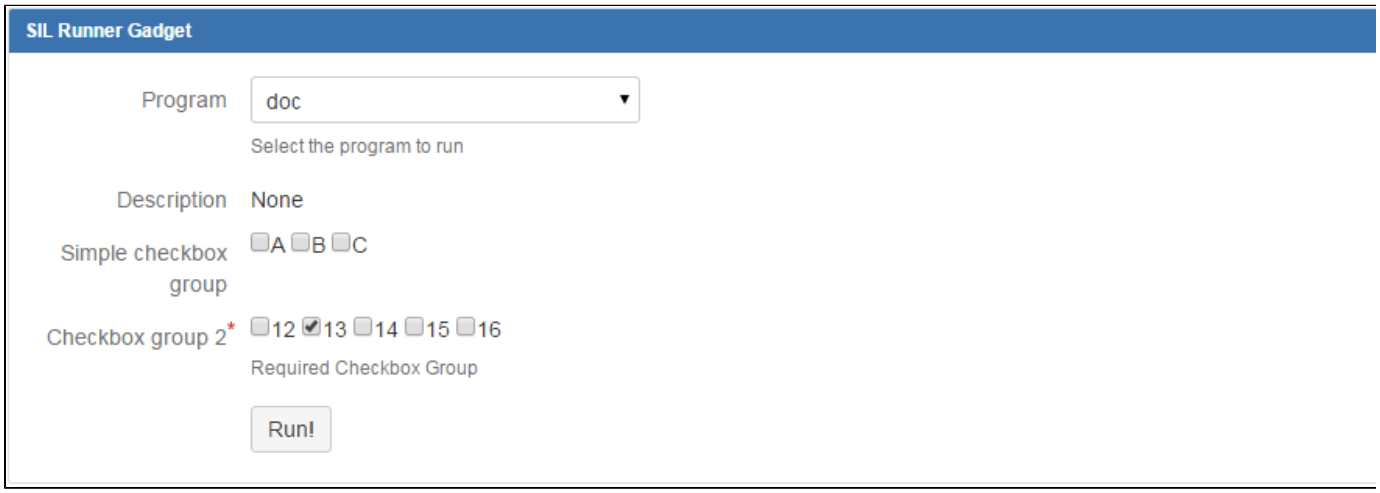
Parameter name	Type	Required	Description
label	string	Yes	The label of the field
options	string []	Yes	The list of selectable options.
defaultValue	string	Yes	A default value (one of the options provided) or an empty string
isRequired	boolean	No	Specifies if the field is should be marked as required with a dark red asterisk. Note that marking the field as required does not add any validation.
fieldDescription	string	No	A description of the field to be displayed immediately under the input box.

Return type:

string []

Example:

```
gadget_createCheckboxGroup("Simple checkbox group", {"A", "B", "C"}, "Not
required");
gadget_createCheckboxGroup("Checkbox group 2", {"12", "13", "14",
"15","16"}, {"13"}, true, "Required Checkbox Group");
```



gadget_createDatePicker

Syntax:

gadget_createDatePicker(label, defaultValue, isRequired, fieldDescription)

Description:

Creates a date picker.

Parameters:

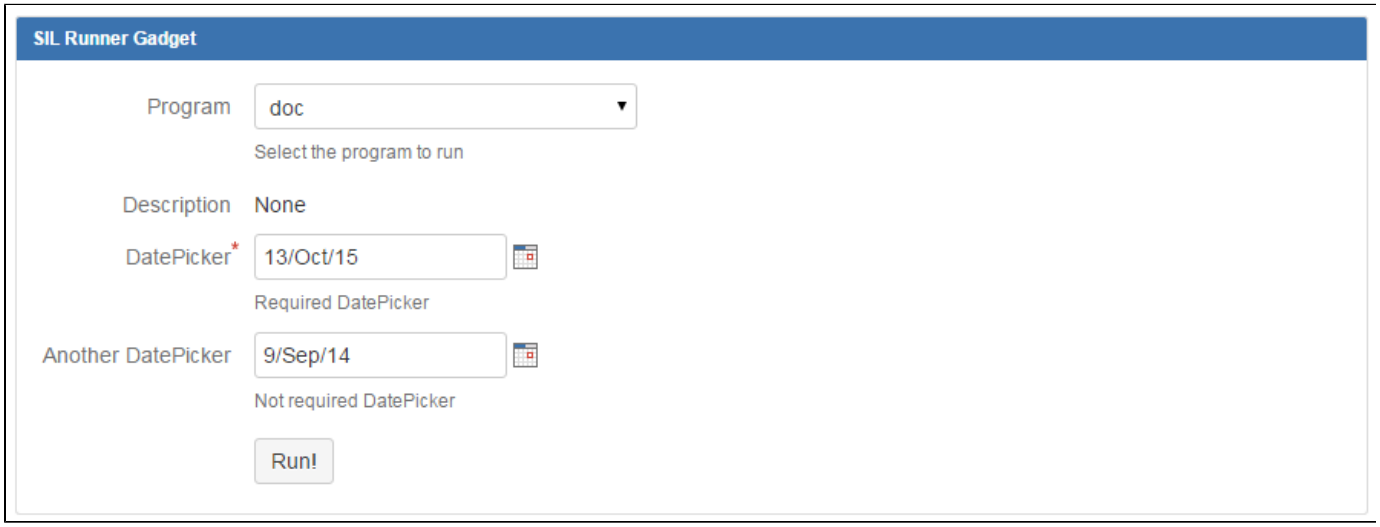
Parameter name	Type	Required	Description
label	string	Yes	The label of the field
defaultValue	date	Yes	A default value or an empty string
isRequired	boolean	Yes	Specifies if the field is should be marked as required with a dark red asterisk. Note that marking the field as required does not add any validation.
fieldDescription	string	Yes	A description of the field to be displayed immediately under the input box.

Return type:

The returned value has no meaning

Example:

```
gadget_createDatePicker("DatePicker", currentDate(), true, "Required
DatePicker");
gadget_createDatePicker("Another DatePicker", (date)"2014-09-09", false,
"Not required DatePicker");
```



gadget_createDateTimePicker

Syntax:

gadget_createDateTimePicker(label, defaultValue, isRequired, fieldDescription)

Description:

Creates a date/time picker.

Parameters:

Parameter name	Type	Required	Description
label	string	Yes	The label of the field
defaultValue	date	Yes	A default value or an empty string
isRequired	boolean	Yes	Specifies if the field is should be marked as required with a dark red asterisk. Note that marking the field as required does not add any validation.

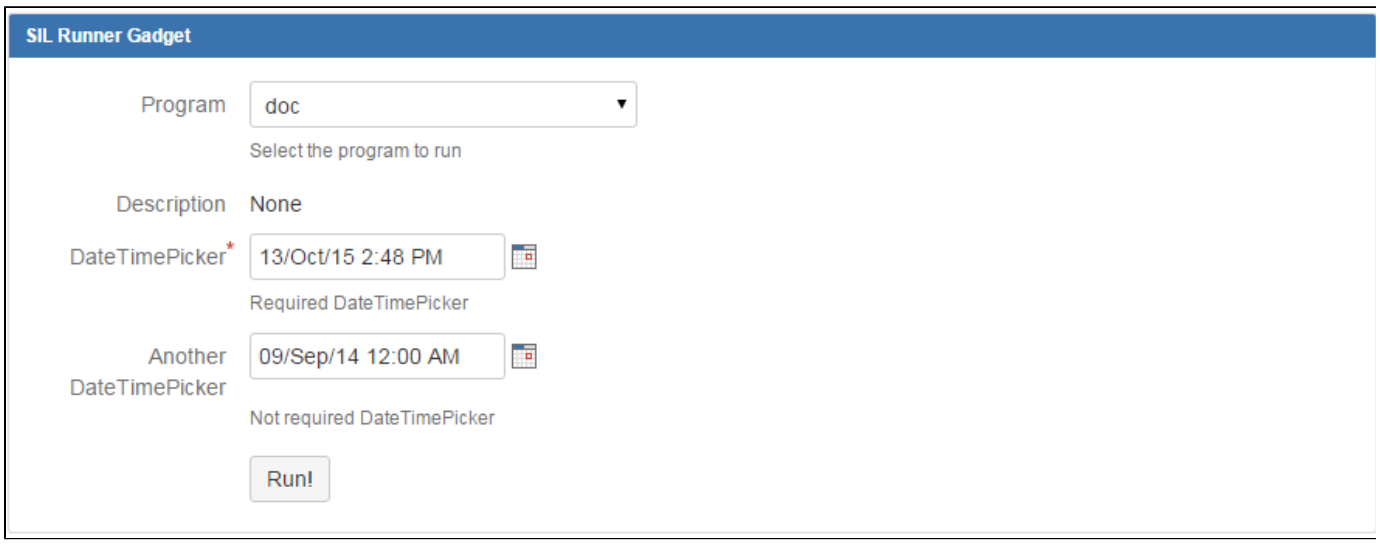
fieldDescription	string	Yes	A description of the field to be displayed immediately under the input box.

Return type:

The returned value has no meaning

Example:

```
gadget_createDateTimePicker("DateTimePicker", currentDate(), true,
"Required DateTimePicker");
gadget_createDateTimePicker("Another DateTimePicker", (date)"2014-09-09",
false, "Not required DateTimePicker");
```



gadget_createInput

Syntax:

gadget_createInput(label, defaultValue[, isRequired, fieldDescription])

Description:

Creates a simple text input suitable for short values.

Parameters:

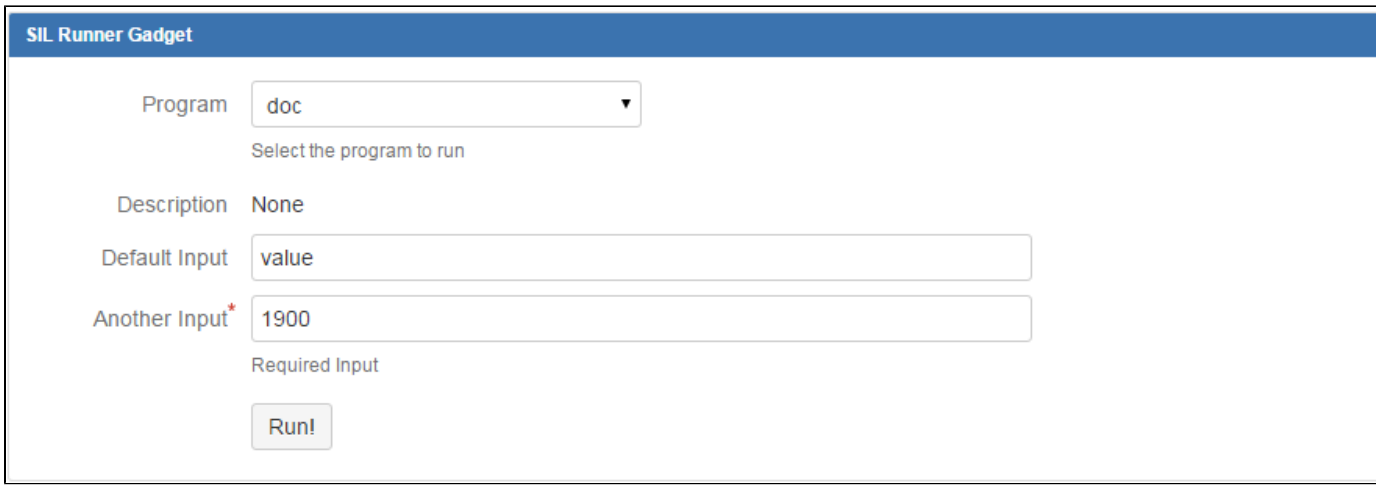
Parameter name	Type	Required	Description
label	string	Yes	The label of the field
defaultValue	string	Yes	A default value or an empty string
isRequired	boolean	No	Specifies if the field is should be marked as required with a dark red asterisk. Note that marking the field as required does not add any validation.
fieldDescription	string	No	A description of the field to be displayed immediately under the input box.

Return type:

string []

Example

```
gadget_createInput("Default Input", "value");
gadget_createInput("Another Input", "1900", true, "Required Input");
```



gadget_createMultiSelectList

Syntax:

gadget_createMultiSelectList(label, options, defaultValues[, isRequired, fieldDescription])

Description:

Creates a multi select list.

Parameters:

Parameter name	Type	Required	Description
label	string	Yes	The label of the field
options	string []	Yes	The list of selectable options.
defaultValue	string []	Yes	A sub-list of the options provided or an empty array
isRequired	boolean	No	Specifies if the field is should be marked as required with a dark red asterisk. Note that marking the field as required does not add any validation.
fieldDescription	string	No	A description of the field to be displayed immediately under the input box.

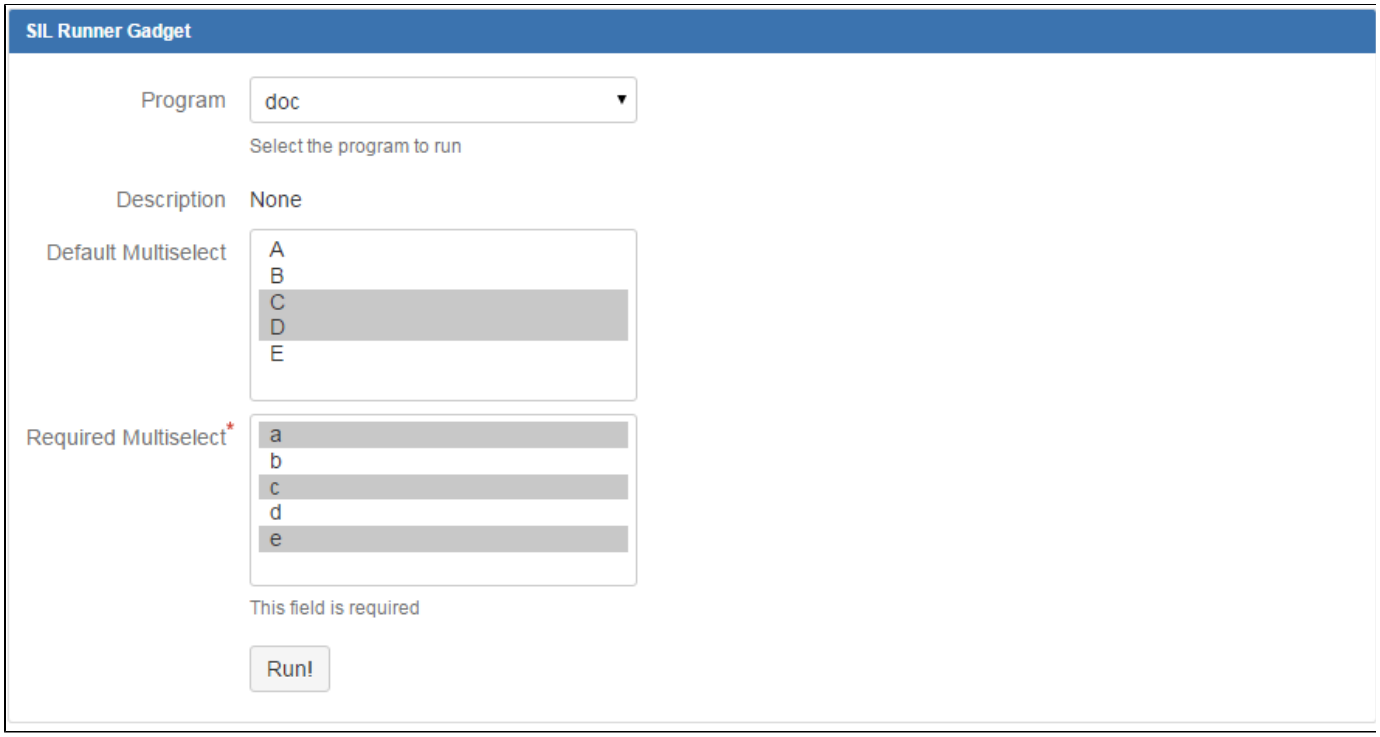
Return type:

string []

Examples:

Example 1:

```
gadget_createMultiSelectList("Default Multiselect", {"A", "B", "C", "D",
"E"}, {"C", "D"});
gadget_createMultiSelectList("Required Multiselect", {"a", "b", "c", "d",
"e"}, {"a", "c", "e"}, true, "This field is required");
```



gadget_createMultiUserPicker

Syntax:

gadget_createMultiUserPicker(label, defaultValues, isRequired, fieldDescription)

Description:

Creates a multi user picker.

Parameters:

Parameter name	Type	Required	Description
label	string	Yes	The label of the field
defaultValues	string []	Yes	A few default values or an empty array. If one the default values is not a valid username, it will be discarded and an empty default value will be provided.
isRequired	boolean	Yes	Specifies if the field is should be marked as required with a dark red asterisk. Note that marking the field as required does not add any validation.
fieldDescription	string	Yes	A description of the field to be displayed immediately under the input box.

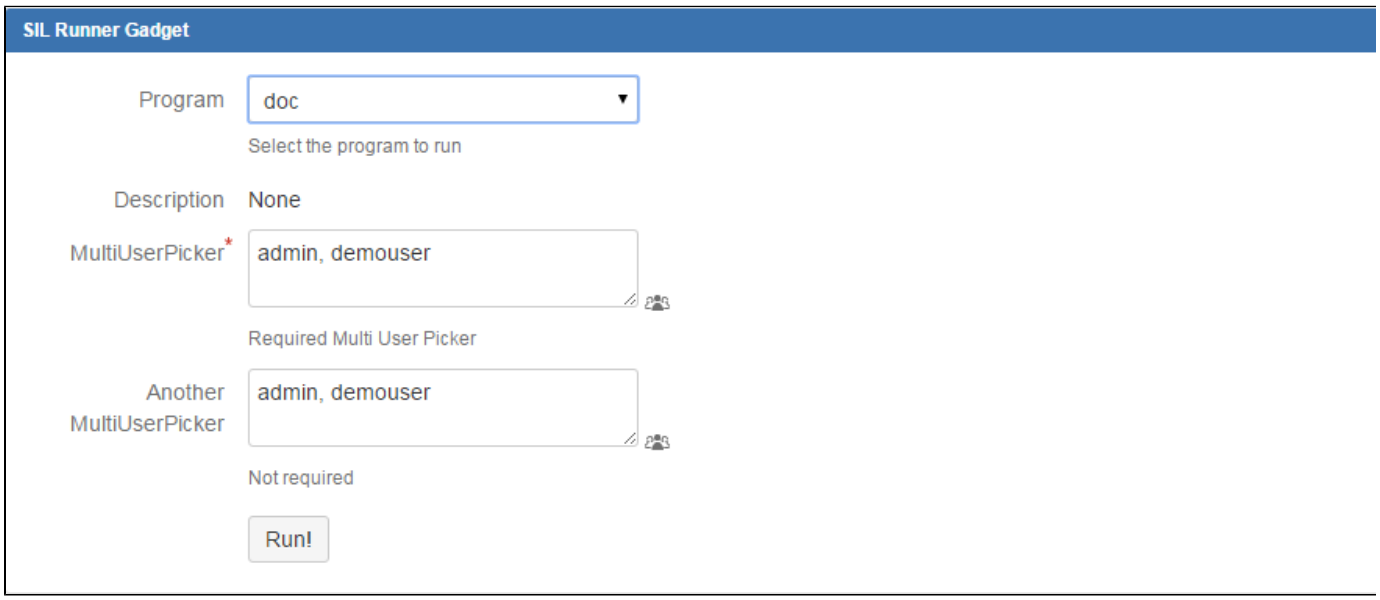
Return type:

The returned value has no meaning

Examples:

Example 1:

```
gadget_createMultiUserPicker("MultiUserPicker", {"admin", "demouser"},
true, "Required Multi User Picker");
gadget_createMultiUserPicker("Another MultiUserPicker", {"admin",
"demouser"}, false, "Not required");
```



gadget_createRadioGroup

Syntax:

gadget_createRadioGroup(label, options, defaultValue[, isRequired, fieldDescription])

Description:

Creates a radio group.

Parameters:

Parameter name	Type	Required	Description
label	string	Yes	The label of the field
options	string []	Yes	The list of selectable options.
defaultValue	string	Yes	A default value (one of the options provided) or an empty string
isRequired	boolean	No	Specifies if the field is should be marked as required with a dark red asterisk. Note that marking the field as required does not add any validation.
fieldDescription	string	No	A description of the field to be displayed immediately under the input box.

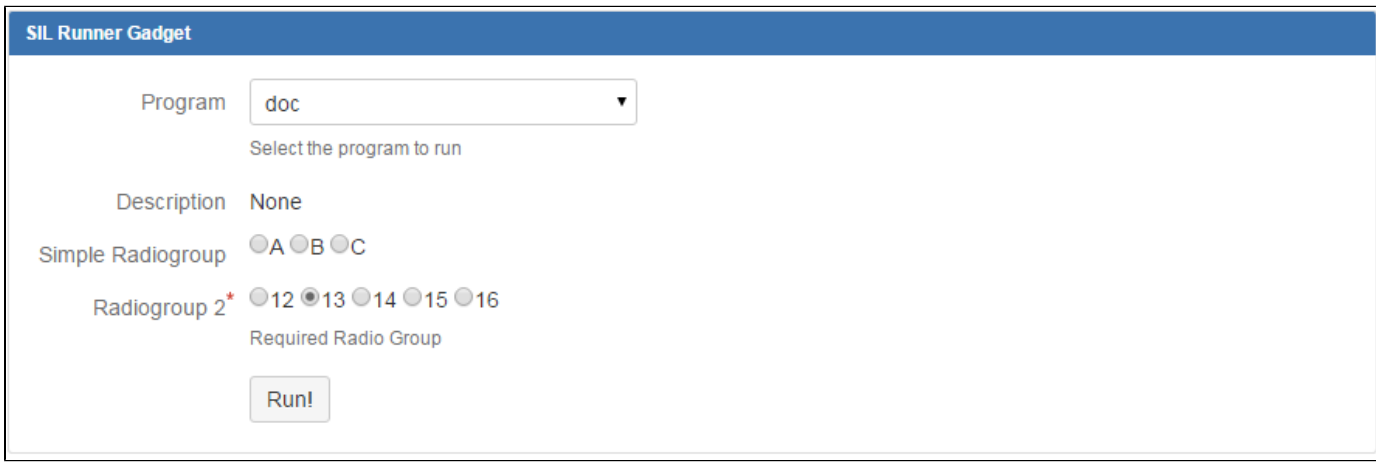
Return type:

string []

Examples:

Example 1:

```
gadget_createRadioGroup("Simple Radiogroup", {"A", "B", "C"}, "Not
required");
gadget_createRadioGroup("Radiogroup 2", {"12", "13", "14", "15","16"},
{"13"}, true, "Required Radio Group");
```



gadget_createSelectList

Syntax:

gadget_createSelectList(label, options, defaultValue[, isRequired, fieldDescription])

Description:

Creates a select list (combo).

Parameters:

Parameter name	Type	Required	Description
label	string	Yes	The label of the field
options	string []	Yes	The list of selectable options.
defaultValue	string	Yes	A default value (one of the options provided) or an empty string
isRequired	boolean	No	Specifies if the field is should be marked as required with a dark red asterisk. Note that marking the field as required does not add any validation.
fieldDescription	string	No	A description of the field to be displayed immediately under the input box.

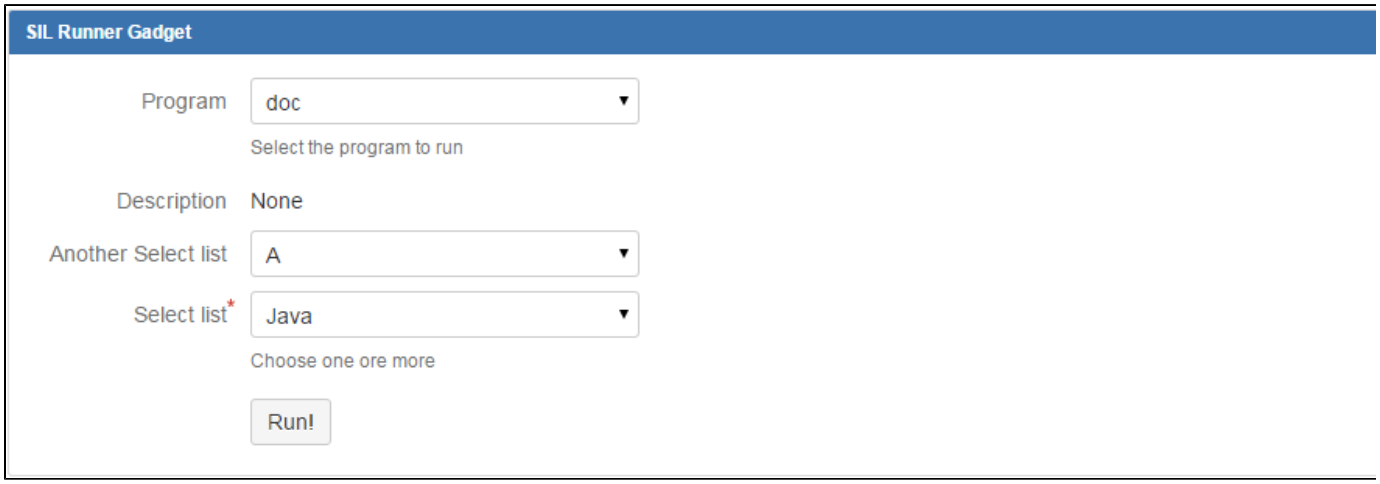
Return type:

string []

Examples:

Example 1:

```
gadget_createSelectList("Another Select list", {"A", "B", "C", "D"}, {});
gadget_createSelectList("Select list", {"C/C++", "Java", "Python", "Ruby"},
{"Java"}, true, "Choose one ore more");
```



gadget_createSingleCheckbox

Syntax:

gadget_createSingleCheckbox(label, isChecked [, isRequired, fieldDescription])])

Description:

Creates a single checkbox

Parameters:

Parameter name	Type	Required	Description
label	string	Yes	The label of the field
isChecked	boolean	Yes	The default value
isRequired	boolean	No	Specifies if the field is should be marked as required with a dark red asterisk. Note that marking the field as required does not add any validation.
fieldDescription	string	No	A description of the field to be displayed immediately under the input box.

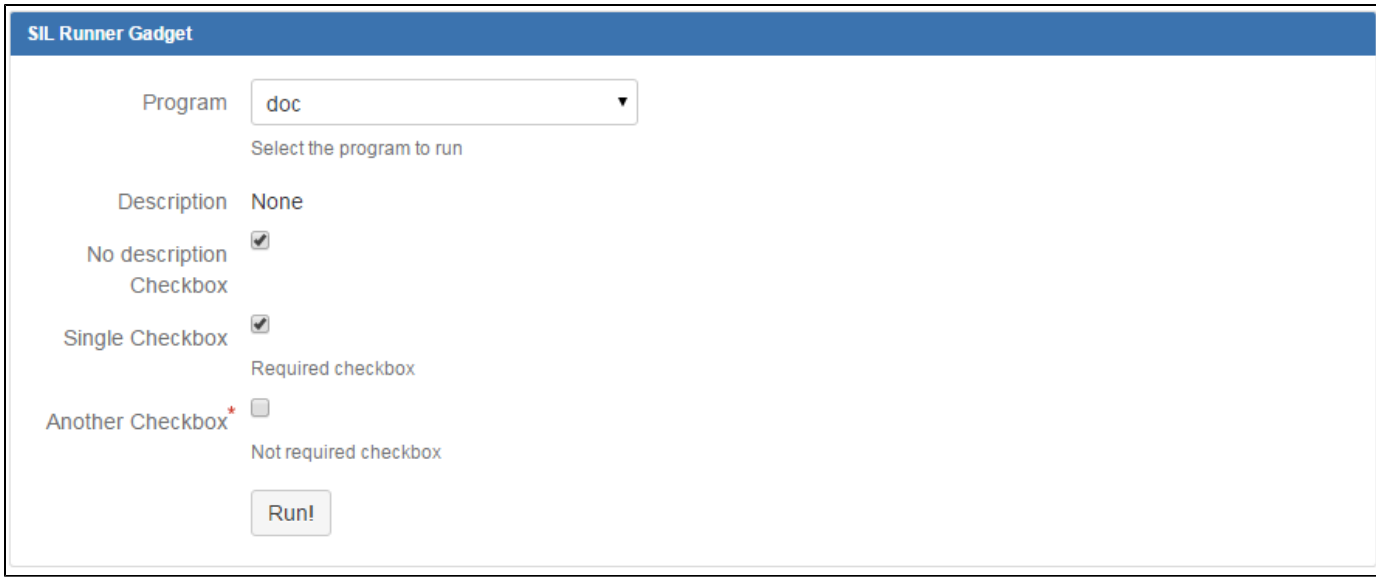
Return type:

string []

Examples:

Example 1:

```
gadget_createSingleCheckbox("No description Checkbox", true);
gadget_createSingleCheckbox("Single Checkbox", true, false, "Required
checkbox");
gadget_createSingleCheckbox("Another Checkbox", false, true, "Not required
checkbox");
```



gadget_createTextArea

Syntax:

gadget_createTextArea(label, defaultValue, isDisabled [, rows[, isRequired, fieldDescription]])

Description:

Creates a textarea suitable for longer text values like comments.

Parameters:

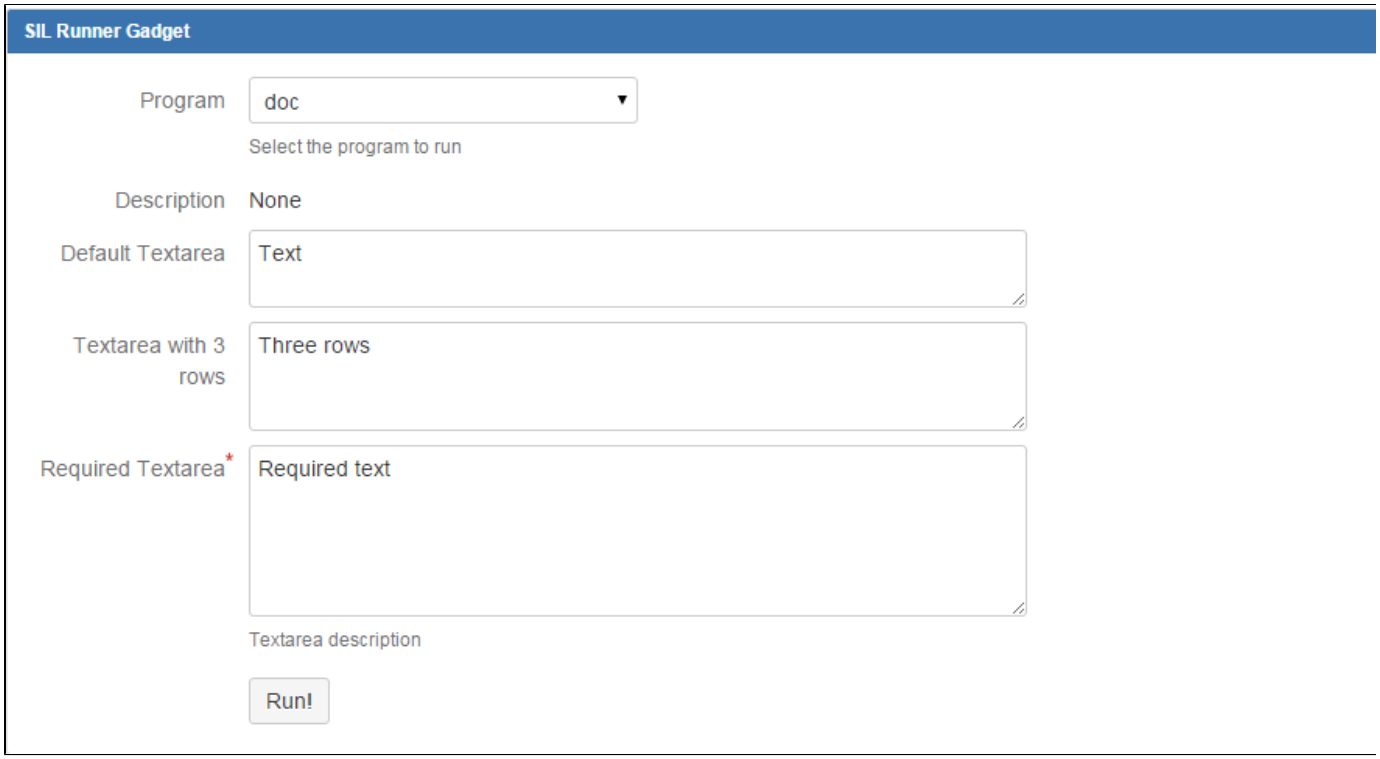
Parameter name	Type	Required	Description
label	string	Yes	Specifies a label for the field
defaultValue	string	Yes	Specifies a default value for the textarea
rows	number	No	The initial height of the textarea (defaults to 5). Some browsers will allow resizing.
isRequired	boolean	No	Specifies if the field is should be marked as required with a dark red asterisk. Note that marking the field as required does not add any validation.
fieldDescription	string	No	A description of the field to be displayed immediately under the input box.

Return type:

string []

Examples:

```
gadget_createTextArea("Default Textarea", "Text");
gadget_createTextArea("Textarea with 3 rows", "Three rows", 3);
gadget_createTextArea("Required Textarea", "Required text", 5, true,
"Textarea description");
```



gadget_createUserPicker

Syntax:

gadget_createUserPicker(label, defaultValue, isDisabled, isRequired, fieldDescription)

Description:

Creates a user picker.

Parameters:

Parameter name	Type	Required	Description
label	string	Yes	The label of the field
defaultValue	string	Yes	A default value or an empty string. If the default value is not a valid username, it will be discarded and an empty default value will be provided.
isRequired	boolean	Yes	Specifies if the field is should be marked as required with a dark red asterisk. Note that marking the field as required does not add any validation.
fieldDescription	string	Yes	A description of the field to be displayed immediately under the input box.

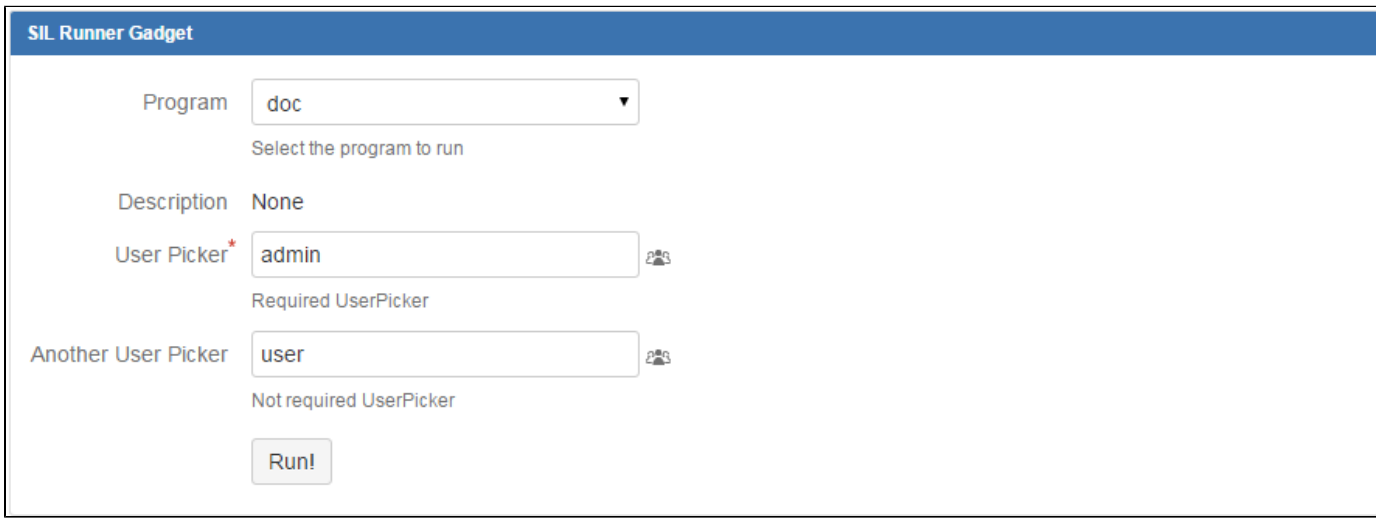
Return type:

The returned value has no meaning

Examples:

Example 1:

```
gadget_createUserPicker("User Picker", "admin", true, "Required
UserPicker");
gadget_createUserPicker("Another User Picker", "admin", false, "Not
required UserPicker");
```



Parameter Retrieval Routines

The following routines are meant to help the user retrieve values from the parameters defined by the input routines. The values obtained are used further by the reporting script.

A similar set of routines may be found in another one of our plugins, namely the BA plugin.

Routine	Description	Used with	Syntax
gadget_getDateValue	Retrieves the value of a DatePicker/DateTimePicker input.	• date picker • date time picker	gadget_getDateValue(argv, label)
gadget_getMultiUserPickerValue	Retrieves the values from a MultiUserPicker input.	• multi user picker	gadget_getMultiUserPickerValue(argv, label)
gadget_getMultiValues	Retrieves the selected values from a MultiSelectList or a similar input control.	• multi select list • checkbox group	gadget_getMultiValues(argv, label)
gadget_getSingleValue	Retrieves the value from an Input or other single value input control.	• text • text area • select list • radio group	gadget_getSingleValue(argv, label)
gadget_isChecked	Returns true if the checkbox is selected and false otherwise	• checkbox	gadget_isChecked(argv, label)

gadget_getDateValue

> **Availability**
> This routine is available since **katl-commons 3.0.8.**

Syntax:

gadget_getDateValue(argv, label)

Description:

Retrieves the date from a datePicker or dateTimePicker.

Parameters:

Parameter name	Required	Description
argv	Yes	the argv variable
label	Yes	the label of the datePicker

Return type:

date

Example:

For the Date type fields let's assume we have the following script:

```
gadget_createDatePicker("Start Date", currentDate(), true, "Required
DatePicker");
```

The date selected in the DatePicker field created above can be obtained in the execution script using the following code:

```
date res = gadget_getDateValue(argv, "Start Date");
```

gadget_getMultiUserPickerValue

> **Availability**
> This routine is available since **katl-commons 3.0.8.**

Syntax:

gadget_getMultiUserPickerValue(argv, label)

Description:

Retrieves the value from a multi user picker.

Parameters:

Parameter name	Required	Description
argv	Yes	the argv variable
label	Yes	the label of the multi user picker

Return type:

string[]

Example:

Assume we have the following script that creates a multi user picker:

```
gadget_createMultiUserPicker("MultiUserPicker", {"admin", "demouser"},
true, "Required Multi User Picker");
```

in order to retrieve the values entered in the field above we need to use the **gadget_getMultiUserPicker** routine as follows:

```
string[] res = gadget_getMultiUserPickerValue(argv, "MultiUserPicker");
//res[0] = admin
//res[1] = demouser
```

The routine has returned in this case an array of two strings, "admin" and "demouser".

gadget_getMultiValues

> **Availability**
> This routine is available since **katl-commons 3.0.8.**

Syntax:

gadget_getMultiValues(argv, label)

Description:

Retrieves the value from a multi select list or a checkbox group.

Parameters:

Parameter name	Required	Description
argv	Yes	the argv variable
label	Yes	the label of the multi select list/checkbox group

Return type:

string[]

Example:

For a script that creates a multi select list like the following

```
gadget_createMultiSelectList("Multiselect", {"a", "b", "c", "d", "e"},
{"a", "c", "e"}, true, "This field is required");
```

the selected values may be obtained with the **gadget_getMultiValues** routine:

```
string[] res = gadget_getMultiValues(argv, "Multiselect");
//res[0] = a
//res[1] = c
//res[2] = e
```

The routine has returned in this case an array of three strings, "a", "c" and "e".

gadget_getSingleValue

> **Availability**
> This routine is available since **katl-commons 3.0.8.**

Syntax:

gadget_getSingleValue(argv, label)

Description:

Retrieves the value from a text, text area, select list or radio group.

Parameters:

Parameter name	Required	Description
argv	Yes	the argv variable
label	Yes	the label of the text/text area/select list/radio group

Return type:

string

Example:

For any field that may contain only a single string type value we may use the **gadget_getSingleValue** routine. For the following script

```
gadget_createInput("Enter keyword", "demo");
```

we may obtain the field's value as in the next code sample:

```
string res = gadget_getSingleValue(argv, "Enter keyword");
//res = demo
```

gadget_isChecked

> **Availability**
> This routine is available since **katl-commons 3.0.8.**

Syntax:

gadget_isChecked(argv, label)

Description:

Retrieves the value from a checkbox.

Parameters:

Parameter name	Required	Description
argv	Yes	the argv variable
label	Yes	the label of the checkbox

Return type:

boolean

Example:

Assume we have the following parameter script:

```
gadget_createSingleCheckbox("Single Checkbox", true, false, "Required
checkbox");
```

The following call is used in the execution script to determine if the checkbox created above is checked:

```
string[] res = gadget_isChecked(argv, "Single Checkbox");
//res[0] = true
```

Scheduling Routines

Routines added by SIL Scheduler functionality

With the katl-commons 3.0.8 version, you can schedule jobs programatically.

Routine	Description	Syntax
runJobIn	Runs a job in a specified interval.	runJobIn(silFile, args, interval)
runJobInAndRepeat	Runs a job every specified interval.	runJobInAndRepeat(silFile, args, interval)
runJobAt	Runs a job at the specified date.	runJobAt(silFile, args, date)
runJobByCron	Runs the job according to the specified cron schedule.	runJobByCron(silFile, args, cronexpr)
unscheduleJob	Unschedules a job.	unscheduleJob(jobKey)
getScheduledJobKeys	Returns the list of scheduled jobs .	getScheduledJobKeys()

IMPORTANT!

1. The scheduled scripts run with no issue context and with no user. If you want to impersonate a user or run on some privileged account, use in the script the runAs routine
2. Current implementation allow you to run jobs per cluster and not per instance. If you run on only one server, all the jobs are local, but if you're running in a cluster, you need to know it. We're open to suggestions here, if you need it per instance, let us know.
3. Jobs are not persistent. This means that with every JIRA restart, you will loose the already scheduled jobs. To alleviate this, we added the startup script functionality, available only in JJUPIN.
4. Minimal amount of time for interval based schedule routines is 1s.
5. Re-scheduling a job with the same key deletes the old job and schedules the new one. Please see below for an explanation.

These routines can be used from anywhere, including Blitz Actions, KCF, KCF PRO, etc. All the above restrictions apply.

Of course, you can add jobs manually, see: SIL Services & Scheduler

Understanding uniqueness of the job

To prevent accidental overloading of the scheduler with tons of scheduled jobs, we created a very loose definition of a job key. The job key is formed from the SIL program path plus arguments. This means that you cannot, for instance, schedule the same file with the same arguments on two different intervals (or crons). You can overcome this intentional limitation by passing an extra parameter in the arguments list.

In other words:

```
string [] args = {"1"};
runJobAt("/silprograms/job.sil", args, currentDate() + "1d");
runJobAt("/silprograms/job.sil", args, currentDate() + "5d"); //will
replace the above schedule
```

while

```
string [] args1 = {"1"};
string [] args2 = {"2"};
runJobAt("/silprograms/job.sil", args1, currentDate() + "1d"); //ok,
scheduled next day
runJobAt("/silprograms/job.sil", args2, currentDate() + "5d"); //scheduled
in 5d
```

getScheduledJobKeys

> **Availability**
> This routine is available since **jjupin 3.0.8 .**

Syntax:

getScheduledJobKeys()

Description:

Returns the list of scheduled jobs .

Return type:

string[]

Returns the list of the scheduled job keys .

Example:

```
runnerLog("The job keys are : " + getScheduledJobKeys());
```

The exemple above will return the list of all scheduled job keys created.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

runJobAt

> **Availability**
> This routine is available since **jjupin 3.0.8 .**

Syntax:

runJobAt(silFile, args, date)

Description:

Runs a job at the specified date.

Parameters:

Parameter name	Type	Required	Description
silFile	string	Yes	The sil file name.
args	array string	Yes	The list of the arguments of the job.
date	date	Yes	The date

Return type:

none

The returned value has no meaning.

Example:

```
date varDate = "2015-08-20T18:30:55";
runJobAt("script.sil", {"project", "issueType"}, varDate);
```

The example above will create a job that will run the sil script at the date specified.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

runJobByCron

Availability
This routine is available since **jjupin 3.0.8 .**

Syntax:

runJobByCron(silFile, args, cronExpr)

Description:

Runs the job according to the specified cron schedule.

Parameters:

Parameter name	Type	Required	Description
silFile	string	Yes	The sil file name.
args	array string	Yes	The list of the arguments of the job.
cronExpr	string	Yes	The cron expression

Note
1.You can find more details about using the cron expression according to the used API at the link below :

www.quartz-scheduler.org/documentation/quartz-1.x/tutorials/crontrigger

Return type:

none

The returned value has no meaning.

Example:

```
runJobByCron("script.sil", {"project", "issueType"}, "0 0 12 * * ?");
```

The SIL script above will create a job that will run the script "script.sil" at 12pm (noon) every day.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

runJobIn

> **Availability**
> This routine is available since **jjupin 3.0.8 .**

Syntax:

runJobIn(silFile, args, interval)

Description:

Runs a job in a specified interval.

Parameters:

Parameter name	Type	Required	Description
silFile	string	Yes	The sil file name.
args	array string	Yes	The list of the arguments of the job.
interval	int	Yes	The interval

Return type:

none

The returned value has no meaning.

Example:

```
string[] args = {"issueType", "project"};
interval varInterval = "30m";
runJobIn("script.sil", args, interval);
```

The example above will create a job that will run the script "script.sil" after 30 minutes. (the script will run only once)

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

runJobInAndRepeat

> **Availability**
> This routine is available since **jjupin 3.0.8 .**

Syntax:

runJobInAndRepeat(silFile, args, interval)

Description:

Runs a job every specified interval.

Parameters:

Parameter name	Type	Required	Description
silFile	string	Yes	The sil file name.
args	array string	Yes	The list of the arguments of the job.
interval	int	Yes	The interval

Return type:

none

The returned value has no meaning.

Example:

```
string[] args = {"issueType", "project"};
interval varInterval = "30m";
runJobInAndRepeat("script.sil", args, varInterval);
```

The example above will create a job that will run the script "script.sil" every 30 minutes.(the job running will repeat every 30 minutes)

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

unscheduleJob

> **Availability**
> This routine is available since **jjupin 3.0.8 .**

Syntax:

unscheduleJob(jobKey)

Description:

Unschedules a job.

Parameters:

Parameter name	Type	Required	Description
jobKey	string	Yes	The job key.

Return type:

none

The returned value has no meaning.

Example 1

```
string jobKey = "C:\\Program Files\\Atlassian\\Application
Data\\JIRA6.4.5\\silprograms\\a.sil [priority project]";
unscheduleJob(jobKey);
```

The SIL script above will delete the job with the key defined by the string jobKey .

> **Note**
> 1. The value of the job key can be found in the first column of the jobs table entry from SIL Scheduler Manager or using the getScheduledJobKeys routine.

Example 2

```
for(string jobKey in getScheduledJobKeys()){
    unscheduleJob(jobKey);
}
```

The SIL script above will delete all the scheduled jobs created using SIL Scheduler or SIL Scheduler routines.

See Also:

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

Advanced features

Email Templates

When you install the plugins, in the "Administration" main menu, a new option will appear named "*Kepler General Parameters*". Selecting "katl-commons" plugin a configuration option named "*Email Template Directory*" depicts the directory where we keep the templates, relative to the **jira.home.dir/kepler**. The rest ot the email configuration is extracted from the standard JIRA configuration (Administration->Mail Servers->SMTP Mail Server).

Let me give you an example, a post-function:

```
string [] to = {"spam.receiver@kepler-rominfo.com"};

string subject = "Xmas";

string body = "template.tpl";

string cookie = "We have cookies!";

sendEmail(to, subject, body);
```

and the template will contain:

template.tpl

```
Hello $recipient$,

This is a test template sent from $assignee$'s issue $key$. The summary
for this issue is: $summary$.

$assignee$ says that $cookie$
```

In this way, we expose each variable from our script (see the 'cookie' variable) including the standard and custom fields, into the mail body.

For clarification, here's a list in my kepler dir (Linux):

```
#pwd
/opt/java/servers/jira-ee-4.3/home/kepler
# ls -lR
.:
total 56
drwxr-xr-x 2 rdumitriu rdumitriu 4096 Jun 21 16:08 emails
-rw-rw-r-- 1 rdumitriu rdumitriu  832 Jun 15 18:16 sil.properties
-rw-rw-r-- 1 rdumitriu rdumitriu  823 Jun 21 15:59 jjupin.lic
-rw-r--r-- 1 rdumitriu rdumitriu  168 Apr 1  16:42 ws-client.properties
.............................
./emails:
total 8
-rw-r--r-- 1 root root 169 Jun 21 16:08 template.tpl
```

Environment Variables

Introduction

In addition to the **local variables** (defined in a SIL program) and **issue variables** (Variable Resolution), there are **environment variables**. These are **global constants** you can access in any SIL program using the silEnv routine.

Usage

To be able to use these variables, you must follow two steps:

1. Environment variables must be defined in a special file named *sil.properties*. This file can be edited from the Administration Page in JJupin.

sil.properties

```
VAT=0.24
```

> The *sil.properties* file is located in the default folder configured in the Administration Page. If you change the path of this folder, a **new** *sil.properties* file will be created and you will have to **copy the old contents manually**.

2. Retrieve the value in the SIL program and then use it just like any other variable.

```
number VAT = silEnv("VAT");
number price = customfield_10019;
print("VAT is:" + (price * VAT));
```

See Also

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

Inclusions

Introduction

The **include** statement allows you to execute code and use UDRs that are written in other SIL files. This can improve readability and make your code more manageable.

> You can use the **include** statement to import entire libraries of UDRs.

Syntax

include "path/to/file.incl";

Topology

Include statements must be the first statements in your program, **before** the definition of UDRs.

> For more information see Structure Of A SIL Program.

Variable visibility

There are two categories of variables you can use in the **included programs**.

1. Local Variables

These are the variables you define in the body of the included program. These can be used throughout the included program, as well as in your SIL program which included the file.

file.incl

```
function increment(int x) {
  return x + 1;
}
number a = increment(0);
```

program.sil

```
include "C:/file.incl";

number b = a + increment(2);
```

Notice the use of variable **a** in the SIL program even though it was declared in the included file. Also notice the use of the **increment()** UDR whic h was defined in the included file as well.

2. Global variables

These are the variables that are already defined and can be used right away (issue fields and customfields). You can use these anywhere in your code without having to declare them.

file.incl

```
function getKey() {
  return key;
}
```

See Also

Error formatting macro: contentbylabel: com.atlassian.confluence.api.service.exceptions.BadRequestException: Could not parse cql : null

JIRA instance-independent programming

General Considerations:

Using aliases in your SIL scripts, allows you to move them from one JIRA instance to another (think testing and production) without any changes in the code. In JIRA custom field numbers are provided automatically by the custom field manager, in a sequence. That means that only if the custom fields between 2 JIRA instances are created in the same sequence one can have the same custom field number among them. Furthermore, if a custom field is deleted that number is not used anymore. For example after deleting a field by mistake, recreating the field immediately does not reuse the old id, even if the deleted field was the last one.

> Because of that the migration of SIL code between JIRA instances can pose problems.

For example one ca use a custom field number directly in code:

```
if(isNotNull(customfield_10001) // or #{The Reported User)} if accessed by
name
{
  assignee=customfield_10001; // assign the report to him or her
}
```

or, one ca use the custom fields name in code:

```
if(isNotNull(#{The Reported User}) // or customfield_10001 if accessed by
number
{
  assignee=#{The Reported User}; // assign the report to him or her
}
```

In the previous examples one uses the custom field either by it's number or by it's name.
While the second option is better than the first one because:

- one can understand the code better
- provides independence from the JIRA instance

There are still some problems:

- the second option is not independent to custom field changes
- If the custom field name is used several times by several different custom fields, the result of accessing that variable by name is unpredictable

SIL has the possibility to overcome these problems by following the next steps:
1. Create a global alias of the custom field

sil.aliases

```
# custom fields aliases
reportedUser=customfield_10001
```

2. Use the alias in the code instead

```
if(isNotNull(reportedUser)) // The Reported User
{
  assignee=reportedUser; // assign the report to him or her
}
```

Thus, the code becomes JIRA instance or custom field independent. When one needs to migrate the code needs only to ensure that the correct custom field number (or name) has been used in the *sil.aliases* file.

> Use the Administration Page from JJupin to edit the *sil.aliases* file.

Example:

First, we define some aliases for our customfields: customfield_10000 and customfield_10001 are date pickers; customfield_10002 is a userpicker

sil.aliases

```
initialDate=customfield_10000
finalDate=customfield_10001
contact=customfield_10002
```

Then we use them in our SIL program

test.sil

```
if(initialDate < currentDate() && finalDate > currentDate()){
 assignee = contact;
}
```

This is equivalent to:

test.sil

```
if(customfield_10000 < currentDate() && customfield_10001 > currentDate()){
 assignee = customfield_10002;
 }
```

Example (revisited)

In the beginning of this guide (SIL Usage) you were presented with an example to jump right into the world of SIL. If it did not make much sense then, take a look at it again, now, after reading the guide.

```
string k;

assignee = "admin";
reporter = assignee;
created = currentDate();
description = "some description";
dueDate = currentDate() + "1d";
env = "environment";
estimate = "2d" + "3h";
originalEstimate = "1d 4h" + "21h";
priority = "Critical";

if(not contains(summary, "test")){
    summary = "test " + summary;
} else {
    summary = "random summary assigned";
}

spent = "2d";
updated = currentDate() - "1d";
votes = votes + 1;
workflow = "TWFLScheme";

if(issueType == "Bug"){
    issueType = "Task";
} else {
    issueType = "Bug";
}
project = "TSTP";

//Custom fields here
```

```
UPPG = "admin";

//time interval custom field
if(isNull(tt1)) {
    tt1 = estimate + "1h";
} else {
    tt1 = tt1 + "1h";
}

//number custom field
if(isNull(cfnumber)){
    cfnumber = 1;
} else {
    cfnumber = cfnumber + 1;
}

//create routine
k = createIssue("TSTP", "", issueType, "auto-created issue");
%k%.votes = %k%.votes + 1;

//autotransition
autotransition(721,key);
```

```
//or:
//autotransition("Send report","PRJ-123");
```

Breaking Changes in v3.0

While we always try to keep the compatibility with older versions, there may be some changes that break this compatibility. Here is the list of known changes from versions 2.5/2.6 to 3.0 along with the reasoning behind them.

If you find any behavior that is not mentioned on this page and is different from version 2.5/2.6, please report it on our public issue tracker.

Code	Old	New
None that we know of.		

Additional Documentation

Tutorials

You can find out more in our extra-documentation space: Tutorials & Recipes

Plugins

Plugins using SIL

- JJUPIN 3.0
- JJUPIN Agile 3.0
- Kepler Blitz Actions 4.0
- Kepler Custom Fields 4.0
- Database Custom Field 4.0
- Kontinuum

Developer Documentation

As a first for us, we published the standard Javadoc generated documentation.

katl-commons version	Documentation URL
3.0.8 (SNAPSHOT)	https://confluence.kepler-rominfo.com/javadoc/katl-commons/3.0.8/index.html https://confluence.kepler-rominfo.com/javadoc/sil/3.0.2/index.html
3.0.4	http://confluence.kepler-rominfo.com/javadoc/katl-commons/3.0.4/index.html https://confluence.kepler-rominfo.com/javadoc/sil/3.0.1/index.html
2.6.7	http://confluence.kepler-rominfo.com/javadoc/katl-commons/2.6.7/index.html
2.5.15	http://confluence.kepler-rominfo.com/javadoc/katl-commons/2.5.15/index.html
2.5.6	http://confluence.kepler-rominfo.com/javadoc/katl-commons/2.5.6/index.html
2.5.5	http://confluence.kepler-rominfo.com/javadoc/katl-commons/2.5.5/index.html
2.5	http://confluence.kepler-rominfo.com/javadoc/katl-commons/2.5/index.html

Development using SIL

The **katl-commons** library offers a suite of features that you can use to develop your own plugins based on SIL.

- Common REST Service
- REST client
- Tutorial
- Kepler SIL support for nFeed plugin

A Foreword

Many people don't realize, but katl-commons is free software. You can use it to create your own plugins.

It is true:

1. we didn't expect to open it to the outside world, but this is what it is happening.
2. at this very moment, this doesn't come without a risk. We may change signatures of routines, alter the descriptor objects and so on. We do not have a stable API approach. Policies for the stability of the API will be introduced starting with version **2.5.5**. For sure we'll have that stable API at version 3.0 of the plugin, when we'll introduce data structures.
3. What doesn't change is the script interpretation. You should base your code on the assumption that the interpreted script will yield the same results.

Developer Documentation

Starting with version 2.5, we'll publish the standard Javadoc generated documentation pages.

katl-commons version	Documentation URL	Obs
3.0.8 / SIL **3.0.2** **(SNAPSHOTS)**	https://confluence.kepler-rominfo.com/javadoc/katl-commons/3.0.8/index.html https://confluence.kepler-rominfo.com/javadoc/sil/3.0.2/index.html	JIRA 6.x Most notable change is at the Date representation.
3.0.4 / SIL **3.0.1**	https://confluence.kepler-rominfo.com/javadoc/katl-commons/3.0.4/index.html https://confluence.kepler-rominfo.com/javadoc/sil/3.0.1/index.html	JIRA 6.x
2.6.7	http://confluence.kepler-rominfo.com/javadoc/katl-commons/2.6.7/index.html	All changes, including those made for performance / JIRA 6
2.5.15	http://confluence.kepler-rominfo.com/javadoc/katl-commons/2.5.15/index.html	All changes, including those made for performance / JIRA 5
2.5.6	http://confluence.kepler-rominfo.com/javadoc/katl-commons/2.5.6/index.html	Minor changes from the 2.5.5; specifically the dates routines have been improved a bit.
2.5.5	http://confluence.kepler-rominfo.com/javadoc/katl-commons/2.5.5/index.html	This is the first version when we opened some implementations to general usage, specifically: 1. the custom field implementation classes 2. the REST utilities 3. Licensing support was moved out in Warden plugin 4. Redirect added (for those in need) If you will develop something based on katl-commons, you should refer to this version as a minimal starting point.
2.5	http://confluence.kepler-rominfo.com/javadoc/katl-commons/2.5/index.html	Initial 2.5 version, lots of things still missing

Common REST Service

> **Availability**
> Available since katl-commons 2.5.5.

Introduction

Since version 2.5.5, katl-commons exposes a REST service to facilitate arbitrary execution of SIL scripts. These services are available via HTTP POST at **<your_base_url>/rest/keplerrominfo/commons/latest/api**. The service responses are all JSON formatted (see Beans below).

You will notice that most of the methods will schedule a task for execution and will provide a unique identifier for the task. You will have to call getResults and provide the identifier to check the status of the task (running or finished) and retrieve the results.

> **Authentication**
> Note that all the services require that the calling user is authenticated. You can use Basic authentication with your HTTP request.

Services

Method	Full Path	Availability
findFiles	<your_base_url>/rest/keplerrominfo/commons/latest/api/findFiles	2.5.5+
checkScript	<your_base_url>/rest/keplerrominfo/commons/latest/api/checkScript	2.5.5+
checkFileScript	<your_base_url>/rest/keplerrominfo/commons/latest/api/checkFileScript	2.5.5+
runScript	<your_base_url>/rest/keplerrominfo/commons/latest/api/runScript	2.5.5+
runFileScript	<your_base_url>/rest/keplerrominfo/commons/latest/api/runFileScript	2.5.5+
getResult	<your_base_url>/rest/keplerrominfo/commons/latest/api/getResult	2.5.5+

findFiles

Allows you to scan a folder for specific files for which the filename matches a specific regular expression.

Parameter	Type	Required	Description
dirPath	string	yes	The path of the directory to scan
regex	string	yes	The regular expression to match the filenames against

Returns a FindFilesResponse.

Example Request using AJS

```
AJS.$.ajax({
 type: 'POST',
 url :
"http://localhost:8522/rest/keplerrominfo/commons/latest/api/findFiles",
 data : {
  dirPath : "D:/test",
 regex : "[^.]*\.sil"
 },
 success : function(data) {
 console.log(data);
 },
 beforeSend: function (xhr){
  xhr.setRequestHeader('Authorization', "Basic " + btoa(username + ":" +
password));
 }
});
```

checkScript

Schedules a check script task to be run asynchronously for the given script and returns a ScheduleResponse.

Parameter	Type	Required	Description
script	string	yes	The script to check

Example Request using AJS

```
AJS.$.ajax({
 type: 'POST',
 url :
"http://localhost:8522/rest/keplerrominfo/commons/latest/api/checkScript",
 data : {
 script : "return 1;"
 },
 success : function(data) {
 console.log(data);
 },
 beforeSend: function (xhr){
 xhr.setRequestHeader('Authorization', "Basic " + btoa(username + ":" +
password));
 }
});
```

checkFileScript

Schedules a check script task to be run asynchronously for the script provided as a file on the server side and returns a ScheduleResponse.

Parameter	Type	Required	Description
scriptPath	string	yes	The server-side path to the file containing the script.

Example Request using AJS

```
AJS.$.ajax({
 type: 'POST',
 url :
"http://localhost:8522/rest/keplerrominfo/commons/latest/api/checkFileScript",
 data : {
 scriptPath : "D:/test/test_1234567890.sil"
 },
 success : function(data) {
  console.log(data);
 },
 beforeSend: function (xhr){
  xhr.setRequestHeader('Authorization', "Basic " + btoa(username + ":" +
password));
 }
});
```

runScript

Schedules a run script task to be run asynchronously for the given script and returns a ScheduleResponse.

Parameter	Type	Required	Description
issueKey	string	no	The issue key, if you want the script to be run in an issue context.
script	string	yes	The script to run
args	string	no	Any additional parameters to be sent in the argv variable.

Example Request using AJS

```
AJS.$.ajax({
 type: 'POST',
 url :
"http://localhost:8522/rest/keplerrominfo/commons/latest/api/runScript",
 data : {
 issueKey : "TEST-1",
  script : "return key, argv[0];",
 args : ["testing"]
 },
 success : function(data) {
  console.log(data);
 },
 beforeSend: function (xhr){
  xhr.setRequestHeader('Authorization', "Basic " + btoa(username + ":" +
password));
 }
});
```

runFileScript

Schedules a run script task to be run asynchronously for the script provided as a file on the server side and returns a ScheduleResponse.

Parameter	Type	Required	Description
issueKey	string	no	The issue key, if you want the script to be run in an issue context.
scriptPath	string	yes	The server-side path to the file containing the script to run.
args	string	no	Any additional parameters to be sent in the argv variable.

Example Request using AJS

```
AJS.$.ajax({
 type: 'POST',
 url :
"http://localhost:8522/rest/keplerrominfo/commons/latest/api/runFileScript",
 data : {
  issueKey : "TEST-1",
 scriptPath : "D:/test/test_1234567890.sil",
 args : ["testing"]
 },
 success : function(data) {
 console.log(data);
 },
 beforeSend: function (xhr){
  xhr.setRequestHeader('Authorization', "Basic " + btoa(username + ":" +
password));
 }
});
```

getResult

Checks if the task identified by the provided id is still running or has finished and returns an ExecutionResponse. If the task has finished execution, the result is removed from memory and returned in the response. Subsequent calls to this method will not return the result.

Parameter	Type	Required	Description
key	int	yes	The unique key identifying a scheduled task.

Example Request using AJS

```
AJS.$.ajax({
 type: 'POST',
 url :
"http://localhost:8522/rest/keplerrominfo/commons/latest/api/getResult",
 data : {
  key : 4
 },
 success : function(data) {
  console.log(data);
 },
 beforeSend: function (xhr){
  xhr.setRequestHeader('Authorization', "Basic " + btoa(username + ":" +
password));
 }
});
```

Beans

FindFilesResponse

Field	Type	Description
files	string []	List of absolute paths
error	string	The error, if any occurred

ExecutionResponse

Field	Type	Description
isRunning	boolean	Signals that the script is still running
error	string	The error, if any
isOk	boolean	Signals that the script has finished execution successfully
results	string []	The results returned by the script

ScheduleResponse

Field	Type	Description
key	int	The unique key of the scheduled task
error	string	The error, if any

CURL Example

The following curl calls show how you would call SIL scripts from your favourite system (that's Linux, I hope). The commands assume that username / password is 'admin'

Notice the way you pass multiple parameters to the SIL scripts. The logic is the same, you post the script to start running asynchronously, then you come back later for the results.

```
$curl -u admin:admin -d "issueKey=TST-1" -d "script=return key,
argv[0],argv[1];" -d "args=var1" -d "args=var2"
http://127.0.0.1:6470/rest/keplerrominfo/commons/latest/api/runScript
{"key":10}

$curl -u admin:admin -d "key=10"
http://127.0.0.1:6470/rest/keplerrominfo/commons/latest/api/getResult
{"isRunning":false,"isOk":true,"results":["TST-1","var1","var2"]}
```

REST client

::TODO::

Tutorial

Introduction

This tutorial shows you how to create a simple routine and how to offer support for an unknown custom field.

You only need some Java experience to do it, but we believe this tutorial is simple enough to realize how easy is to add custom stuff into your

katl-commons plugin.

Creating A New SIL Routine

Creating A New SIL Routine

Problem:

We need to reverse certain strings within JIRA.

> **Source code**
> Full source code is available here: The Simple Example Sources

The solution:

First, create a standard Maven project with your favorite IDE.

Create a package (***com.mycompany.silexample***) and in this package create a class (we'll name it ***ReverseStringRoutine***)

The code is presented below, along with the comments:

```
/*
 * Created at Sep 2, 2011T5:39:27 PM+02.
 *
 * File: ReverseStringRoutine.java
 */
package com.mycompany.silexample;
import java.util.*;
import com.keplerrominfo.sil.lang.*;
import com.keplerrominfo.sil.lang.type.TypeInst;
import com.keplerrominfo.sil.lang.value.SILValue;
import com.keplerrominfo.sil.lang.value.SILValueFactory;

/**
 * Reverses a string
 *
 * @author Radu Dumitriu (rdumitriu@gmail.com)
 * @since 1.0
 */
public class ReverseStringRoutine extends AbstractRoutine<Object> {
    private static final SILType[][] types = {{ TypeInst.STRING }};
    public ReverseStringRoutine() {
        super("myReverseString", types);
    }

    /**
     * Returns the type of the returned value. For routines that return
     * a value of variable type, this should return null
     *
```

```
     * @return the type of the returned value
     */
    @Override
    public SILType getReturnType() {
        return TypeInst.STRING;
    }

    /**
     * The execution of the routine
     * @param silValues the list of values (parameters)
     * @return the SIL value
     */
    @Override
    protected SILValue executeRoutine(SILContext context, List<SILValue>
silValues) {
        //AbstractRoutine checks the parameters and their types
        SILValue param = silValues.get(0);
        //We know for sure this is a string
        String val = param.toStringValue();
        //we calculate the reversed value
        String reversedVal = new StringBuilder(val).reverse().toString();
        //We'll prepare the return here
        return SILValueFactory.string(reversedVal);
    }

    /**
     * This returns the description of the parameters
     * @return the part that will be appended to the routine at editing
time
     */
    @Override
    public String getParams() {
        return "(str)"; //that's all
    }
```

```
}
```

Our routine is now completed, we need to register it on the SIL. For that, we'll use the standard OSGI bundle mechanism, so let's write an OSGI bundle activator:

```
package com.mycompany.silexample;

import com.keplerrominfo.jira.commons.ivm.UserRoutineRegistry;

/**
 * This is the bundle activator. Since it's OSGI, it will boot in the same
 * directory as any JIRA plugin (JIRA_HOME/plugins/installed-plugins/)
 *
 * @since 1.0
 */
public class BundleActivator implements org.osgi.framework.BundleActivator
{
 /**
  * Default constructor
  */
 public BundleActivator() {
 }
 /**
  * Standard OSGI startup hook
  * @param bundleContext the bundle context
  */
 public void start(org.osgi.framework.BundleContext bundleContext) {
   //register the routine
   RoutineRegistry.register(new ReverseStringRoutine());
 }
 /**
  * Standard OSGI shutdown hook
  * @param bundleContext the bundle context
  */
 public void stop(org.osgi.framework.BundleContext bundleContext) {
   //does nothing
 }
}
```

The coding part is now completed. Now all we have to do is to create a package, but before that we need to write the Maven pom.xml (see The Simple Example Sources).

Ok, now you must run "**mvn package**" and you're ready to install it via **Administration > Add-ons > Manage Add-ons > Upload add-on** and off you go!

We hope at least you got an idea on how easy it is to extend the SIL language.

> For additional info, project javadoc and any kind of assistance, please contact us.
>
> Check the Additional Documentation page for javadocs.

Under the hood (some explanations)

The above routine inherits a lot of the functionality from *AbstractRoutine* (http://confluence.kepler-rominfo.com/javadoc/katl-commons/2.5/com/keplerrominfo/jira/commons/sil/AbstractRoutine.html). The *AbstractRoutine* class is responsible for:

1. Checking the type of the parameters
2. Converting automatically the parameters to the required types
3. Calling the real execution code with the parameters transformed.

The *AbstractRoutine* class follows in fact a Template Method Pattern.

You are not required to inherit *AbstractRoutine*. In fact you should sometimes implement Routine interface (http://confluence.kepler-rominfo.com/javadoc/katl-commons/2.5/com/keplerrominfo/jira/commons/sil/Routine.html) to avoid or implement custom checking of the parameters.

Don't forget to:

- register the Kepler jars in your **maven repository** before building your OSGI module (If you don't, **maven** will complain about it and will tell you how).
- unit test your code (the above example lacks it, but you should feel obliged to do it).

Creating a Custom Field Descriptor

Problem:

We need to provide support for some custom field. Our example is based on the previous example (check Creating A New SIL Routine).

By default, JJupin does offer support for the project picker CF, but for educational purposes, let's see how you could add the support if it didn't.

Source code
Full source code is available here: The Simple Example Sources

The solution:

The problem is just to register a translator from SILValue to the native Object and the other way around.

The code is presented below:

ProjectCFDescriptor.java

```
/*
 * Created at: 3/28/13, 3:02 PM
 *
 * File: ProjectCFDescriptor.java
 */
package com.mycompany.silexample;
import com.atlassian.jira.project.Project;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.ofbiz.core.entity.GenericValue;
import com.keplerrominfo.common.util.MutableString;
import com.keplerrominfo.common.util.StringUtils;
import com.keplerrominfo.jira.commons.ivm.FieldDescriptor;
import com.keplerrominfo.jira.commons.jira.JiraManagers;
import com.keplerrominfo.sil.lang.SILType;
```

```
import com.keplerrominfo.sil.lang.type.TypeInst;
import com.keplerrominfo.sil.lang.value.SILValue;
import com.keplerrominfo.sil.lang.value.SILValueFactory;
/**
 * The project CF descriptor,
com.atlassian.jira.plugin.system.customfieldtypes:project
 *
 * @author Radu Dumitriu (rdumitriu@gmail.com)
 */
public class ProjectCFDescriptor implements FieldDescriptor<GenericValue,
MutableString> {
    private static final Log LOG =
LogFactory.getLog(ProjectCFDescriptor.class);
    @Override
    public SILType<MutableString> getType() {
        return TypeInst.STRING;
    }
    @SuppressWarnings("deprecation")
    @Override
    public GenericValue toJiraValue(SILValue<MutableString> value) {
        String projectKey = StringUtils.trim(value.toStringValue());
        if (projectKey == null) {
            return null;
        }
        Project project =
JiraManagers.getProjectManager().getProjectObjByKey(projectKey);
        if(project == null) {
            LOG.warn(String.format(">>%s<< is not a valid project key",
projectKey));
            return null;
        }
        // Project picker CF stores the value as GenericValue
        return project.getGenericValue();
    }
    @Override
    public SILValue<MutableString> toSILValue(GenericValue value) {
        return value != null ?
SILValueFactory.string(value.getString("key")) : SILValueFactory.string();
    }
}
```

You will need to update your Bundle Activator in this way:

BundleActivator.java

```
/*
 * Created at Sep 2, 2011T4:45:45 PM+02.
 *
 * File: BundleActivator.java
 */
package com.mycompany.silexample;

import com.keplerrominfo.jira.commons.ivm.CustomFieldDescriptor;
import com.keplerrominfo.jira.commons.sil.RoutineRegistry;

/**
 * This is the bundle activator. Since it's OSGI, it will boot in the same
 * directory as any Jira plugin (JIRA_HOME/plugins/installed-plugins/)
 *
 * @author Radu Dumitriu (rdumitriu@gmail.com)
 * @since 1.0
 */
public class BundleActivator implements org.osgi.framework.BundleActivator
{

 /**
 * Default constructor
 */
 public BundleActivator() {
 }

 /**
  * Standard OSGI startup hook
  * @param bundleContext the bundle context
  */
 public void start(org.osgi.framework.BundleContext bundleContext) {
   //register the routine
   RoutineRegistry.register(new ReverseStringRoutine());
   CustomFieldDescriptorRegistry

.register("com.atlassian.jira.plugin.system.customfieldtypes:project",
                        new AbstractCustomFieldDescriptorFactory(
                                "example.project.cfdf",
                                "Project Key to String",
                                "Extract the project key using
GenericValue of project") {
                            @Override
                            public FieldDescriptor createDescriptor(Issue
issue, CustomField cf) {
```

```
                                return new ProjectCFDescriptor();
                            }
                        }
        );
}


/**
* Standard OSGI shutdown hook
* @param bundleContext the bundle context
*/
public void stop(org.osgi.framework.BundleContext bundleContext) {
  //does nothing
}
```

```
}
```

Line 33 registers the CF into our CF translators map. Hooray !

That's all you need. You can now play with SIL like such:

something.sil

```
summary += " -- " + KProject; //where KProject is your CF name !
```

The Simple Example Sources

In the above examples (Creating A New SIL Routine / Creating a Custom Field Descriptor) we have shown how you should add support for new routines and custom fields.

This page contains the full example, packed in zip.

- example.zip

Have fun in creating your own routines, and, if you think your routine should stay in core SIL, please mail us; we're happy to add them.

Kepler SIL support for nFeed plugin

Availability
This tutorial assumes you use a minimal **katl-commons 3.0**. There were important changes in this version, and prior versions would require more work.

Kepler is not affiliated or in commercial relation(s) with the provider of the **nFeed** plugin (Valiantys), other than those imposed by the general terms at Atlassian (we're both plugin providers). In other words, this is totally **unsupported**, wacky tweak, can harm you nFeed plugin if not used properly (values on set need to really exist in the DB!), may change in the future (we do not plan support for that).

In fact, you should insert here all the possible warnings you can think of. Just don't blame us.

Introduction

This plugin is used to offer **SIL** support for the **SQL Feed Custom Field** offered by the **nFeed Jira plugin**. The **nFeed** plugin connect your data to **JIRA** issues by creating custom fields with data from remote files, web services and databases, multi-selecting values from your data sources, creating infinite cascading selections, integrating the notion of conditional requests within **JIRA**'s custom fields. This page show you how this custom field is handled by our Kepler plugins and especially by **SIL**.

Solution

First of all we needed to create a CF Descriptor that allow us to handle the values that the nFeed Custom Field accepts and returns:

NFeedCFDescriptor

```
/*
 * File: NFeedCFDescriptor.java
 */
package com.keplerrominfo.jira.plugins.keplernfeed.nfeedcf;
import java.util.Collection;
import java.util.List;
import com.atlassian.jira.issue.Issue;
import com.atlassian.jira.issue.fields.CustomField;
import com.keplerrominfo.common.util.MutableString;
import com.keplerrominfo.jira.commons.ivm.AbstractCustomFieldDescriptor;
import com.keplerrominfo.sil.lang.*;
import com.keplerrominfo.sil.lang.type.TypeInst;
import com.keplerrominfo.sil.lang.value.SILValue;
import com.keplerrominfo.sil.lang.value.SILValueFactory;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
/**
 * The nFeed CF descriptor for nFeed plugin
 *
 * @author Florin Manaila (florin.manaila@kepler-rominfo.com)
 * @author Alexandru Iacob (alexandru.iacob@kepler-rominfo.com)
 * @since 1.0
 * @version 3.0.0
 */
public class NFeedCFDescriptor extends
AbstractCustomFieldDescriptor<Object, KeyableArraySILObject<MutableString>>
{

 private static final Log LOG = LogFactory.getLog(NFeedCFDescriptor.class);

 /**
  * Constructor
  * @param issue
  * @param customField
  */
 public NFeedCFDescriptor(Issue issue, CustomField customField) {
  super(issue, customField);
 }
 /**
  * @see com.keplerrominfo.jira.commons.ivm.FieldDescriptor#getType()
  */
 @Override
 public SILType<KeyableArraySILObject<MutableString>> getType() {
  return TypeInst.STRING_ARR;
 }
 /**
  * @see com.keplerrominfo.jira.commons.ivm.FieldDescriptor
```

```
  * #toJiraValue(com.keplerrominfo.sil.lang.value.SILValue)
  */
 @Override
 public Object toJiraValue(SILValue<KeyableArraySILObject<MutableString>>
value) {
  ClassLoader nfCl =
getCustomField().getCustomFieldType().getClass().getClassLoader();
  Object ret = null;
  try {
   Class clazz =
nfCl.loadClass("com.valiantys.jira.plugins.sql.customfield.SQLFeedContent"
);
   ret = clazz.getConstructor().newInstance();
   clazz.getDeclaredMethod("setValues", List.class).invoke(ret,
value.toStringArray());
  } catch (Exception e) {
   String msg = String.format("Could not convert %s to nFeed value",
          value.toString());
   LOG.error(msg, e);
   throw new SILException(msg, e);
  }
  return ret;
 }
 /**
  * @see com.keplerrominfo.jira.commons.ivm.FieldDescriptor
  * #toSILValue(java.lang.Object)
  */
 @Override
 public SILValue<KeyableArraySILObject<MutableString>> toSILValue(Object
value) {
  SILValue<KeyableArraySILObject<MutableString>> ret =
        SILValueFactory.stringArray((Collection<String>)value);
  if(LOG.isDebugEnabled()){
   LOG.debug(String.format("Translated (JIRA) %s to (SIL) %s",
         value, ret.toString()));
  }
  return ret;
 }
```

```
}
```

This class extends the abstract class AbstractCustomFieldDescriptor form **Katl-commons plugin**. Due to the fact that the nFeed CF accepts only SQLFeedContent objects in order to set the value of this custom field we needed to get this type through **Java Reflection**. The method toJiraValue() is overridden in order to provide a converter from the SIL Value to the objects that the nFeed CF supports.

The values returned by the nFeed CFs are in fact Collection<String> and this is why we are able to create the SIL value for this CF type in a simple manner only by calling the SILValue constructor.

Also, we implemented a customized Bundle Activator class like such:

BundleActivator

```
/**
 * Apr 2, 2013, 3:39:06 PM
 *
 * KplerNFeedBundleActivator.java
 */
package com.keplerrominfo.jira.plugins.keplernfeed.admin;
import java.util.Map;
import org.osgi.framework.BundleContext;
import com.atlassian.jira.issue.Issue;
import com.atlassian.jira.issue.fields.CustomField;
import com.keplerrominfo.common.util.MutableString;
import com.keplerrominfo.jira.commons.ivm.*;
import
com.keplerrominfo.jira.commons.jira.kconfig.AbstractPluginLifecycleActivat
or;
import
com.keplerrominfo.jira.plugins.keplernfeed.nfeedcf.NFeedCFDescriptor;
import com.keplerrominfo.sil.lang.KeyableArraySILObject;
/**
 * Bundle Activator for Kepler nFeed plugin.
 * @author Alexandru Iacob (alexandru.iacob@kepler-rominfo.com)
 * @since 1.0
 * @version 3.0.0
 */
public class KeplerNFeedBundleActivator extends
AbstractPluginLifecycleActivator {
 public static final String PLUGIN_NAME = "Kepler SIL nFeed Support";

 private static final String NFEED_CF_KEY =

"com.valiantys.jira.plugins.SQLFeed:com.valiantys.jira.plugins.sqlfeed.cus
tomfield.type";

    private static final CustomFieldDescriptorFactory<Object,
KeyableArraySILObject<MutableString>> NFEED_CFDF =
      new AbstractCustomFieldDescriptorFactory<Object,
KeyableArraySILObject<MutableString>>(
                  "sil.adapters.cf.nfeed",
                  "nFeed -> string[]",
```

```
                  "Translates to an array of string values") {
        @Override
        public FieldDescriptor<Object,
KeyableArraySILObject<MutableString>> createDescriptor(Issue issue,
                            CustomField cf) {
            return new NFeedCFDescriptor(issue, cf);
        }
    };
 /**
  * @see
com.keplerrominfo.jira.commons.jira.kconfig.AbstractPluginLifecycleActivat
or
  * #getConfigurationValues()
  */
 @Override
 protected Map<String, String> getConfigurationValues() {
  return null;
 }
 /**
  * @see
com.keplerrominfo.jira.commons.jira.kconfig.AbstractPluginLifecycleActivat
or
  * #getPluginName()
  */
 @Override
 protected String getPluginName() {
  return PLUGIN_NAME;
 }

 /**
  * @see
com.keplerrominfo.jira.commons.jira.kconfig.AbstractPluginLifecycleActivat
or
  * #start(org.osgi.framework.BundleContext)
  */
 @Override
 public void start(BundleContext bundleContext) {
  super.start(bundleContext);
  //register the custom field descriptor
  CustomFieldDescriptorRegistry.register(NFEED_CF_KEY, NFEED_CFDF);
 }

 /**
  * @see
com.keplerrominfo.jira.commons.jira.kconfig.AbstractPluginLifecycleActivat
or
  * #stop(org.osgi.framework.BundleContext)
  */
 @Override
 public void stop(BundleContext bundleContext) {
  super.stop(bundleContext);
  CustomFieldDescriptorRegistry.unregister(NFEED_CF_KEY);
 }
```

```
}
```

The line 65 registers the nFeed CF into our CF translators map.

That is all. You can use this CF in your SIL programs by typing the name of your nFeed custom field - like any other JIRA CF accepted by SIL.

Example

```
string[] array = nFeed; //where nFeed is your CF name !
print(array[0]);
```

Binaries and Sources

Ok, now that you read it through, here they are: the binary (jar) and the sources, packed as zip

- keplernfeed-3.0.0-J6-SNAPSHOT.jar - the binary, ready to be deployed
- keplernfeed.zip - sources, zipped

Additional Features

This page contains other features provided by katl-commons that are not necessarily related to SIL.

Configuration

Katl-commons offers to the JIRA administrator some configuration pages (well, we tried hard to improve them, we just hope you like the result).

These pages are described here, in our flagship product space, JJupin:

Page Reference	What you can configure	Link
General configuration	charset, caching of the scripts, mails	http://confluence.kepler-rominfo.com/display/JJUP30/SIL+Configuration http://confluence.kepler-rominfo.com/display/JJUP30/Mail+Configuration
Remote systems	REST / SOAP remote systems	http://confluence.kepler-rominfo.com/display/JJUP30/Remote+Systems 1. http://confluence.kepler-rominfo.com/display/JJUP30/REST+Remote+Systems 2. http://confluence.kepler-rominfo.com/display/JJUP30/SOAP+Remote+Systems
-	Additional datasources	http://confluence.kepler-rominfo.com/display/JJUP30/SQL+Configuration
LDAP parameters	LDAP lookup	http://confluence.kepler-rominfo.com/display/JJUP30/LDAP+Configuration
-	logging	http://confluence.kepler-rominfo.com/display/JJUP30/Configure+JIRA+Logging

KRedi - The Redirect Assistant

Availability
This feature is available since katl-commons 2.5.5.

When integrating JIRA with other external systems, it often happens that we need to provide access from those systems directly to JIRA, while

not being fully aware of the contents of the JIRA instance. For this purpose, we designed KRedi, The Redirect Assistant.

The idea behind **KRedi** is that we can build a custom URL, containing some parameters and pointing to our JIRA instance. We then take these parameters and pass them into a SIL script that generates a valid JIRA URL. This allows us to create a one-way dependency between the two systems, since the external system has no knowledge of the contents of our JIRA instance.

Accessing KRedi

External access into our JIRA instance will be provided as a simple URL: ***<base_url>/secure/KRedi.jspa***

Accessing this URL will trigger the script and redirect to whatever the script returns.

Writing the SIL Script

Once you have updated katl-commons to at least 2.5.5, the **Kepler General Parameters** (Administration -> Kepler General Parameters) page should provide a new tab: **Redirect**. Here you will be presented with the already familiar SIL Editor, where you can create the script.

The script works by **returning a string value**, which represents a context-relative URL.

Example Redirect Script

```
return "/browse/TEST";
```

The above script will automatically take care to append the context path to the generated URL. Consequently, it will also work if you access JIRA via *mydomain.com* or *mydomain.com/myjira.*

Passing Parameters

It's not really useful to not have any control from the external system, so in order to provide some guidance to the script, we can pass in some parameters in the URL: *<base_url>/secure/KRedi,jspa****?param1=value1&param2=value2****.*

These parameters will be passed into the SIL Script via the **argv** variable, and are accessible using the indexing operator. (e.g. **argv["param1"]**)

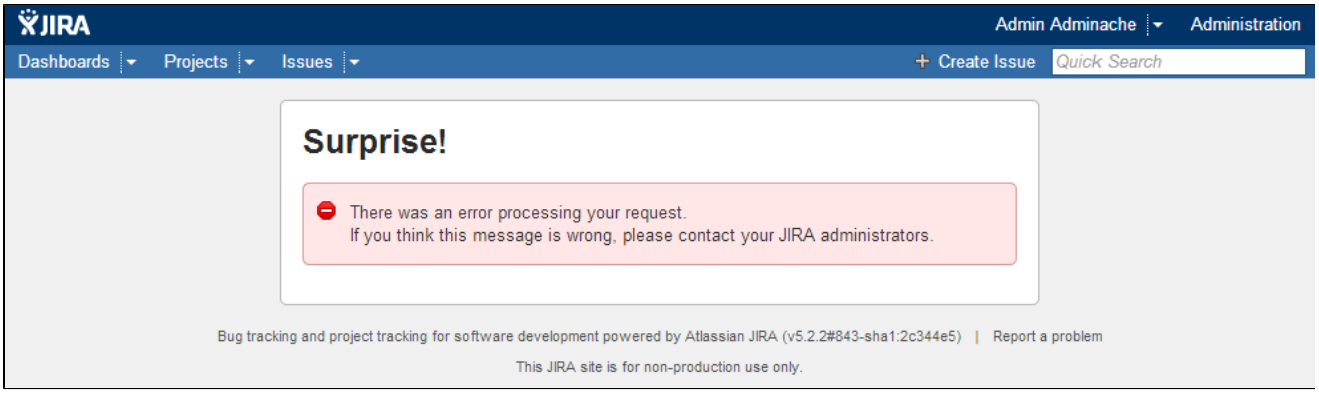
Example

```
// Let's assume that our external system generates URLs in the form of
mydomain.com/secure/KRedi.jspa?externalId=1234
// and that we generate the URL based on that id
number extId = argv["externalId"];
string url;
// generate the url here based on the extId
return url;
```

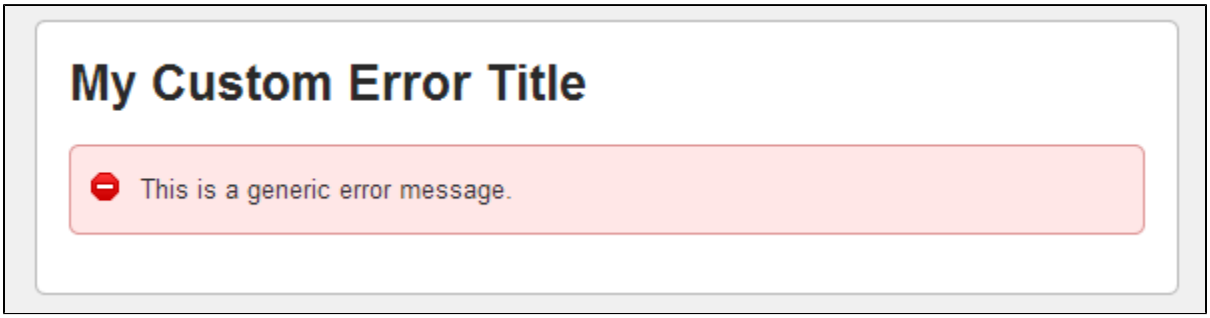
Returning Errors

What if the script needs a certain parameter to do the redirect, but the user does not provide it? Where should we redirect to?

KRedi also provides an error page accessible at */secure/KRedi!error.jspa* which contains a generic error message.



If you would like to customize the error message or the error title, you can specify them in the URL using the **customErrorMessage** and/or **customErrorTitle** parameters. (e.g. */secure/KRedi!error.jspa?**customErrorTitle**=My+Custom+Error+Title&**customErrorMessage**=This+is+a+generic+error+message.*)



Note
Note that you must URL-encode the text.

That's all there is to it! Easy!

Getting started

Simple Issue Language

SIL (Simple Issue Language) is an easy to learn scripting language (see SIL syntax) that is used by Kepler plugins.

Example

For example with SIL and JJupin plugin you can create a postfunction that automatically creates a sub-task.

```
string projectKey= "TSTP";

string issueType = "Sub-task";
string issueSummary = "Issue created using SIL";
//create routine
issue = createIssue(projectKey, key, issueType, issueSummary);
```

See more examples here.

Syntax

For the language syntax, check out the syntax page.

JIRA standard variables

SIL has a list of standard variables which you can use to change issues fields.

For example, to change the issue description you can use the standard variable description:

```
description = "Issue description";
```

Learn more about standard variables here.

Routines

SIL comes with a library of standard routines to use in scripts. There are routines for handling arrays, strings, dates and there are specific plugin routines.

Learn more about SIL routines here.

Plugins

Add-on name	Product page	Documentation	Usage
JJupin (commercial)	JJupin product page	JJupin Documentation	• Write workflow conditions, validators or post-functions • Write (Live Fields) scripts for client browser scripting (avoiding java script!). One can intercept create, edit, view, transition screens. • Write scripts to run regularly to be used in SIL Services • Write event listening SIL scripts • Write scripts to be run on user request with SIL Script Runner
Blitz Actions (commercial)	BA product page	Kepler Blitz Actions Documentation	• Highly customizable dialog-based conversation inside JIRA. • Write custom actions to be called within issue view screen • Contains all required steps: conditions, screen definition, data validation and action itself
Kepler Custom Fields (free)	KCF product page	Kepler Custom Fields Documentation	• SIL Script Custom Field uses the language to define calculated custom fields • Interval CF and Regex CF may be used by other plugins (read/write)
Kepler Custom Fields Pro (commercial)	KCFPro product page	Kepler Custom Fields Pro Documentation	• Custom fields with data populated from a SIL script • Execute a SIL script when the custom field is modified
Database Custom Field (free)	DBCF product page	Database Custom Field Documentation	• Data Table Custom Field can be configured to display data from a SIL script data source • Plugin provides routines to manipulate the data from Data Table CF programmatically • Database Custom Fields may be used in SIL scripts (read/write)

KIWI (commercial)	KIWI product page	KIWI Documentation	• Export/ import workflows • Export/ import workflow related screens, issue types, custom fields
SIL Excel Reporting (commercial)	SIL Excel Reporting product page	SIL Excel Reporting Documentation	• Offers reporting capabilities to JIRA • Write reports using SIL scripts to excel files
User Group Picker PRO (commercial)	User Group Picker PRO page	User Group Picker PRO Documentation	• SIL User Picker CF uses SIL scripting for retrieving restricted users in a Jira style user picker field

workHoursFromCalendar

Availability
This routine is available since

1. **Kontinnum 2.0.2**
2. **katl-commons 2.5**

Syntax:

workHoursFromCalendar (username, start_date, end_date)

Description:

Returns the number of hours that a user can log for his work during a certain period of time.

This routine computes the number of hours that a **JIRA** user can work during the time period since start_date until end_date.

It uses the user calendar in order to detect the weekends, holiday or unpaid leaves (or any type of **Exceptions**) during the selected period - these exceptions will not be counted in the result.

Link
For more information about this routine you could consult the page: workHoursFromCalendar.