JavaScript in IBPM

The JavaScript programming language is used extensively in IBPM solutions. JavaScript can be seen to be used in service definitions and also in expression evaluations. Understand that the JavaScript being executed here is executed at the server side and **not** within a browser. A common mistake I have seen is working in JavaScript on the server side and then working in JavaScript in the browser and mistakenly thinking that in both cases, code runs in the same environment.

In addition, JavaScript can be defined as the direct implementation type for a BPD activity. This might be useful for testing or quick hacks but is unlikely to be a good long term practice. There is no reuse available for code implemented in this fashion.

It is believed that a knowledge of JavaScript will be essential to long term construction of IBPM solutions. Although simple assignments can be performed without detailed JavaScript skills, more subtle activities such as defining variables, loops and other language constructs will require skills. Fortunately, JavaScript is not a complex language to learn and can be picked up quickly especially if you already know the Java programming language.

See:

Editing JavaScript

The editing of the JavaScript code should be done within IBPM PD. Each of the JavaScript editor areas provided in PD has content assist and syntax checking. Unfortunately, the content assist doesn't address all the possible content and the syntax checking assistance sometimes indicates perfectly valid and correct JavaScript as containing warnings or errors (on occasion) so although both these functions are useful, don't rely on them 100% of the time. The fact that the assistance features are present is probably better than them not being present but don't panic if a variable you think should be in scope doesn't show up in the content assist or a statement is flagged as in error when no amount of review shows a problem … simply ignore and move on as though the assistance tools were not present.

When the JavaScript entry assistance does let us down, we can get into quite a mess. JavaScript is a loosely typed language. As such, declaration of variables with types can't be easily checked at editing time (by PD). Making a typo in JavaScript code is very common as is calling a function by the wrong name. Debugging and correcting such errors can be frustrating. For example, to find a Process Instance by its ID, we call the following function:

tw.system.findProcessInstanceByID()

There are many opportunities here to make an error that won't be caught. I had mistakenly coded:

tw.system.findProcessInstanceBy**Id**()

and it took me far too long to find that mistake.

IBPM JavaScript name spaces

IBPM exposes a number of JavaScript name spaces which contain various data items.

`tw.local.*` Variables within the current process/service.

The JavaScript language doesn't have the notion of namespaces so this concept is provided by properties of objects. For example, the notion of "tw" is actually a JavaScript object which has a property called "local". This property is itself an object.

Reusing JavaScript

Commonly used JavaScript functions can be added to a Java Script source file (a file ending in "`.js`"). This file can then be added as a managed file in a Process Application. The Managed File has to be of type "Server". Once added, any JavaScript found within becomes part of the environment when running other JavaScript routines. This becomes especially useful if the JavaScript source file contains function declarations. These can then be used in other Java Script code fragments within the solution without any further work being needed. Although not investigated yet, it is believed that the re-use of the JavaScript code by adding it as server managed files needs to be carefully managed to prevent name collisions since the scope of Server Managed files appears quite large. It is currently seen that the Java Script editor flags the use of such functions as a warning as they are said to be unknown functions. However, they can be run just fine.

BPM Data Types

The IBPM environment utilizes and provides some pre-existing data types. Understanding these data types is useful in advanced situations. Many of these data types have a method on them called `getTypeName()` which when called will return a String representation of the data type being examined.

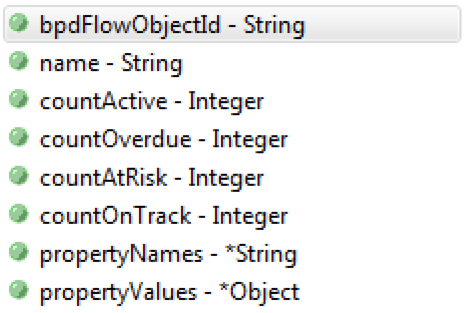
Data Type – ActivityDetails

Property	Type
actions	String[]
activityType	String
dueDate	Date
endDate	Date
executionState	String
executionType	String
hidden	Boolean
id	String
isPreconditionMatchAll	Boolean
name	String
narrative	String
optionType	String
preconditionExpressions	String[]
preconditionExpressionsDisplay	String[]
preconditionTrigger	String[]
preconditionType	String
repeatable	Boolean
required	Boolean
startDate	Date

Property	Type
taskId	String
taskOwnerGroupId	String
taskOwnerUserId	String

Data Type – ActivitySummary

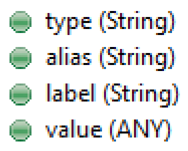
(As of 8.5.5)



- bpdFlowObjectId - String
- name - String
- countActive - Integer
- countOverdue - Integer
- countAtRisk - Integer
- countOnTrack - Integer
- propertyNames - *String
- propertyValues - *Object

Data Type – BusinessData

(As of 8.5.5)

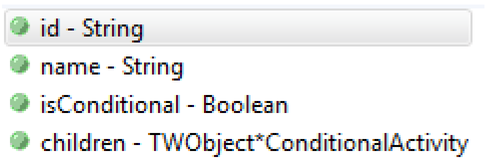


- type (String)
- alias (String)
- label (String)
- value (ANY)

See also:

Data Type – ConditionalActivity

A `ConditionalActivity` object represents an Activity in the BPD that is defined as conditional.



- id - String
- name - String
- isConditional - Boolean
- children - TWObject*ConditionalActivity

An array of these objects for the currently executing process can be found in the variable:

tw.system.currentProcess.conditionalActivities

One can use the `toXMLString()` method on variables of this type to dump an XML representation.

See also:

Data Type – InstanceListData

Return data from `retrieveInstanceList()`.

Property	Type
instanceIndexLastIndexStartTime	String
instanceIndexUpdateInterval	Integer
instances	InstanceListItem[]
total	Integer

The call to `retrieveInstanceList()` returns a list of process instances. This is encapsulated in this object. They core of this object is the array of `instances`.

See also:

Data Type – InstanceListItem

Property	Type
completionDate	String
creationDate	String
dueDate	String
id	String
isFollowed	Boolean
name	String
owningTeamId	String
owningTeamName	String
riskState	String

Property	Type
status	String

See also:

Data Type – InstanceListProperties

Property	Type
searchFilter	String
statusFilter	String[]

Data Type – InstanceTasksData

Property	Type
tasks	TaskListItem[]
total	Integer

Data Type – InstanceTasksFilter

Property	Type
statusFilter	String[]

Data Type – InstanceTasksProperties

|| |Property|Type| |checkActions|String[]| |filters|InstanceTaskFilter[]| |sortCriteria|String[]|

Data Type – Map

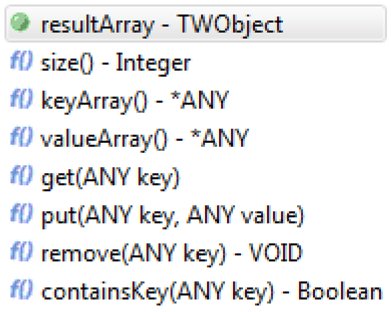
The Map data type holds an unordered collection of key/value pairs. You can think of this very much like the Java HashMap class. A value is added to the map using the `put()` method with the key as the first parameter and the value as the second. A value can subsequently be retrieved from the map using the `get()` method.

The array of all keys can be retrieved with the `keyArray()` method. Similarly, an array of all values an be obtained with `valueArray()` method. For both these methods, the order of returned entries is undefined as there is no ordering in maps.

The number of items contained within the map can be obtained with the `size()` method.

An entry can be removed from the map using the `remove()` method.

If we simply want to know if a map contains a value for a specific key, we can use the `containsKey()` method.


resultArray - TWObject
size() - Integer
keyArray() - *ANY
valueArray() - *ANY
get(ANY key)
put(ANY key, ANY value)
remove(ANY key) - VOID
containsKey(ANY key) - Boolean

An instance of this object can be created through:

tw.local.myMap = new tw.object.Map();

If passed to a Java class, the data type passed is `java.util.HashMap`.

Data Type – NameValuePair

A Business Object data structure that is pre-defined by IBPM that contains two fields:

Data Type – PathStep

An instance is available as `traversedPaths` in a `TWProcessInstance`.

Note: When accessing this type if variable, is is essential that the toolkit called "Dashboards" be added to your project as the data types involved in this data are contained within that toolkit.

The Step data structure contains:

`id` (`String`) – ID of the step (eg. `bpdid:xxx`)

`name` (`String`) – The name of the step.

`type` (`String`) – The type of the step. Values seen include:

phaseId (String)

nextSteps[] (String)

`parentBPD` (`String`) – The BPD ID of the parent (eg. `25.xxx`).

parentStepId (String)

duration (Integer)

See also:

Data Type – ProcessInstanceListItem

Property	Type
age	Integer
dueDate	String
estimatedCompletionDateEnabled	Boolean
id	String
name	String
riskState	String

Data Type – ProcessInstanceListProperties

Property	Type
beginIndex	Integer

Property	Type
maxRows	Integer
riskState	String
searchFilter	String
sortCriteria	String
stepId	String
stepRiskState	String

Data Type – ProcessSummary

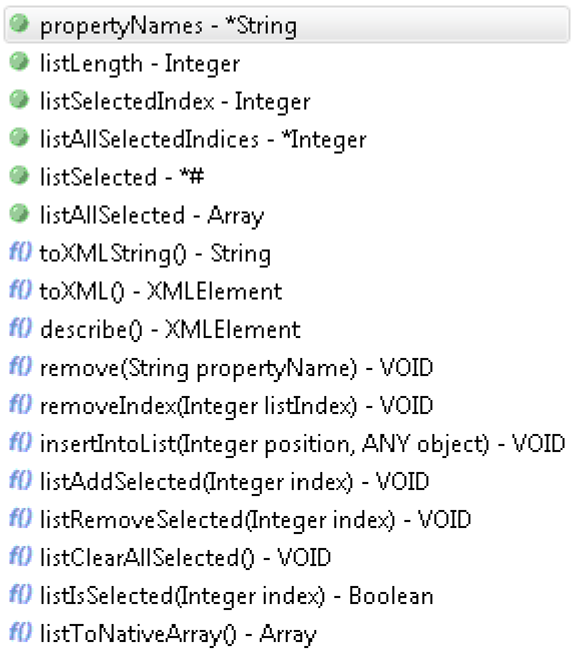
Property	Type
countActive	Integer
countAtRisk	Integer
countOnTrack	Integer
countOverdue	Integer
description	String
name	String
processAppId	String
processAppName	String
processId	String

This data type can be returned from `tw.system.retrieveProcessSummaries()`, `tw.system.retrieve.ProcessSummary()`, `TWProcessPerformanceMetric.retrieveProcessSummary()`.

See also:

Data Type – Record

The `Record` data type appears to be a dynamic set of properties associated with the an instance of the object. It looks like one can assign to and retrieve values from this object as though they were properties. Quite why it shows that it has the properties of a list object is unknown and may be an error in the PD tool.



propertyNames - *String
listLength - Integer
listSelectedIndex - Integer
listAllSelectedIndices - *Integer
listSelected - *#
listAllSelected - Array
f() toXMLString() - String
f() toXML() - XMLElement
f() describe() - XMLElement
f() remove(String propertyName) - VOID
f() removeIndex(Integer listIndex) - VOID
f() insertIntoList(Integer position, ANY object) - VOID
f() listAddSelected(Integer index) - VOID
f() listRemoveSelected(Integer index) - VOID
f() listClearAllSelected() - VOID
f() listIsSelected(Integer index) - Boolean
f() listToNativeArray() - Array

For example, if we define a variable called "`myRecord`" of type `Record` then in one activity I can assign a value to a property within it such as:

tw.local.myRecord = new tw.object.Record(); tw.local.myRecord.myProp = "Hello";

and then in a different activity we can code:

log.info(tw.local.myRecord.myProp);

The `Record` object appears to behave as a cross between a standard JavaScript object and a BPM Business Object. We can assign values to its properties and they are persisted across activities however they do not need to be predefined.

The `propertyNames` attribute returns a string which is a comma separated list of the explicitly added properties to the record. This allows us to enumerate the properties contained within it.

Since the Record data type can host properties that were previously undefined, this makes it especially useful for hosting data of types that can not be known during development. If we receive a JavaScript object, we may wish to turn this into a Record instance. This is easy if the properties of the received JavaScript object are simple types but we get into more complexity if the properties are arrays or other objects. Fortunately, there is a simple algorithm that can solve this puzzle for us. It is reproduced here:

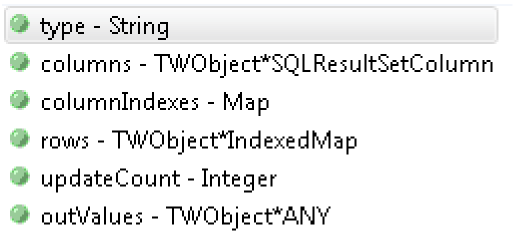
function toBusinessObject(current) { var container; if (current instanceof Array) { container = new tw.object.listOf.ANY(); } else { container = new tw.object.Record(); } // Loop through all the properties in the data for (var p in current) { if (current.hasOwnProperty(p)) { switch(typeof current[p]) { case 'string': case 'number': case 'boolean': container[p] = current[p]; break; case 'object': container[p] = toBusinessObject(current[p]); break; } // End of switch } // End of has own property } // End of for return container; };

This defines a function called "toBusinessObject" which accepts a JavaScript object or array as input. It returns a Record or list of ANY type business objects which can then be accessed or worked upon as desired.

See also:

Data Type – SQLResult

This data type is defined in the System Data toolkit. It appears to have been designed to hold results from executing DB queries through the SQL query services supplied in the System Data Toolkit.



type - String
columns - TWObject*SQLResultSetColumn
columnIndexes - Map
rows - TWObject*IndexedMap
updateCount - Integer
outValues - TWObject*ANY

Imagine a SQL Query executed using the System Data Toolkit function called `SQLExecuteStatement` that returns a `SQLResult[]` array. It is strange that this returns an array as `SQLResult[0]` seems to hold everything we need. Specifically `SQLResult[0]` holds an array object called `rows`. Each element of the `rows` array has a property named after the retrieved data column.

For example, if the data returned from a query in a SQLResult were:

A	B
"A1"	"B1"
"A2"	"B2"
"A3"	"B3"

Then `myVar[0].rows[1].B` would be "`B2`".

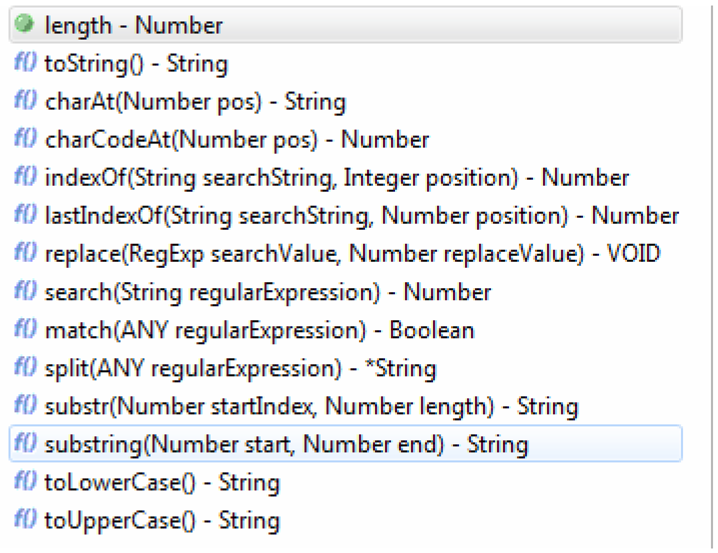
To see how many rows are returned, use `SQLResult[0].rows.listLength`.

Data Type – Step

Property	Type
duration	Integer
id	String
name	String
nextSteps	String[]
parentBPD	String
parentStepId	String
phaseId	String
type	String

Data Type – String

The String data type represents a piece of text within the process.



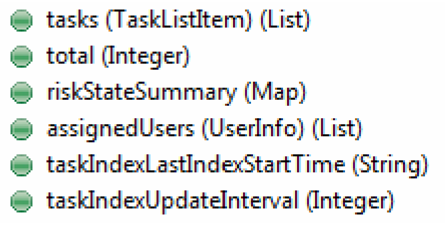
length - Number
toString() - String
charAt(Number pos) - String
charCodeAt(Number pos) - Number
indexOf(String searchString, Integer position) - Number
lastIndexOf(String searchString, Number position) - Number
replace(RegExp searchValue, Number replaceValue) - VOID
search(String regularExpression) - Number
match(ANY regularExpression) - Boolean
split(ANY regularExpression) - *String
substr(Number startIndex, Number length) - String
substring(Number start, Number end) - String
toLowerCase() - String
toUpperCase() - String

The search() method will return 0 if found and -1 if not found.

The replace() method seems to work with formats such as /<pattern>/g.

Data Type – TaskListData

This data type appears to be used as the return type from `tw.system.retrieveTaskList()`. It contains details of the tasks that are eligible to be performed by a user. It is primarily used to populate the Task List Coach View.

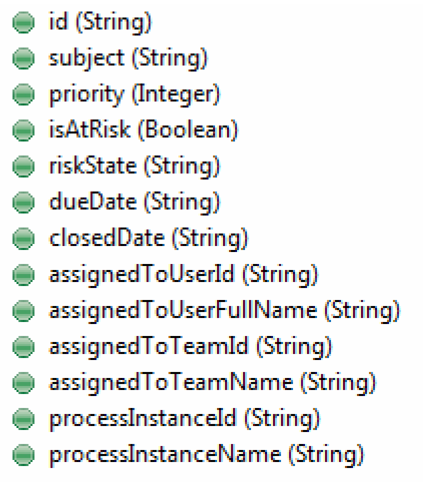


- tasks (TaskListItem) (List)
- total (Integer)
- riskStateSummary (Map)
- assignedUsers (UserInfo) (List)
- taskIndexLastIndexStartTime (String)
- taskIndexUpdateInterval (Integer)

See also:

Data Type – TaskListItem

This data type describes the details of a single task. A list of such items can be found in the `TaskListData` object.



- id (String)
- subject (String)
- priority (Integer)
- isAtRisk (Boolean)
- riskState (String)
- dueDate (String)
- closedDate (String)
- assignedToUserId (String)
- assignedToUserFullName (String)
- assignedToTeamId (String)
- assignedToTeamName (String)
- processInstanceId (String)
- processInstanceName (String)

`id` – The unique identifier of this task

`subject` – The subject property of the task

`priority` – The priority of the task (integer value)

`isAtRisk` – A boolean which flags this task as at risk (or not)

`riskState` – One of

- Overdue

- ....

`dueDate` – When the task is due. Note that this is a string!! Unfortunately as on 8.5.5, we can't parse this String to a date because the time zone formatter needs to be 'X' and that didn't arrive until Java 7. The current IBM BPM uses Java 6.

`closedDate` – When the task was closed

`assignedToUserId` – The userid that owns the task. Off the format "2048.xxxx".

`assignedToUserFullName` – The text full name of the user that owns the task.

`assignedToTeamId` – The id of the team to which this task is assigned.

`assignedToTeamName` – The human readable name of the team to which this task is assigned.

`processInstanceId` – The unique identifier of the process which created this task

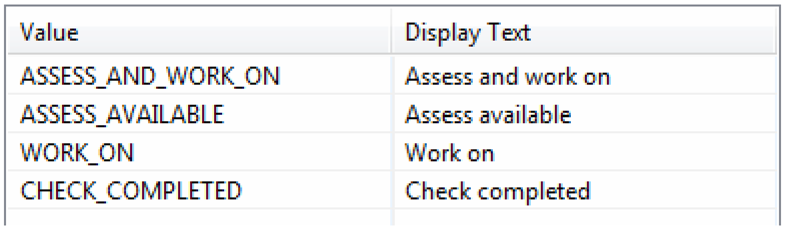
`processInstanceName` – The instance description of the process instance

See also:

- Data Type – TaskListData

Data Type – TaskListInteactionFilter

A string defined as one of the following:



Value	Display Text
ASSESS_AND_WORK_ON	Assess and work on
ASSESS_AVAILABLE	Assess available
WORK_ON	Work on
CHECK_COMPLETED	Check completed

See also:

Data Type – TaskListProperties

This data type is passed as a parameter to `tw.system.retrieveTaskList()`. It controls which tasks are returned.

Attribute	Type
collapsedRiskStates	String[]
dueSlice	DateRangeString
includeAssignedUsers	Boolean
includeRiskStateSummary	Boolean
interactionFilter	String
searchFilter	String
teamId	String
userId	String

`teamId` (optional) – Return tasks that are valid for the specific team Id.

`userId` (optional) – Return tasks that are valid for the specific userid.

`interactionFilter` (optional) – Return tasks which match the interaction filter. One of

- `ASSESS_AND_WORK_ON` – Tasks that are either claimed or not claimed but not suspended.
- `ASSESS_AVAILABLE` – Tasks that are not claimed and not suspended.
- `WORK_ON` – Tasks that are claimed and can be progressed. This means that suspended tasks are not included.
- `CHECK_COMPLETED` – Tasks which have been completed

`includeAssignedUsers` (optional) – Should the results include assigned users? Default is false.

`includeRiskStateSummary` (optional) – Should the result include a map of which tasks are at risk? Default is false.

`dueSlice` (optional) – Filter the tasks that are returned to be in the due time range.

`searchFilter` (optional) – Return tasks that contain the text.

`collapsedRiskStates` (optional) – Filter the return values such that only tasks in the given state are returned. The states available are:

- Overdue
- AtRisk
- DueToday

- DueTomorrow
- DueThisWeek
- DueLater

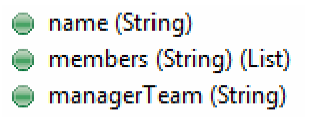
This data type is part of the Dashboards Toolkit.

See also:

- Data Type – TaskListInteactionFilter
- Method: tw.system.retrieveTaskList()
- Task List Control

Data Type – Team

This data type defines a dynamic Team.



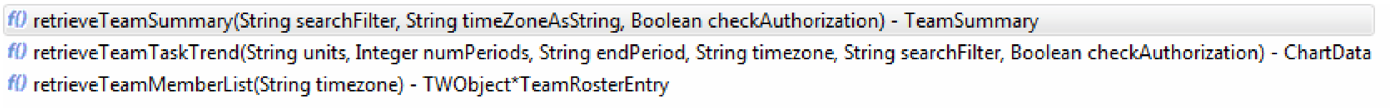
name (String)
members (String) (List)
managerTeam (String)

The name property is the name of the Team. Apparently this is ignored. The members property is a list of members of the team. The `managerTeam` property is the name of another Team that holds the managers of this Team.

See also:

Data Type – TeamDashboardSupport

(as of 8.5)



f() retrieveTeamSummary(String searchFilter, String timeZoneAsString, Boolean checkAuthorization) - TeamSummary
f() retrieveTeamTaskTrend(String units, Integer numPeriods, String endPeriod, String timezone, String searchFilter, Boolean checkAuthorization) - ChartData
f() retrieveTeamMemberList(String timezone) - TWObject*TeamRosterEntry

See also:

Data Type – TWAdhocStartingPoint

This JavaScript data structure represents an "ad-hoc" starting point within an already running IBPM process instance.

Properties:

Name/Type	Description
id:String	The id value of this ad-hoc starting point
name:String	The name value of this ad-hoc starting point
processInstance:TWProcessInstance	The process instance associated with this ad-hoc starting point

Methods:

Name	Description
startNew	Starts an ad-hoc activity

A field in the `TWProcessInstance` data type called `adhocStartingPoints` contains a list of `TWAdhocStartingPoints` associated with that instance of the process. The `TWProcessInstance` also has convenience methods called `findAdhocStartingPointByID()` and `findAdhocStartingPointByName()` to walk this list looking for specific entries.

Data Type – TWDate

The `TWDate` object represents a date/time. An instance of this object can be created with:

new tw.object.Date()

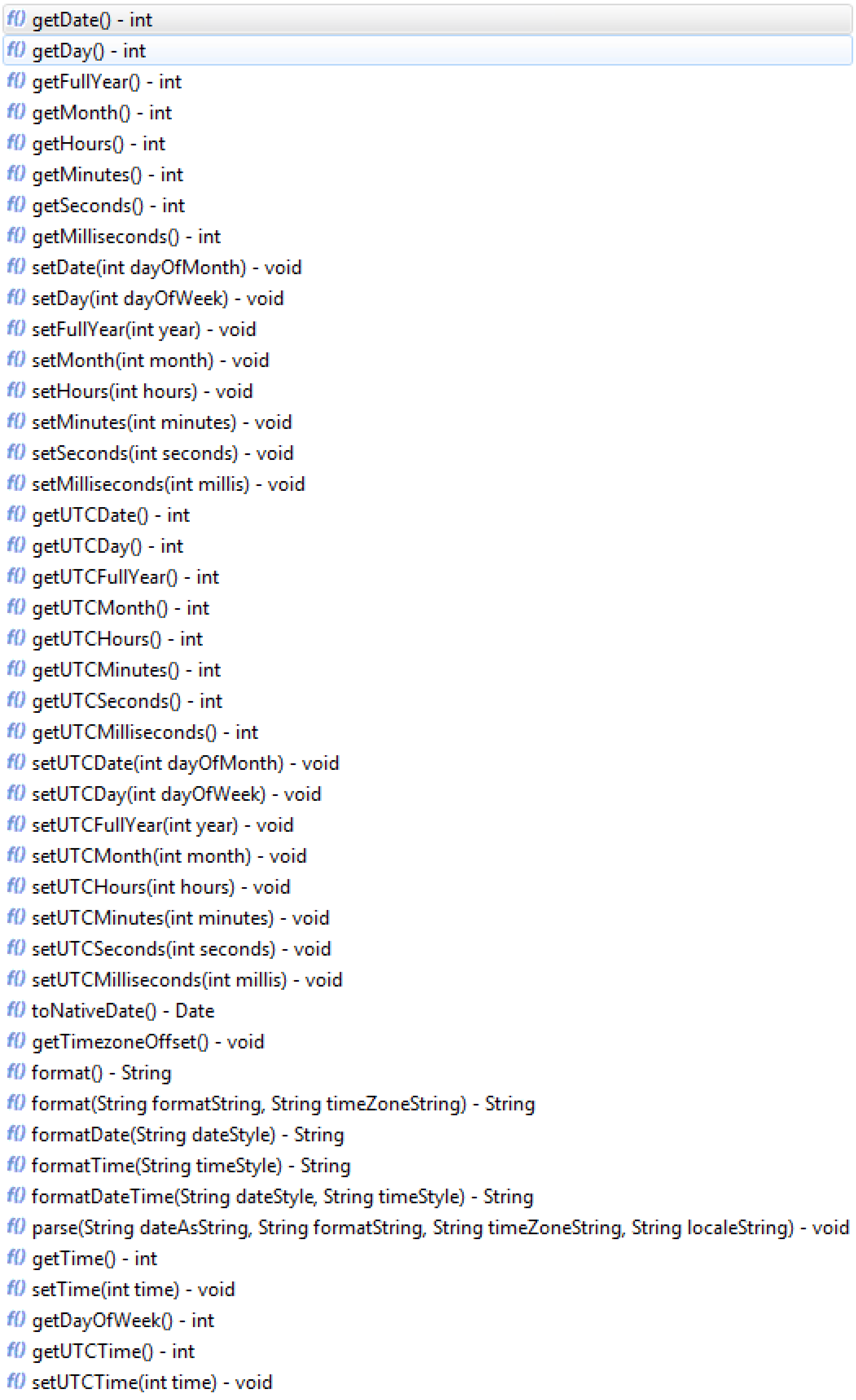
The format fields in some of the methods allow for custom date/time parsing or construction. An example of such a function is `format()`. A date can also be constructed using the parse() function. The format is a String that is encoded as follows:

|**Code**|**Description**|**Example**| |-|-|-| |G|Era designator|AD ("GG")| |y|Year|2011 ("yyyy"); 11 ("yy")| |M|Month in year|July; Jul; 07| |w|Week in year|26| |W|Week in month|3| |D|Day in year|203| |d|Day in month|11| |F|Day of week in month|3| |E|Day in week|Tuesday ("EEEE") ; Tue ("EEE")| |a|AM/PM indicator|PM ("GG")| |H|Hour in the day (0-23)|0| |k|Hour in the day (1-24)|24| |K|Hour in am/pm (0-11)|0| |h|Hour in am/pm (1-12)|12| |m|Minute in hour|45| |s|Second in minute|33| |S|Millisecond|934| |z|Time zone|Central Standard Time; CST; GMT-06:00| |Z|Time zone|-800|

Examples:

Pattern	Example
MM/dd/yyyy	11/08/2012
yyyy-MM-dd HH:mm:ss	
yyyy-MM-dd HH:mm:ss.SSS zzz	

(As of 8.5)



```
getDate() - int
getDay() - int
getFullYear() - int
getMonth() - int
getHours() - int
getMinutes() - int
getSeconds() - int
getMilliseconds() - int
setDate(int dayOfMonth) - void
setDay(int dayOfWeek) - void
setFullYear(int year) - void
setMonth(int month) - void
setHours(int hours) - void
setMinutes(int minutes) - void
setSeconds(int seconds) - void
setMilliseconds(int millis) - void
getUTCDate() - int
getUTCDay() - int
getUTCFullYear() - int
getUTCMonth() - int
getUTCHours() - int
getUTCMinutes() - int
getUTCSeconds() - int
getUTCMilliseconds() - int
setUTCDate(int dayOfMonth) - void
setUTCDay(int dayOfWeek) - void
setUTCFullYear(int year) - void
setUTCMonth(int month) - void
setUTCHours(int hours) - void
setUTCMinutes(int minutes) - void
setUTCSeconds(int seconds) - void
setUTCMilliseconds(int millis) - void
toNativeDate() - Date
getTimezoneOffset() - void
format() - String
format(String formatString, String timeZoneString) - String
formatDate(String dateStyle) - String
formatTime(String timeStyle) - String
formatDateTime(String dateStyle, String timeStyle) - String
parse(String dateAsString, String formatString, String timeZoneString, String localeString) - void
getTime() - int
setTime(int time) - void
getDayOfWeek() - int
getUTCTime() - int
setUTCTime(int time) - void
```

Notice the `getTime()` and `setTime()` methods. These return and set times as seconds from an epoch.

The formatString in the parse() method uses the Java SimpleDataFormat rules. See:

http://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html

Unfortunately this object has no explicit date/time arithmetic available to it so adding or subtracting time can be a challenge. However, if we look to Java, we find a powerful class called `java.util.Calendar`. The `Calendar` class has all kinds of mechanisms to perform arithmetic and other functions. The task then is to create an instance of a Java `Calendar` from the `TWDate` object, perform the arithmetic on the `Calendar` and then create a new `TWDate` from the update `Calendar` object. The following will subtract one day from the original date:

```
tw.local.dueDate1.setFullYear(2010); tw.local.dueDate1.setMonth(2);
tw.local.dueDate1.setDate(1); tw.local.dueDate1.setHours(15);
tw.local.dueDate1.setMinutes(18); tw.local.dueDate1.setSeconds(00);
tw.local.dueDate1.setMilliseconds(0); log.info("Original Date is: " +
tw.local.dueDate1.format("yyyy-MM-dd HH:mm:ss.SSS zzz")); var cal1 = new
java.util.GregorianCalendar( tw.local.dueDate1.getFullYear(), tw.local.dueDate1.getMonth(),
tw.local.dueDate1.getDate(), tw.local.dueDate1.getHours(), tw.local.dueDate1.getMinutes()
tw.local.dueDate1.getSeconds()); cal1.add(java.util.Calendar.DATE, -1); tw.local.secondDate =
new tw.object.Date(); tw.local.secondDate.setMilliseconds(0);
tw.local.secondDate.setFullYear(cal1.get(java.util.Calendar.YEAR));
tw.local.secondDate.setMonth(cal1.get(java.util.Calendar.MONTH));
tw.local.secondDate.setDate(cal1.get(java.util.Calendar.DAY_OF_MONTH));
tw.local.secondDate.setHours(cal1.get(java.util.Calendar.HOUR));
tw.local.secondDate.setMinutes(cal1.get(java.util.Calendar.MINUTE));
tw.local.secondDate.setSeconds(cal1.get(java.util.Calendar.SECOND)); log.info("New Date is: " +
tw.local.secondDate.format("yyyy-MM-dd HH:mm:ss.SSS zzz"));
```

There is a general function in the Kolban JavaScript toolkit that can be used to add/subtract dates (see: Error: Reference source not found)

Data Type – TWDocument

The TWDocument object represents a document uploaded through a Coach and stored/managed by IBPM.

Properties:

Property	Description
`id:String`	The internal *id* of the document. Will be unique for every document.
`name:String`	The *name* of the document.
`version:Integer`	The version number of the document.
`type:String`	URL or File
`uri:String`	The URI of the document if its type is URL.
`contentType:String`	The HTTP MIME type of the document.
`allVersions:TWDocument[]`	An array of all the versions of this document.
`processInstance:TWProcessInstance`	The Process Instance that this document belongs to.
`hideInPortal:Boolean`	True of the document should be *hidden* in Process Portal.
`modifiedBy:TWUser`	The user who last modified/added the document.
`modificationDate:Date`	The date when the document was added.

Methods:

|`deleteAllVersions`|| |`getTypeName`|| |`writeDataToFile`|Write the content of the document to a file on the local file system where Process Server is executing.|

The `TWProcessInstance` object has an `addDocument()` method to create a new document.

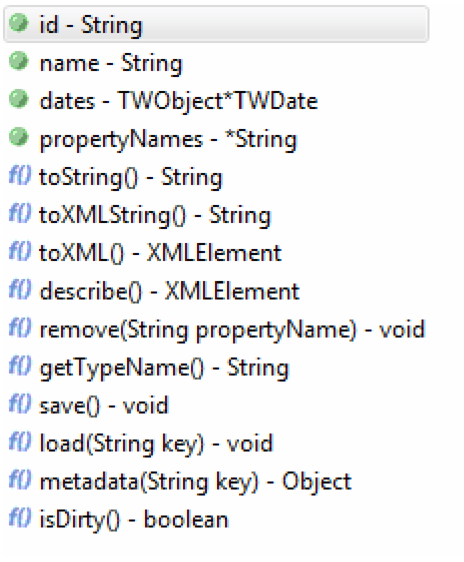
There does not appear to be a way to get "simple" access to the content of the document however it appears that we can write it to a file and then load the file content. As to why there is no way to access the content, there are a couple of thoughts (unverified) as to why IBM's designers chose not to provide this. One thought is that working with binary data in the context of a Business Process is simply not a desirable trait. By not providing any getters for the data, the designer of a solution is forced to immediately look into Java alternatives from the start. A second possible reason is that the JavaScript language has no binary data handlers.

A possible solution to this is to encode the data into a String representation. We appear to have two choices for this. The first is hexBinary which takes a single byte of data and represents it by two characters. The two characters are the hexadecimal notation for the byte value. A class is supplied as part of the Axis2 package to perform hexBinary encoding. The class is called "HexBinary". The second technique is to encode as base64. Base64 encoding is a little more complex but basically encodes 3 bytes of data into 4 characters. Base64 encoding can be used from the Apache commons.codec package.

See also:

Data Type – TWHolidaySchedule

This data type describes a holiday schedule. A holiday schedule is a list of dates that are considered holidays. This comes into play when we are calculating a due date for a process or activity. If the date on which a task would otherwise complete is a holiday, then we need to accommodate for that by moving the due date to the right to the first non-holiday date.



```
id - String
name - String
dates - TWObject*TWDate
propertyNames - *String
toString() - String
toXMLString() - String
toXML() - XMLElement
describe() - XMLElement
remove(String propertyName) - void
getTypeName() - String
save() - void
load(String key) - void
metadata(String key) - Object
isDirty() - boolean
```

The variable called tw.system.holidaySchedules contains a list of all the different holiday schedules available.

We can add/define a new holiday schedule with

tw.system.addHolidaySchedule(TWHolidaySchedule holidaySchedule).

We can find a holiday schedule by name with tw.system.findHolidayScheduleByName(String name).

We can remove a holiday schedule with

tw.system.removeHolidaySchedule(String id).

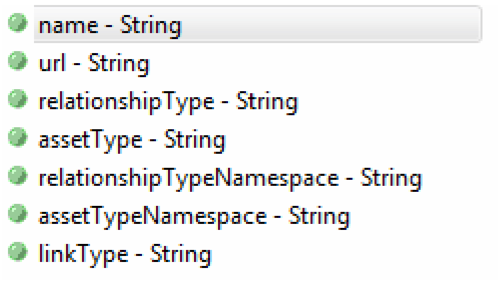
We can update a holiday schedule with

tw.system.updateHolidaySchedule(TWHolidaySchedule holidaySchedule).

See also:

Data Type – TWLink

This item defines the referenced links in the documentation.



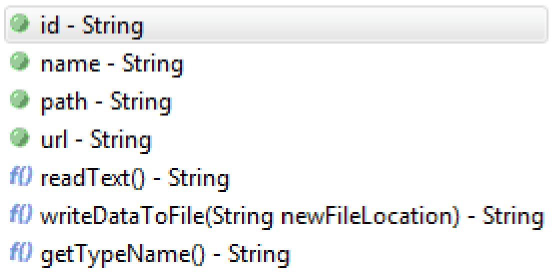
- name - String
- url - String
- relationshipType - String
- assetType - String
- relationshipTypeNamespace - String
- assetTypeNamespace - String
- linkType - String

Instances of this type can be found from instances of TWProcessInstance and TWStep.

See also:

Data Type – TWManagedFile

The managed file type represents a file that can be found as part of a toolkit or Process Application.



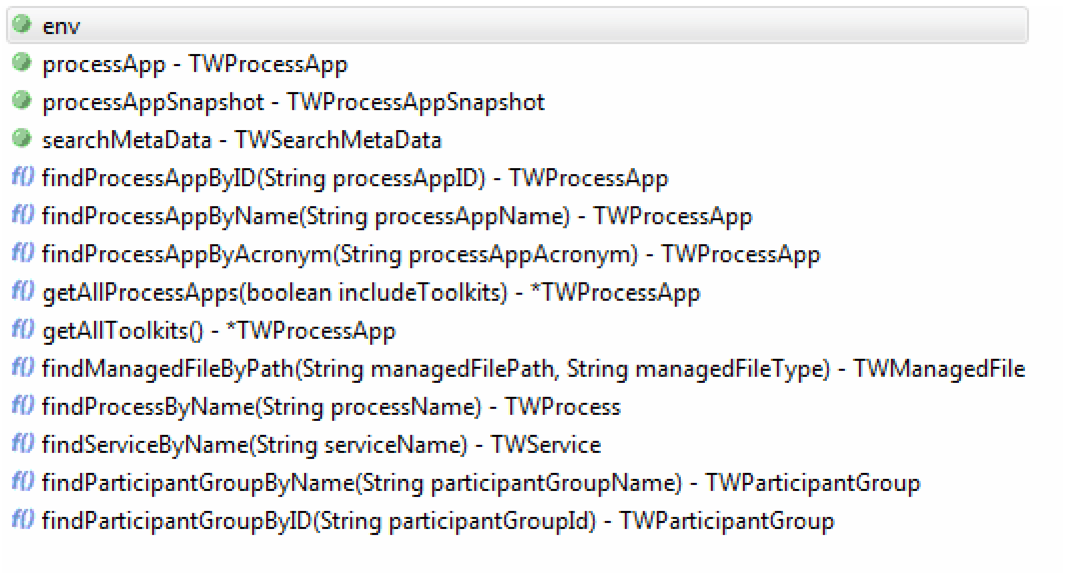
- id - String
- name - String
- path - String
- url - String
- f() readText() - String
- f() writeDataToFile(String newFileLocation) - String
- f() getTypeName() - String

See also:

Data Type – TWModelNamespace

An instance of this object can be retrieved at the BPD level using:

tw.system.model

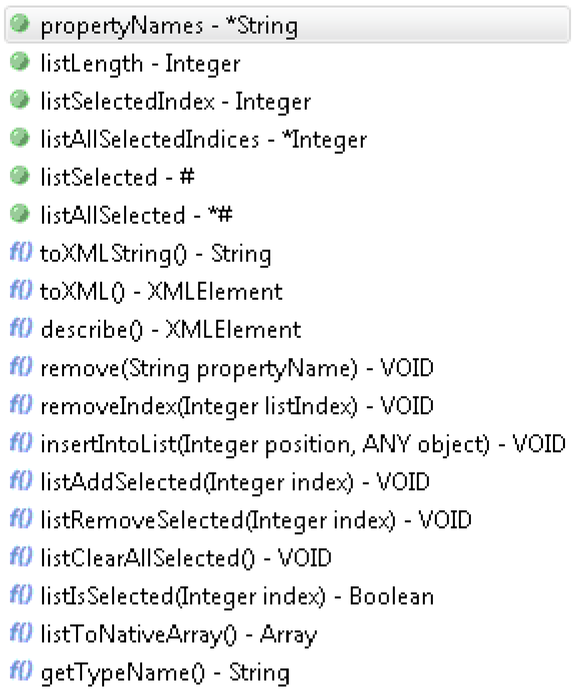


```
env
processApp - TWProcessApp
processAppSnapshot - TWProcessAppSnapshot
searchMetaData - TWSearchMetaData
f() findProcessAppByID(String processAppID) - TWProcessApp
f() findProcessAppByName(String processAppName) - TWProcessApp
f() findProcessAppByAcronym(String processAppAcronym) - TWProcessApp
f() getAllProcessApps(boolean includeToolkits) - *TWProcessApp
f() getAllToolkits() - *TWProcessApp
f() findManagedFileByPath(String managedFilePath, String managedFileType) - TWManagedFile
f() findProcessByName(String processName) - TWProcess
f() findServiceByName(String serviceName) - TWService
f() findParticipantGroupByName(String participantGroupName) - TWParticipantGroup
f() findParticipantGroupByID(String participantGroupId) - TWParticipantGroup
```

See also:

Data Type – TWObject

`TWObject` appears to be the implementation of lists, arrays and Business Object types within IBPM. It supports direct assignment through square bracket notation including the idea of sparse arrays (i.e. assigning to an array element past the end of the array).



propertyNames - *String
listLength - Integer
listSelectedIndex - Integer
listAllSelectedIndices - *Integer
listSelected - #
listAllSelected - *#
toXMLString() - String
toXML() - XMLElement
describe() - XMLElement
remove(String propertyName) - VOID
removeIndex(Integer listIndex) - VOID
insertIntoList(Integer position, ANY object) - VOID
listAddSelected(Integer index) - VOID
listRemoveSelected(Integer index) - VOID
listClearAllSelected() - VOID
listIsSelected(Integer index) - Boolean
listToNativeArray() - Array
getTypeName() - String

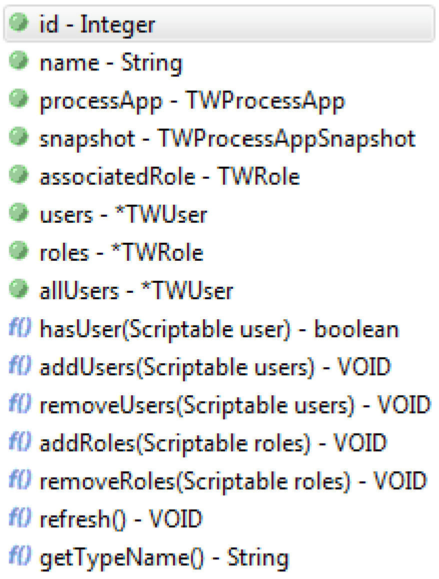
If we want to add a new entry at the very end of the list, use the following pattern:

tw.local.myListVariable[tw.local.myListVariable.listLength] = value;

One of the more interesting default properties of this object is `propertyNames` which is an array of names of the properties in the Business Object which have values associated them. This can be used for introspection of an arbitrary Business Object to determine its internal structure. Take careful note that the propertyNames is **only** fields that have values. For example, if a BO is defined as having fields "a", "b" and "c" but only "b" is populated, then propertyNames will be the list ["b"] and will not show the potential fields of "a" or "c". If a variable is an array, all we get is the first element.

Data Type – TWParticipantGroup

It is believed that this data type has now been deprecated in favor of TWTeam.



```
id - Integer
name - String
processApp - TWProcessApp
snapshot - TWProcessAppSnapshot
associatedRole - TWRole
users - *TWUser
roles - *TWRole
allUsers - *TWUser
f() hasUser(Scriptable user) - boolean
f() addUsers(Scriptable users) - VOID
f() removeUsers(Scriptable users) - VOID
f() addRoles(Scriptable roles) - VOID
f() removeRoles(Scriptable roles) - VOID
f() refresh() - VOID
f() getTypeName() - String
```

The property called `users` lists the explicit users added to the group

The property called `roles` lists the explicit groups added to the group

The property called `allUsers` lists all the users in the group that were added explicitly and also those users resolved from expanding the groups in this group.

Here is an example script to retrieve all users for a participant group:

var myPG1 = tw.system.org.findParticipantGroupByName("myPG1"); var allUsers = myPG1.allUsers; for (var i=0; i<allUsers.length; i++) { log.info(allUsers[i].name); }

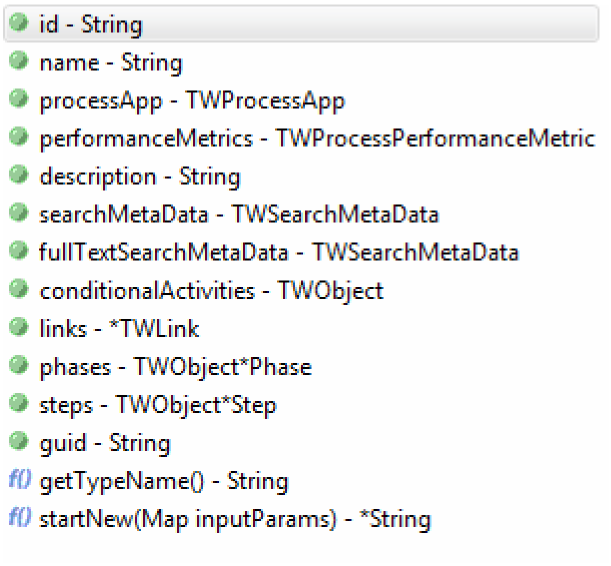
The `addUsers` function doesn't appear to do anything on Process Center. It is believed that `addUsers` will work when deployed to a Process Server.

See also:

Data Type – TWProcess

The TWProcess object represents the model of a process. An instance of this object can be retrieved through tw.system.model.findProcessByName(*processName*).

(The following as of 8.5.5)



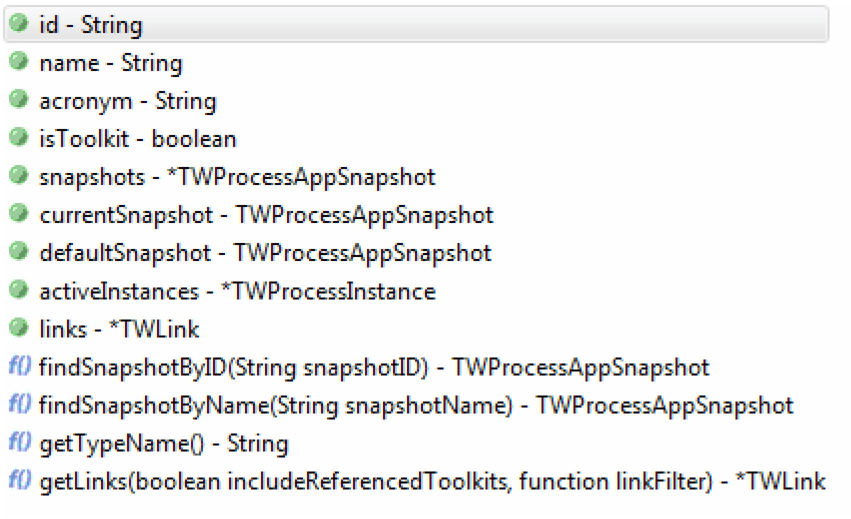
● id - String
● name - String
● processApp - TWProcessApp
● performanceMetrics - TWProcessPerformanceMetric
● description - String
● searchMetaData - TWSearchMetaData
● fullTextSearchMetaData - TWSearchMetaData
● conditionalActivities - TWObject
● links - *TWLink
● phases - TWObject*Phase
● steps - TWObject*Step
● guid - String
f() getTypeName() - String
f() startNew(Map inputParams) - *String

See also:

Data Type – TWProcessApp

The TWProcessApp object represents an instance of a process app.

(The following as of 8.5.5)

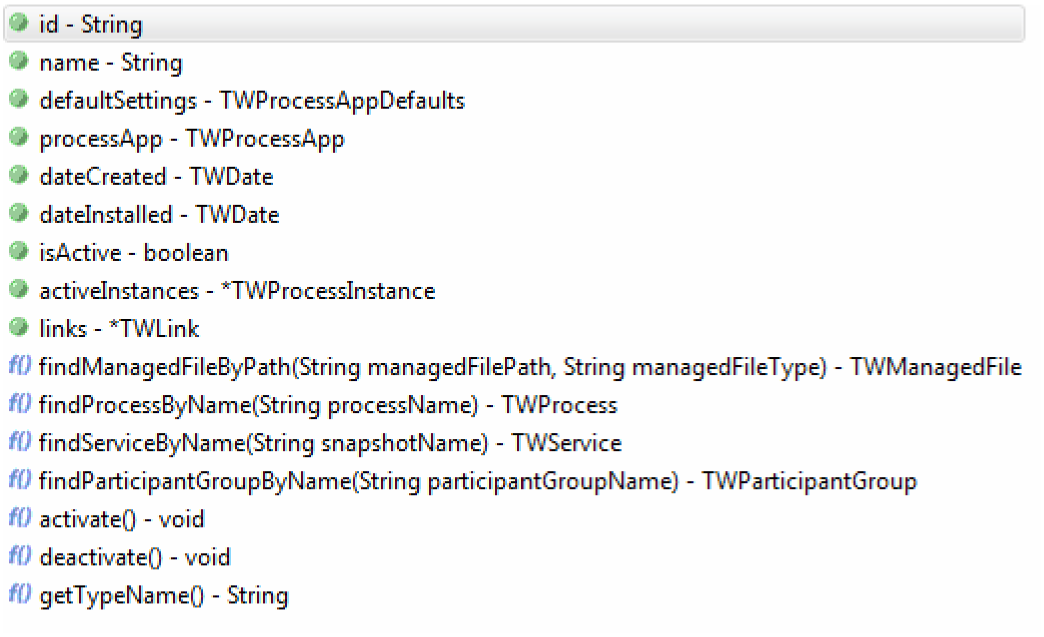


● id - String
● name - String
● acronym - String
● isToolkit - boolean
● snapshots - *TWProcessAppSnapshot
● currentSnapshot - TWProcessAppSnapshot
● defaultSnapshot - TWProcessAppSnapshot
● activeInstances - *TWProcessInstance
● links - *TWLink
f() findSnapshotByID(String snapshotID) - TWProcessAppSnapshot
f() findSnapshotByName(String snapshotName) - TWProcessAppSnapshot
f() getTypeName() - String
f() getLinks(boolean includeReferencedToolkits, function linkFilter) - *TWLink

See also:

Data Type – TWProcessAppSnapshot

The following (as of 8.5):



- id - String
- name - String
- defaultSettings - TWProcessAppDefaults
- processApp - TWProcessApp
- dateCreated - TWDate
- dateInstalled - TWDate
- isActive - boolean
- activeInstances - *TWProcessInstance
- links - *TWLink
- f() findManagedFileByPath(String managedFilePath, String managedFileType) - TWManagedFile
- f() findProcessByName(String processName) - TWProcess
- f() findServiceByName(String snapshotName) - TWService
- f() findParticipantGroupByName(String participantGroupName) - TWParticipantGroup
- f() activate() - void
- f() deactivate() - void
- f() getTypeName() - String

See also:

Data Type – TWProcessInstance

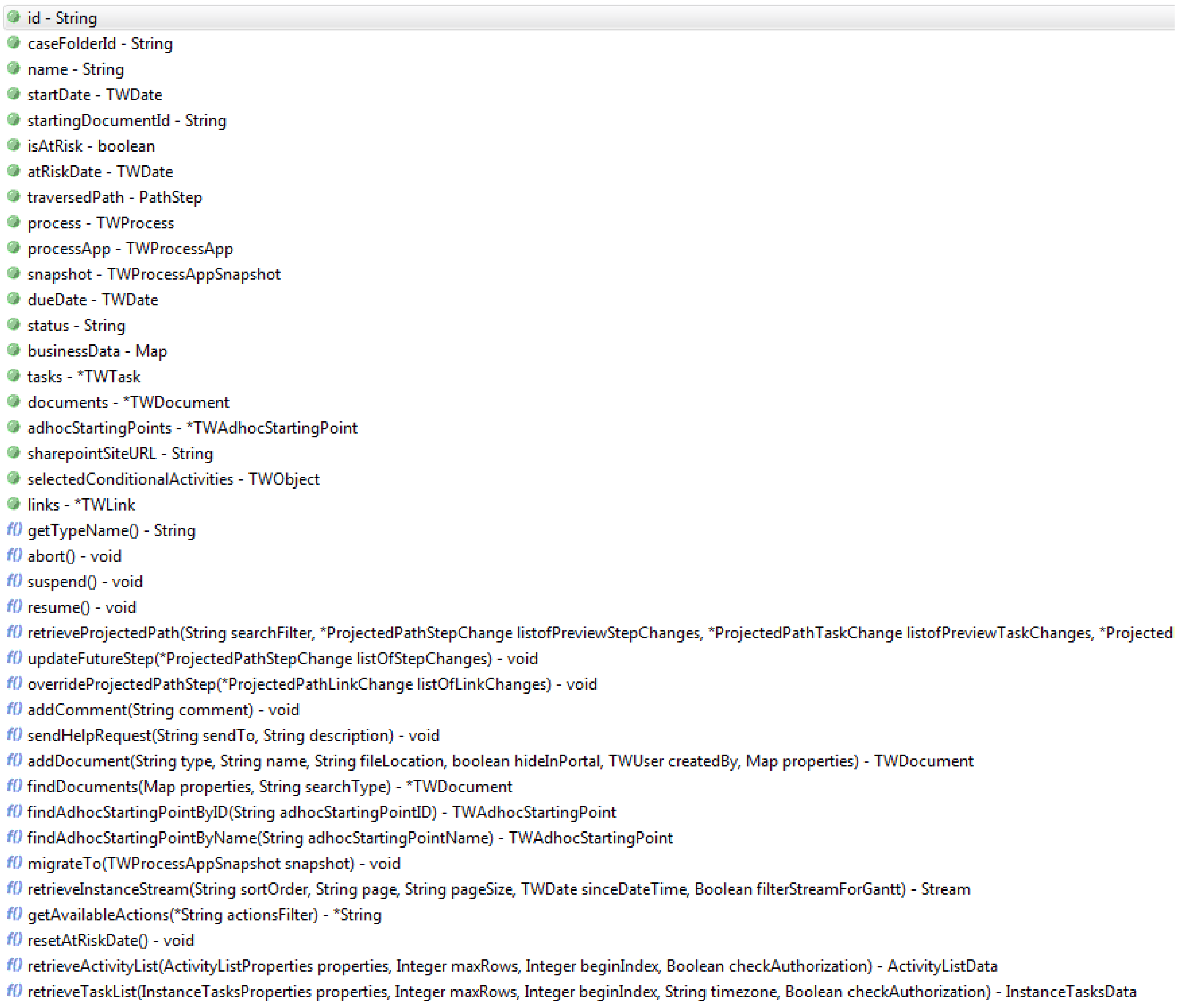
This data type reflects an instance of a process. A process is an instantiated BPD or Case.

(the following as of 8.5.5)

Property	Type
adhocStartingPoints	TWAdhocStartingPoint[]
atRiskDate	TWDate
businessData	Map
caseFolderId	String
caseFolderServerName	String
documents	TWDocument[]
documentSearchTypes	TWDocumentSearchTypes
dueDate	TWDate

Property	Type
id	String
isAtRisk	boolean
links	TWLink[]
name	String
process	TWProcess
processApp	TWProcessApp
selectedConditionalActivities	String[]
sharepointSiteURL	String
snapshot	TWProcessAppSnapshot
startDate	TWDate
startingDocumentId	String
startingDocumentServerName	String
status	String
statuses	TWProcessInstanceStatuses
tasks	TWTask[]
traversedPath	PathStep

|| || |void abort()| |void addComment(String comment)| |BPMRelationship addDependencyOnProcess(String processInstanceId, String description, Boolean checkAuthorization)| |BPMRelationship addDependentProcess(String processInstanceId, String description, Boolean checkAuthorization)| |TWDocument addDocument(String type, String name, String filelocation, boolean hideInPortal, TWUser createdBy, Map properties, Boolean checkAuthorization)| |addRelatedProcess| |findAdhocStartingPointByID| |findAdhocStartingPointByName| |findDocuments| |getAvailableActions()| |getDependedOnProcess| |getDependentProcess| |getRelatedProcesses| |getRelationships| |getTypeName| |migrateTo| |migrateWithContextTo| |overrideProjectedPathStep| |removeDependedOnProcessRelationship| |removeDependentProcessRelationship| |removeRelatedProcessRelationship| |resetAtRiskDate| |resume| |retrieveActivityList| |retrieveInstanceStream| |retrieveProjectedPath| |retrieveTaskList| |sendHelpRequest| |suspend| |updateFutureStep|



```
id - String
caseFolderId - String
name - String
startDate - TWDate
startingDocumentId - String
isAtRisk - boolean
atRiskDate - TWDate
traversedPath - PathStep
process - TWProcess
processApp - TWProcessApp
snapshot - TWProcessAppSnapshot
dueDate - TWDate
status - String
businessData - Map
tasks - *TWTask
documents - *TWDocument
adhocStartingPoints - *TWAdhocStartingPoint
sharepointSiteURL - String
selectedConditionalActivities - TWObject
links - *TWLink
getTypeName() - String
abort() - void
suspend() - void
resume() - void
retrieveProjectedPath(String searchFilter, *ProjectedPathStepChange listofPreviewStepChanges, *ProjectedPathTaskChange listofPreviewTaskChanges, *Projected
updateFutureStep(*ProjectedPathStepChange listOfStepChanges) - void
overrideProjectedPathStep(*ProjectedPathLinkChange listOfLinkChanges) - void
addComment(String comment) - void
sendHelpRequest(String sendTo, String description) - void
addDocument(String type, String name, String fileLocation, boolean hideInPortal, TWUser createdBy, Map properties) - TWDocument
findDocuments(Map properties, String searchType) - *TWDocument
findAdhocStartingPointByID(String adhocStartingPointID) - TWAdhocStartingPoint
findAdhocStartingPointByName(String adhocStartingPointName) - TWAdhocStartingPoint
migrateTo(TWProcessAppSnapshot snapshot) - void
retrieveInstanceStream(String sortOrder, String page, String pageSize, TWDate sinceDateTime, Boolean filterStreamForGantt) - Stream
getAvailableActions(*String actionsFilter) - *String
resetAtRiskDate() - void
retrieveActivityList(ActivityListProperties properties, Integer maxRows, Integer beginIndex, Boolean checkAuthorization) - ActivityListData
retrieveTaskList(InstanceTasksProperties properties, Integer maxRows, Integer beginIndex, String timezone, Boolean checkAuthorization) - InstanceTasksData
```

The "`id`" property contains the process instance ID of the process. This is encoded in the format:

2072.<PIID>

where the `2072` prefix is the marker that says that this is a process instance.

To find a process instance, we can use the JavaScript method:

tw.system.findProcessInstanceByID()

This has the syntax:

tw.system.findProcessInstanceByID(String processInstanceID);

This returns an instance of a `TWProcessInstance`.

The current process (the one executing) can be found at the variable:

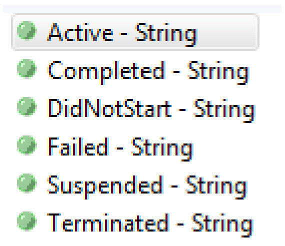
tw.system.currentProcessInstance

This variable is an instance of `TWProcessInstance`.

Note that there is a property called "`businessData`" which is a Map object containing all the variables in the process instance. It is believed that these are read-only and any attempt to change them will **not** result in changes to the variables in the process. Of deeper interest is the notion of a variable which is a Business Object. If a variable called "`bo1`" is of type BO1 which has properties "`a`" and "`b`", one would have imagined that we could retrieve an object called "`bo1`" from the map. Strangely, this isn't the case. Instead, the key to the map is the full path to each of the fields … i.e. "`bo1.a`" and "`bo1.b`".

In addition there are some constants defined:

TWProcessInstance.Statuses



● Active - String
● Completed - String
● DidNotStart - String
● Failed - String
● Suspended - String
● Terminated - String

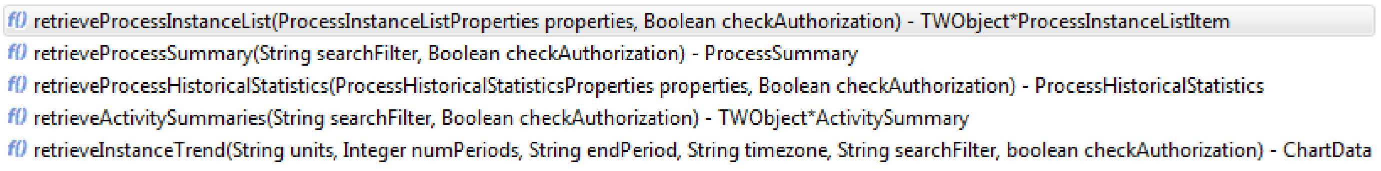
The `addDocument` method can add a document by file or by URI however this technology should no longer be used as it pre-dates the CMIS support and is present only for legacy applications that should ideally be recoded to use the new technologies of either the BPM Document Store or the BPM Content Store.

One of the properties of this object is called "`traversedPath`" which is a `PathStep` object.

See also:

Data Type – TWProcessPerformanceMetric

(The following as of 8.5)



```
f() retrieveProcessInstanceList(ProcessInstanceListProperties properties, Boolean checkAuthorization) - TWObject*ProcessInstanceListItem
f() retrieveProcessSummary(String searchFilter, Boolean checkAuthorization) - ProcessSummary
f() retrieveProcessHistoricalStatistics(ProcessHistoricalStatisticsProperties properties, Boolean checkAuthorization) - ProcessHistoricalStatistics
f() retrieveActivitySummaries(String searchFilter, Boolean checkAuthorization) - TWObject*ActivitySummary
f() retrieveInstanceTrend(String units, Integer numPeriods, String endPeriod, String timezone, String searchFilter, boolean checkAuthorization) - ChartData
```

This is a rich collection of methods. Some of these methods take rich data structures as parameters. As of 8.5, I haven't been able to find these adequately documented so here are notes on what I have found so far:

ProcessHistoricalStatisticsProperties

See also:

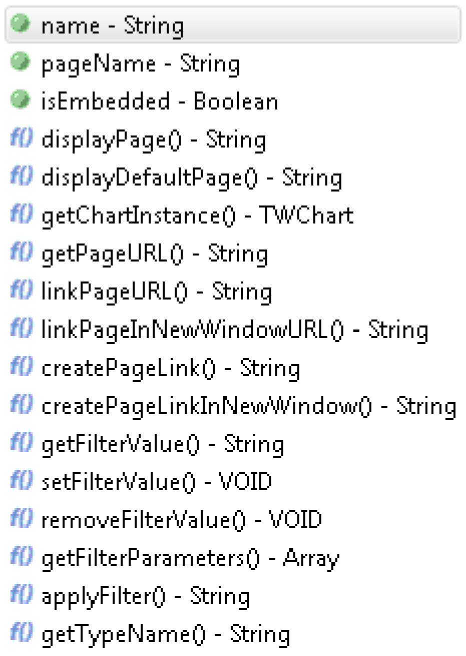
Data Type – TWProcessStepInfo

TBD

id	Integer
guid	String
name	String
counter	Integer
processInstance	TWProcessInstance
task	TWTask
timer	TWTimerInstance
error	XMLElement
isConditionalActivitySelected	Boolean
links	TWLink[]

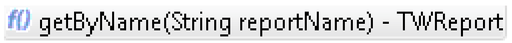
Data Type – TWReport

A `TWReport` represents or models a IBPM Report.



name - String
pageName - String
isEmbedded - Boolean
displayPage() - String
displayDefaultPage() - String
getChartInstance() - TWChart
getPageURL() - String
linkPageURL() - String
linkPageInNewWindowURL() - String
createPageLink() - String
createPageLinkInNewWindow() - String
getFilterValue() - String
setFilterValue() - VOID
removeFilterValue() - VOID
getFilterParameters() - Array
applyFilter() - String
getTypeName() - String

Static methods:



getByName(String reportName) - TWReport

A number of methods in this Object return HTML for inclusion in a page or Coach. These include:

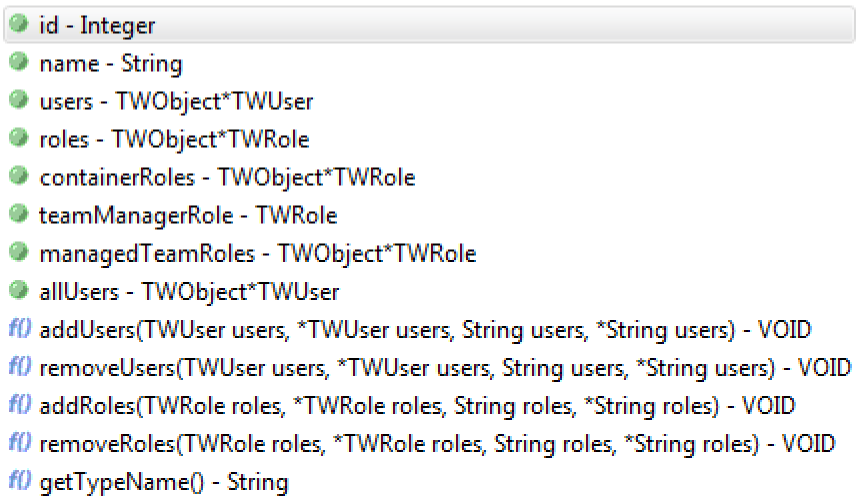
|| |createPageLink|| |createPageInNewWindowLink|| |displayDefaultPage|| |displayPage||

See also:

Data Type – TWRole

Within IBPM we have System Groups which are external definitions of collections of people. The `TWRole` object is the access to these System Groups. Loosely speaking, the phrases “Role” and “System Group” names can be used interchangeably. In discussions with users of versions of the ancestral TeamWorks product, there was a time when the concept of “Participant Groups” did not exist in the product and the only “user grouping” mechanism was the “Role”. The “Role” was used in BPMN swim lanes to describe the “role” that a user was to perform (eg. Insurance clerk vs Forklift driver)

TWRole as of 8.5:


```
id - Integer
name - String
users - TWObject*TWUser
roles - TWObject*TWRole
containerRoles - TWObject*TWRole
teamManagerRole - TWRole
managedTeamRoles - TWObject*TWRole
allUsers - TWObject*TWUser
addUsers(TWUser users, *TWUser users, String users, *String users) - VOID
removeUsers(TWUser users, *TWUser users, String users, *String users) - VOID
addRoles(TWRole roles, *TWRole roles, String roles, *String roles) - VOID
removeRoles(TWRole roles, *TWRole roles, String roles, *String roles) - VOID
getTypeName() - String
```

With the presumption that the `TWRole` is mapped to a System Group and a System Group is managed outside of the IBPM product (commonly as part of LDAP security), these doesn't appear to be any semantic mapping to the idea of performing the functions known as:

This data type can not be sent to "`log.info()`".

See also:

Data Type – TWSearch

This class is the core searching tool within IBM BPM from the JavaScript programming perspective.

Property	**Type**
columns	TWSearchColumn[]
conditions	TWSearchCondition[]
options	TWSearchOptions
orderBy	TWSearchOrdering[]
OrganizeByTypes	TWSearchOrganizeByTypes
organizedBy	String

The `columns` property defines which *columns* will be returned in the search.

The `organizedBy` property can have one of the values of the `TWSearch.OrganizeByTypes`. This will be:

Methods
TWSearchResults execute(
TWUser

Integer maxRows, Integer beginIndex)| |TWProcessInstance[] executeForProcessInstances(TWUser | TWRole | TWParticipantGroup | String userOrRole, Integer maxRows, Integer beginIndex)| |TWTask executeForTasks(TWUser | TWRole | TWParticipantGroup | String userOrRole, Integer maxRows, Integer beginIndex)|

See also:

Data Type – TWSearchColumn

This data structure is used to define which columns to return on a JavaScript `TWSearch` request.

Property	Type
name	String
ProcessColumns	TWSearchColumnProcessColumns
ProcessInstanceColumns	TWSearchColumnProcessInstanceColumns
TaskColumns	TWSearchColumnTaskColumns
type	String
Types	TWSearchColumnTypes

The `name` parameter must be one of the allowable column names taken from:

TWSearchColumn.ProcessInstanceColumns.*

TWSearchColumn.ProcessColumns.*

TWSearchColumn.TaskColumns.*

The type parameter defines which type the name is associated with. It must be one of the allowable values in the `TWSearchColumnTypes` values:

See also:

Data Type – TWSearchColumnMetaData

Instructions on how to interpret a column of data returned from a search.

Property	Type	Notes
displayName	String	
isUsableInSearchCondition	Boolean	
label	String	The label of the column.
name	String	The identity of the column.
type	String	Example: ProcessInstance
valueType	String	Example: "String", "Integer"
ValueTypes	TWSearchColumnMetaDataValueTypes	

This data type describes the data returned/available for a particular field search. When we execute a `TWSearch.execute()` the result is a `TWSearchResults` object. This contains a property called columns which is an array of `TWSearchColumnMetaData`.

A TWSearch can be loosely thought of as returning a rectangular array of data. Each record returned will contain some number of columns. As such, a cell in this array can be thought of as having a row number and a column number. Since the number of columns returned (and their order) is governed by the search being invoked, we need a "map" to be able to allow us to understand which column is which. This data type is such a map.

See also:

Data Type – TWSearchColumnMetaDataValueTypes

Property	Type
Boolean	String
DateTime	String
Decimal	String
Integer	String
String	String

This data type is available from `TWSearchColumnMetaData.ValueTypes`.

See also:

Data Type – TWSearchColumnProcessColumns

Property	Type
Name	String

This data type can be reached from `TWSearchColumn.ProcessColumns`.

I have found the following code useful to examine the values:

```
var x = {
 processInstanceColumns: {
 CaseFolderID: TWSearchColumn.ProcessInstanceColumns.CaseFolderID, CaseFolderServerName:
TWSearchColumn.ProcessInstanceColumns.CaseFolderServerName, DueDate:
TWSearchColumn.ProcessInstanceColumns.DueDate,
 ID: TWSearchColumn.ProcessInstanceColumns.ID, Name:
TWSearchColumn.ProcessInstanceColumns.Name, ProcessApp:
TWSearchColumn.ProcessInstanceColumns.ProcessApp, Snapshot:
TWSearchColumn.ProcessInstanceColumns.Snapshot, StartingDocumentID:
TWSearchColumn.ProcessInstanceColumns.StartingDocumentID, StartingDocumentServerName:
TWSearchColumn.ProcessInstanceColumns.StartingDocumentServerName, Status:
TWSearchColumn.ProcessInstanceColumns.Status
 },
 processColumns: {
 Name: TWSearchColumn.ProcessColumns.Name
 },
 taskColumns: {
 Activity: TWSearchColumn.TaskColumns.Activity, AssignedToRole:
TWSearchColumn.TaskColumns.AssignedToRole, AssignedToRoleDisplayName:
TWSearchColumn.TaskColumns.AssignedToRoleDisplayName, AssignedToUser:
TWSearchColumn.TaskColumns.AssignedToUser,
 ClosedBy: TWSearchColumn.TaskColumns.ClosedBy, ClosedDate:
```

```
TWSearchColumn.TaskColumns.ClosedDate,
 DueDate: TWSearchColumn.TaskColumns.DueDate,
 ID: TWSearchColumn.TaskColumns.ID,
 Priority: TWSearchColumn.TaskColumns.Priority, ReadDate:
TWSearchColumn.TaskColumns.ReadDate,
 ReceivedDate: TWSearchColumn.TaskColumns.ReceivedDate, ReceivedFrom:
TWSearchColumn.TaskColumns.ReceivedFrom,
 SentDate: TWSearchColumn.TaskColumns.SentDate,
 Status: TWSearchColumn.TaskColumns.Status,
 Subject: TWSearchColumn.TaskColumns.Subject
 }
 };
```

See also:

Data Type – TWSearchColumnProcessInstanceColumns

Property	**Type**
CaseFolderID	String
CaseFolderServerName	String
DueDate	String
ID	String
Name	String
ProcessApp	String
Snapshot	String
StartingDocumentID	String
StartingDocumentServerName	String
Status	String

This data type can be reached from `TWSearchColumn.ProcessInstanceColumns`.

I have found the following code useful to examine the values:

```
var x = {
 processInstanceColumns: {
 CaseFolderID: TWSearchColumn.ProcessInstanceColumns.CaseFolderID, CaseFolderServerName:
TWSearchColumn.ProcessInstanceColumns.CaseFolderServerName, DueDate:
TWSearchColumn.ProcessInstanceColumns.DueDate,
 ID: TWSearchColumn.ProcessInstanceColumns.ID, Name:
TWSearchColumn.ProcessInstanceColumns.Name, ProcessApp:
TWSearchColumn.ProcessInstanceColumns.ProcessApp, Snapshot:
TWSearchColumn.ProcessInstanceColumns.Snapshot, StartingDocumentID:
TWSearchColumn.ProcessInstanceColumns.StartingDocumentID, StartingDocumentServerName:
TWSearchColumn.ProcessInstanceColumns.StartingDocumentServerName, Status:
TWSearchColumn.ProcessInstanceColumns.Status
 },
 processColumns: {
 Name: TWSearchColumn.ProcessColumns.Name
 },
 taskColumns: {
 Activity: TWSearchColumn.TaskColumns.Activity, AssignedToRole:
TWSearchColumn.TaskColumns.AssignedToRole, AssignedToRoleDisplayName:
TWSearchColumn.TaskColumns.AssignedToRoleDisplayName, AssignedToUser:
TWSearchColumn.TaskColumns.AssignedToUser,
 ClosedBy: TWSearchColumn.TaskColumns.ClosedBy, ClosedDate:
TWSearchColumn.TaskColumns.ClosedDate,
 DueDate: TWSearchColumn.TaskColumns.DueDate,
 ID: TWSearchColumn.TaskColumns.ID,
 Priority: TWSearchColumn.TaskColumns.Priority, ReadDate:
TWSearchColumn.TaskColumns.ReadDate,
 ReceivedDate: TWSearchColumn.TaskColumns.ReceivedDate, ReceivedFrom:
TWSearchColumn.TaskColumns.ReceivedFrom,
 SentDate: TWSearchColumn.TaskColumns.SentDate,
 Status: TWSearchColumn.TaskColumns.Status,
 Subject: TWSearchColumn.TaskColumns.Subject
 }
 };
```

See also:

Data Type – TWSearchColumnTaskColumns

Property	Type
Activity	String
AssignedToRole	String
AssignedToRoleDisplayName	String
AssignedToUser	String
ClosedBy	String
ClosedDate	String
DueDate	String
ID	String
Priority	String
ReadDate	String
ReceivedDate	String
ReceivedFrom	String
SentDate	String
Status	String
Subject	String

This data type can be reached from `TWSearchColumn.TaskColumns`.

I have found the following code useful to examine the values:

```
var x = {
 processInstanceColumns: {
 CaseFolderID: TWSearchColumn.ProcessInstanceColumns.CaseFolderID, CaseFolderServerName:
TWSearchColumn.ProcessInstanceColumns.CaseFolderServerName, DueDate:
TWSearchColumn.ProcessInstanceColumns.DueDate,
 ID: TWSearchColumn.ProcessInstanceColumns.ID, Name:
```

```
TWSearchColumn.ProcessInstanceColumns.Name, ProcessApp:
TWSearchColumn.ProcessInstanceColumns.ProcessApp, Snapshot:
TWSearchColumn.ProcessInstanceColumns.Snapshot, StartingDocumentID:
TWSearchColumn.ProcessInstanceColumns.StartingDocumentID, StartingDocumentServerName:
TWSearchColumn.ProcessInstanceColumns.StartingDocumentServerName, Status:
TWSearchColumn.ProcessInstanceColumns.Status
 },
 processColumns: {
 Name: TWSearchColumn.ProcessColumns.Name
 },
 taskColumns: {
 Activity: TWSearchColumn.TaskColumns.Activity, AssignedToRole:
TWSearchColumn.TaskColumns.AssignedToRole, AssignedToRoleDisplayName:
TWSearchColumn.TaskColumns.AssignedToRoleDisplayName, AssignedToUser:
TWSearchColumn.TaskColumns.AssignedToUser,
 ClosedBy: TWSearchColumn.TaskColumns.ClosedBy, ClosedDate:
TWSearchColumn.TaskColumns.ClosedDate,
 DueDate: TWSearchColumn.TaskColumns.DueDate,
 ID: TWSearchColumn.TaskColumns.ID,
 Priority: TWSearchColumn.TaskColumns.Priority, ReadDate:
TWSearchColumn.TaskColumns.ReadDate,
 ReceivedDate: TWSearchColumn.TaskColumns.ReceivedDate, ReceivedFrom:
TWSearchColumn.TaskColumns.ReceivedFrom,
 SentDate: TWSearchColumn.TaskColumns.SentDate,
 Status: TWSearchColumn.TaskColumns.Status,
 Subject: TWSearchColumn.TaskColumns.Subject
 }
 };
```

See also:

Data Type – TWSearchColumnTypes

Constants used in TWSearch.

Property	**Type**

BusinessData	String
Process	String
ProcessInstance	String
Task	String

See also:

Data Type – TWSearchCondition

The search condition names a column in the data, an operator to be executed against that column and a parameter for the operation.

|| |**Property**|**Type**| |column|TWSearchColumn| |operator|TWSearchConditionOperations| |value|TWObject| |Operations|TWSearchConditionOperations|

See also:

Data Type – TWSearchConditionOperations

|**Property**|**Type**| |Contains|String| |Equals|String| |GreaterThan|String| |LessThan|String| |NotEquals|String| |StartsWith|String|

This type can be accessed from `TWSearchCondition.Operations`.

See also:

Data Type – TWSearchOptions

A `TWSearchOptions` instance provides additional parameters to a `TWSearch()`. The properties of this object are:

Property	Type
isTieBreakerSorting	boolean

`isTrieBreakerSorting` (`Boolean`) – When a search is performed and the results returned, there may not be enough information returned to uniquely sort the results. If the value of this option is `false` (default) then tied sorts are in an undefined order. However, if set to true, additional logical columns are added to the sort to disambiguate the results. The columns used for the additional sorting are:

See also:

Data Type – TWSearchOrganizeByTypes

Property	Type
ProcessInstance	String
Task	String

See also:

Data Type – TWSearchResults

A `TWSearchResults` object is returned from the method calls of `TWSearch()`.

It contains the following data:

Property	Type
columns	TWSearchColumnMetaData []
rows	TWSearchResultRow []

See also:

Data Type – TWSearchResultRow

This contains:

Property	Type
values	TWObject []

See also:

Data Type – TWService

Properties

id

String

name

String

type

String

The type of service that this object represents. See: Data Type – TWServiceTypes

Types

TWServiceTypes

The different types of services that are possible.

Methods

Map execute(Map inputParams)

Execute the instance of this service. The inputParams are the inputs to the service.

String getTypeName()

Get the type that this service implements.

An instance of this object is returned by the global function called `tw.system.model.findServiceByName`. The signature of this function is:

TWService tw.system.model.findServiceByName(String serviceName)

If one simply wants to a call a service, the function:

Map tw.system.executeServiceByName(String serviceName, Map inputs)

can be used. The service name appears to be a simple service name.

Data Type – TWServiceTypes

Properties

Ajax

String

An Ajax service implementation.

GeneralSystem

String

A General System service implementation.

Human

String

A Human service implementation.

Installation

String

An Installation service implementation.

Integration

String

An Integration service implementation.

Rule

String

A Rule service implementation.

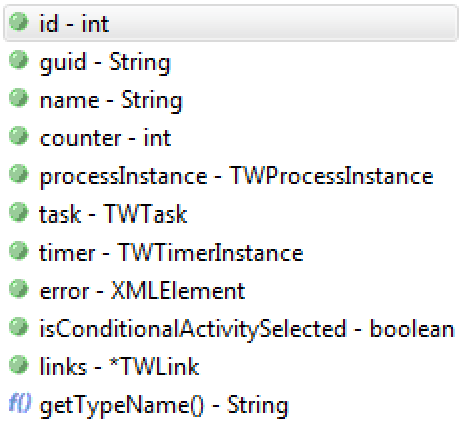
Data Type – TWStep

The current step in the process (BPD) can be found from the variable:

tw.system.step

This returns a `TWStep` object.

(as of 8.5)



id - int
guid - String
name - String
counter - int
processInstance - TWProcessInstance
task - TWTask
timer - TWTimerInstance
error - XMLElement
isConditionalActivitySelected - boolean
links - *TWLink
getTypeName() - String

The timer property is relevant if the step is a timer. The pre-assignment option of the BPD activity can be used to get the timerId and **not** the post-assignment value.

See also:

Data Type – TWTask

As a process instance operates, it can create tasks that users are expected to perform. The `TWTask` data type represents a model of this task that can be accessed from BPM applications.

Within a service, the current task can be found from:

`tw.system.currentTask`

Property	Type
activationTime	TWDate

Property	Type
assignedTo	TWObject
atRiskDate	TWDate
completionTime	TWDate
dueDate	TWDate
flowObjectId	String
id	String
isAtRisk	Boolean
localId	String
narrative	String
originator	String
owner	String
phaseId	String
priorities	TWTaskPriorities
priority	String
priorityValue	Integer
processActivityDescription	String
processActivityBane	String
processActivityRichDescription	String
processInstance	TWProcessInstance
processInstanceStep	TWProcessStepInfo
startDate	TWDate
state	String
status	String
Statuses	TWTaskStatuses
subject	String
transactions	Map[]

|| |long[] complete(TWUser user, Map outputValues)| |String[] getAvailableActions(String actionsFilter[])| |String getTypeName()| |void reassignBackToRole()| |reassignTo(TWUser usersOrRoles)

reassignTo(TWRole usersOrRoles)

reassignTo(TWTeam usersOrRoles)

reassignTo(String usersOrRoles)

reassignTo(Array usersOrRoles)

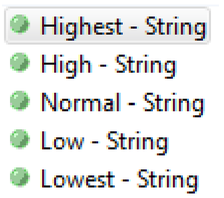
reassignTo(String usersOrRoles[]);| |void start(TWUser user)|

(As of 8.5.5)

The `assignedTo` field contains the identity of the user or role to which the task is currently assigned. This may be either a `TWUser` or a `TWRole`.

In addition, there are some constant definitions found at:

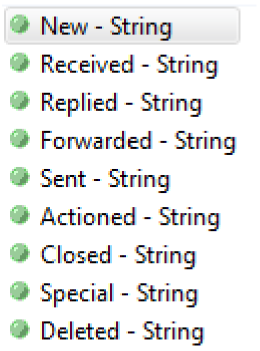
TWTask.Priorities



- Highest - String
- High - String
- Normal - String
- Low - String
- Lowest - String

The String values of these are:

TWTask.Statuses



- New - String
- Received - String
- Replied - String
- Forwarded - String
- Sent - String
- Actioned - String
- Closed - String
- Special - String
- Deleted - String

We can retrieve an arbitrary task as long as we know it's id. We can do this using:

tw.system.findTaskByID()

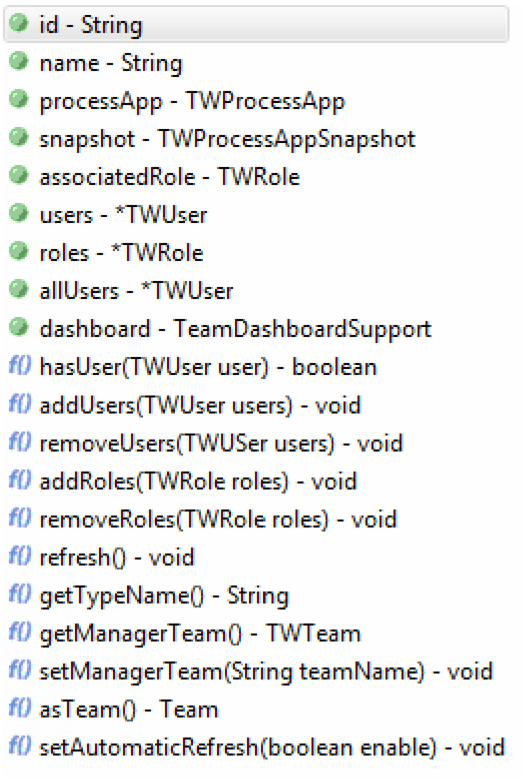
Notice that there are two priority properties. One is called "`priority`" and the other is called "`priorityValue`". These two values appear to be in sync with each other. The `priority` property contains a string representation while the `priorityValue` contains a numeric representation. Changing the priority of a task changes both values. The `priorityValue` can be used when a numeric comparison is required … eg … to answer questions such as "When priority is higher than 'normal'". This would not be easy to achieve if all we had were the string representation.

See also:

Data Type – TWTeam

(As of 8.5)

This data type represents a team within IBM BPM. There are a number of APIs that can be used to obtain a team such as:

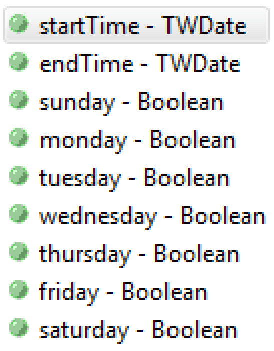


```
id - String
name - String
processApp - TWProcessApp
snapshot - TWProcessAppSnapshot
associatedRole - TWRole
users - *TWUser
roles - *TWRole
allUsers - *TWUser
dashboard - TeamDashboardSupport
hasUser(TWUser user) - boolean
addUsers(TWUser users) - void
removeUsers(TWUSer users) - void
addRoles(TWRole roles) - void
removeRoles(TWRole roles) - void
refresh() - void
getTypeName() - String
getManagerTeam() - TWTeam
setManagerTeam(String teamName) - void
asTeam() - Team
setAutomaticRefresh(boolean enable) - void
```

See also:

Data Type – TWTimePeriod

The `TWTimePeriod` is used to represent a period of time and which days that period is applicable.



startTime - TWDate
endTime - TWDate
sunday - Boolean
monday - Boolean
tuesday - Boolean
wednesday - Boolean
thursday - Boolean
friday - Boolean
saturday - Boolean

See also:

Data Type – TWTimeSchedule

A `TWTimeSchedule` represents an interval of time during which work is processed. An instance of this object can be create in JavaScript through:

var var1 = new tw.object.TWTimeSchedule();

Once created, the properties of this new object can be set. Once the `TWTimeSchedule` has been built, the new Time Schedule can be registered for use with the JavaScript API:

tw.system.addTimeSchedule(TWTimeSchedule timeSchedule);

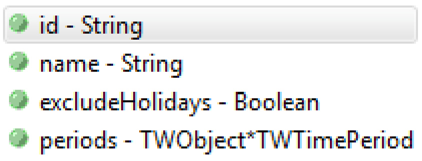
To retrieve an existing schedule, execute:

tw.system.findTimeScheduleByName(String name)

This returns a `TWTimeSchedule` object.

To delete a previously registered Time Schedule, the JavaScript call is:

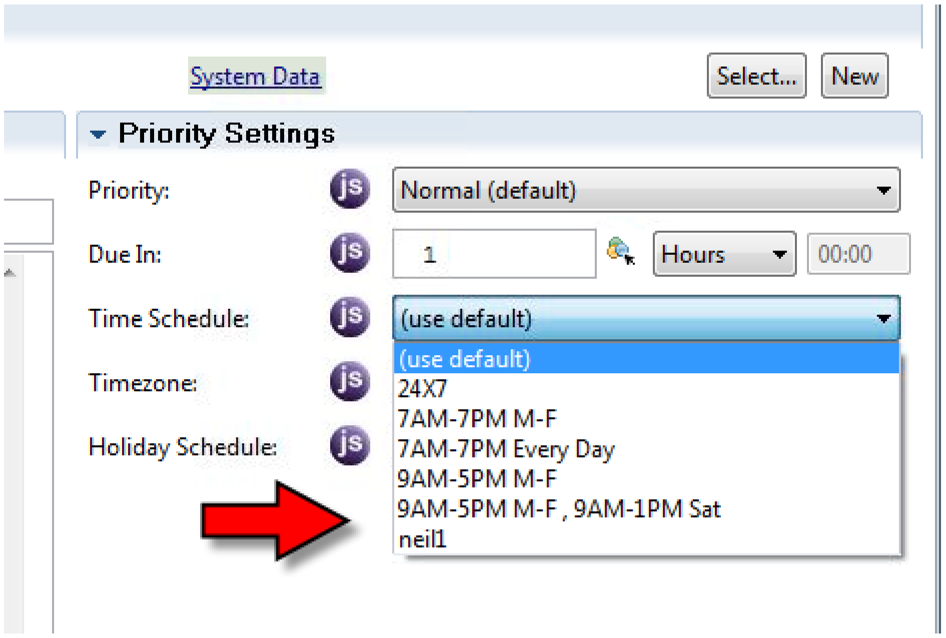
tw.system.removeTimeSchedule(String id);



id - String
name - String
excludeHolidays - Boolean
periods - TWObject*TWTimePeriod

The product provides a set of pre-existing Time Schedules

The list of currently defined Time Schedules can be seen from the pull-down for activities:



Priority: Normal (default)

Due In: 1 Hours 00:00

Time Schedule: (use default)

Timezone:

Holiday Schedule:

(use default)
24X7
7AM-7PM M-F
7AM-7PM Every Day
9AM-5PM M-F
9AM-5PM M-F , 9AM-1PM Sat
neil1

These time schedules can be found in the list at `tw.system.timeSchedules`. The method:

tw.system.addTimeSchedule()

will add a time schedule to this list.

Here is some sample code for adding a Time Schedule:

var tim1 = new tw.object.TWTimeSchedule(); tim1.name = "neil1"; tim1.periods = new tw.object.listOf.TWTimePeriod(); var period1 = new tw.object.TWTimePeriod(); period1.startTime = new tw.object.Date(); period1.endTime = new tw.object.Date(); period1.sunday = false; period1.saturday = false; period1.monday = true; period1.tuesday = true; period1.wednesday = true; period1.thursday = true; period1.friday = true; tim1.periods[0] = period1; var added = tw.system.addTimeSchedule(tim1); log.info("ret = " + added);

There is an IBM BPM supplied function called `tw.system.calculateBusinessDate()` that will calculate a future date/time based on an original date, a logical increment and a `TWWorkSchedule`.

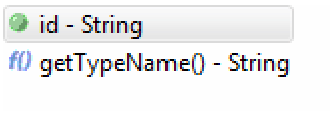
TWDate tw.system.calculateBusinessDate(TWDate originalDate, Integer delta, String units, TWWorkSchedule workSchedule)

The "units" property can be one of:

See also:

Data Type – TWTimerInstance

An instance of this type of object can be retrieved from an instance of the `TWStep` object through its `timer` property.



The timer instance id may be used in the `tw.system.rescheduleTimer()` method. This takes two parameters. The first is the id of the timer instance. The second is the date/time at which the timer should fire. The primary purpose of this function is to re-schedule a ticking timer to fire at a different time. Take care when trying to obtain the timerId as it seems that we have to use the pre-assignment of the attached timer activity and not the post-assignment value.

See also:

Data Type – TWUser

Because various functions can return an instance of either a `TWUser` or a `TWRole`, the `getTypeName()` function is common to both. Calling this function for a `TWUser` object will return the String "`TWUser`". This can be used to determine which methods and properties are available based on the type of object it is determined to be.

Attribute	Type
attributes	Record
dashboard	UserDashboardSupport
fullName	String
id	int
isActive	boolean
name	String
participantGroups	TWParticipantGroup[]
roles	TWRole[]
savedSearches	TWSavedSearch[]
teams	TWTeam[]

Method	

TWUserLocalePreferences getLocalePreferences(TWUser user)	
String getTypeName	
boolean isInParticipantGroup(	
TWParticipantGroup participantGroup)	
boolean isInRole(String	TWRole role)
boolean isInTeam(TWTeam team)	
void removeAttribute(String name)	
StreamsForUser retrieveUserStream(	

String sortOrder, String page, String pageSize, TWDate sinceDataTime, String taskId)|| |void setAttributeValue(String name, String value)|| |void setLocalePreferences(TWUser user, TWUserLocalePreferences localePreferences)||

The property called `attributes` is a `Record` data structure that contains the properties for the user definition. What this means is that we can determine the value of a user attribute directly. For example, if a user definition has an attribute called "level", we can find the user's level attribute value with:

myUser.attributes.level;

Be cautious about using either the `Record` methods `toXML()` or `toXMLString()`. The reason for this is that the IBPM default attributes such as "`Task Email Address`" contain spaces in their names and the XML functions throw exceptions when they attempt to build XML using names with spaces as element names.

We can find a `TWUser` object if we know the userid/name of that user. The method

tw.system.org.findUserByName("<name>")

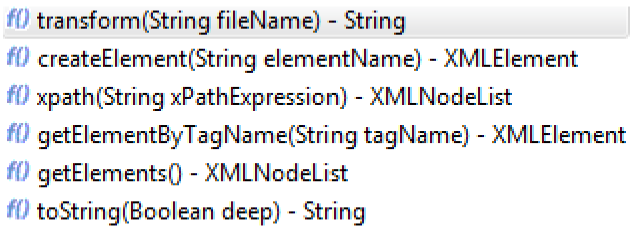
will return a `TWUser` instance.

This data type can not be sent to "`log.info()`".

See also:

Data Type – XMLDocument

This data type is defined in the System Data toolkit.

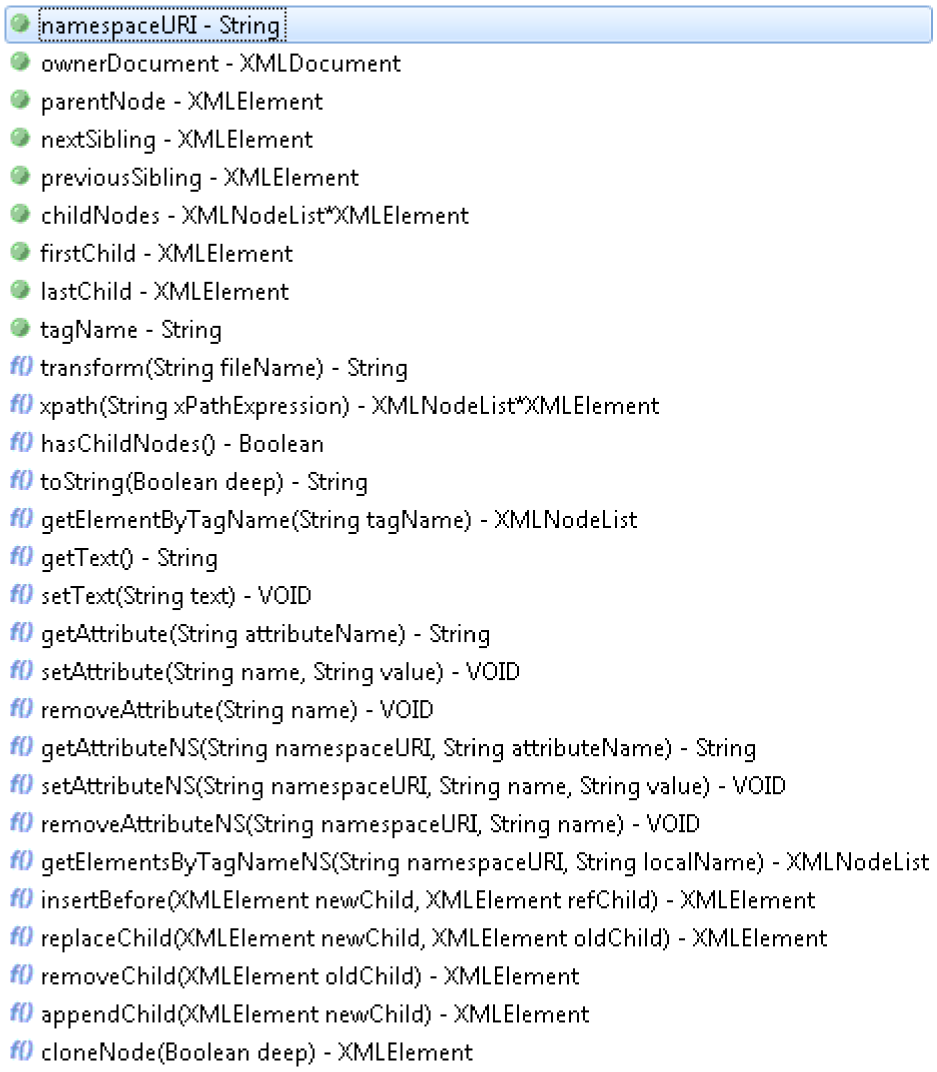


f() transform(String fileName) - String
f() createElement(String elementName) - XMLElement
f() xpath(String xPathExpression) - XMLNodeList
f() getElementByTagName(String tagName) - XMLElement
f() getElements() - XMLNodeList
f() toString(Boolean deep) - String

See also:

Data Type – XMLElement

The `XLMElement` object represents a node in a DOM tree. Each instance of this object provides navigators to the next/previous sibling, first/last child and the parent as well as access to the document as a whole. Through this object common DOM walks and searches can be executed to perform quite sophisticated data access and manipulation. Each node has the following:

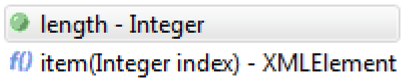

namespaceURI - String
ownerDocument - XMLDocument
parentNode - XMLElement
nextSibling - XMLElement
previousSibling - XMLElement
childNodes - XMLNodeList*XMLElement
firstChild - XMLElement
lastChild - XMLElement
tagName - String
transform(String fileName) - String
xpath(String xPathExpression) - XMLNodeList*XMLElement
hasChildNodes() - Boolean
toString(Boolean deep) - String
getElementByTagName(String tagName) - XMLNodeList
getText() - String
setText(String text) - VOID
getAttribute(String attributeName) - String
setAttribute(String name, String value) - VOID
removeAttribute(String name) - VOID
getAttributeNS(String namespaceURI, String attributeName) - String
setAttributeNS(String namespaceURI, String name, String value) - VOID
removeAttributeNS(String namespaceURI, String name) - VOID
getElementsByTagNameNS(String namespaceURI, String localName) - XMLNodeList
insertBefore(XMLElement newChild, XMLElement refChild) - XMLElement
replaceChild(XMLElement newChild, XMLElement oldChild) - XMLElement
removeChild(XMLElement oldChild) - XMLElement
appendChild(XMLElement newChild) - XMLElement
cloneNode(Boolean deep) - XMLElement

We can walk through an XML document using the `firstChild` and `nextSibling` mechanisms. We can also access the children by name. For example:

See also:

Data Type – XMLNodeList

A list of XMLElement objects.

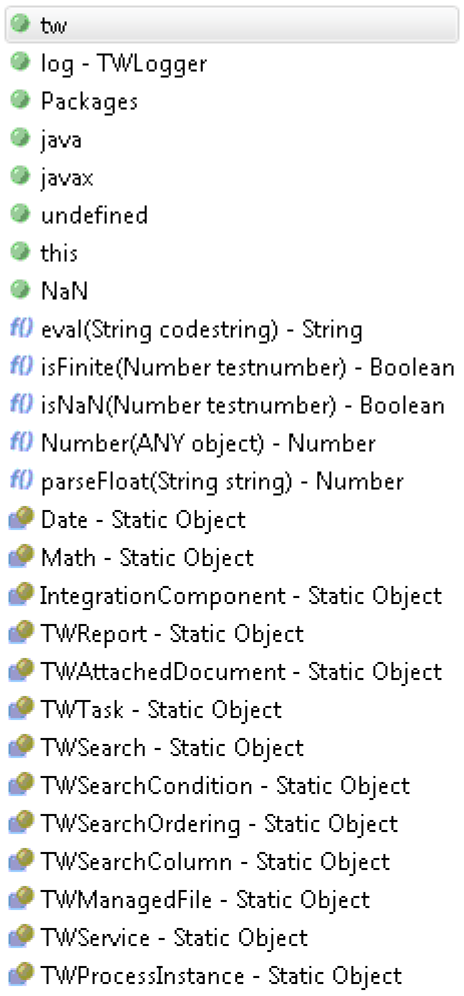

length - Integer
item(Integer index) - XMLElement

See also:

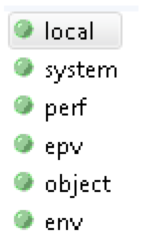
JavaScript Libraries

IBM BPM provides predefined JavaScript libraries that can be used in the applications.

Root:


tw
log - TWLogger
Packages
java
javax
undefined
this
NaN
eval(String codestring) - String
isFinite(Number testnumber) - Boolean
isNaN(Number testnumber) - Boolean
Number(ANY object) - Number
parseFloat(String string) - Number
Date - Static Object
Math - Static Object
IntegrationComponent - Static Object
TWReport - Static Object
TWAttachedDocument - Static Object
TWTask - Static Object
TWSearch - Static Object
TWSearchCondition - Static Object
TWSearchOrdering - Static Object
TWSearchColumn - Static Object
TWManagedFile - Static Object
TWService - Static Object
TWProcessInstance - Static Object

tw.*


local
system
perf
epv
object
env

JavaScript package - tw.system.*

The package called `tw.system.*` contains the following (as of 8.5):

|| ||| |currentAdHocActivtityInstance|ActivityInstance| |currentProcess|TWProcess| |currentProcessInstance|TWProcessInstance| |defaultHolidaySchedule|TWHolidaySchedule| |defaultTimeSchedule|TWTimeSchedule| |defaultTimeZone|String| |enclosingCaseInstance|CaseReference| |holidaySchedules|TWHolidaySchedule[]| |install|TWInstallNamespace| |model|TWModelNamespace| |org|TWBPDSystemOrgNamespace| |process|TWBPDSystemBPDNameSpace| |step|TWProcessStepInfo| |timeSchedules|TWTimeSchedule[]| |user|TWUser|

|| || |boolean addHolidaySchedule(TWHolidaySchedule holidaySchedule)| |boolean addTimeSchedule(TWTimeSchedule timeSchedule)| |String bidiTransform(String src, String inputFormat, String outputFormat, boolean summetricSwapping)| |TWDate calculateBusinessData(TWDate originalDate, Integer delta, String units, TWWorkSchedule workSchedule)| |void cancelStep(Integer stepId)| |String convertIDToDB()| |String createFromSql(String variableTypeName, String jndiName, String sqlQuery, ANY params)| |void createFromXmlElement(String variableTypeName, Element XML)| |BPMFailedOperation[] deleteAlertDefinitions(String ids[], Boolean checkAuthorization)| |String escapeHtml(String content)| |Map executeServiceByName(String name, Map inputValues)| |ActivityInstance findActivityInstanceByID(String activityId)| |TWDocument findDocumentByID(String documentId)| |TWHelpRequest findHelpRequestByID(String id)| |TWHolidaySchedule findHolidayScheduleByName(String name)| |TWProcessInstance findProcessInstanceByID(String id)| |TWTask findTaskByID(Integer | String taskId)| |TWTimeSchedule findTimeScheduleByName(String name)| |BPMAlertDefinition[] getAlertDefinitions(String categoryFilter[], Boolean checkAuthorizations)| |BPMAlertDefinitionsStatusResponse getAlertDefinitionsStatus(String ids[], Boolean checkAuthorization)| |String getEnvironmentVariableValue(String processAppAcronym, String snapshotAcronym, String environmentVariableName)| |boolean removeHolidaySchedule(String id)| |boolean removeTimeSchedule(String id)| |void rescheduleTimer(String timerId, TWDate newFireTime)| |InstanceListData retrieveInstanceList(InstanceListProperties properties, Integer maxRows, Integer beginIndex, Boolean checkAuthorization)| |ProcessSummary[] retrieveProcessSummaries(String searchFilter, Boolean checkAuthorization)| |TaskDueData retrieveTaskDueData(TaskDueProperties properties, String timeZone)| |TaskListData retrieveTaskList(TaskListProperties properties, Integer maxRows, Integer beginIndex, String timezone)| |TeamTaskSummary retrieveTeamSummaries(String searchFilter, String timeZoneAsString, Boolean checkAuthorization)| |BPMAlertDefinition[] saveAlertDefinitions(BPMAlertDefinition definitions[], Boolean importMode, Boolean checkAuthorization)| |void touchVariable(String name)| |String unescapeHtml(String content)| |boolean updateHolidaySchedule(TWHolidaySchedule holidaySchedule)| |boolean

updateTimeSchedule(TWTimeSchedule timeSchedule)| |String variableTypeForVariable(String fullyQualifiedVariableName)|

Method: executeServiceByName(name, inputValues)

The documentation on this method appears to be wrong. It says it returns a `TWProcessInstance` where in reality, it seems to return a `Map` object which contains the output values. This makes more sense.

See also:

Method: tw.system.retrieveInstanceList()

Retrieve a list of instances. The returned data is an instance of "`InstanceListData`".

InstanceListData retrieveInstanceList(InstanceListProperties properties, Integer maxRows, Integer beginIndex, Boolean checkAuthorization)

See also:

Method: tw.system.retrieveTaskList()

Retrieve a list of tasks. The returned data is an instance of "`TaskListData`".

Signature:

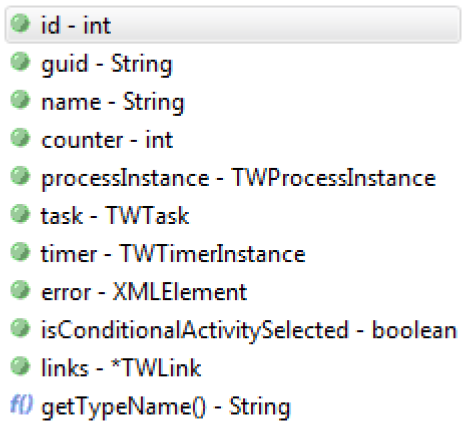
tw.system.retrieveTaskList(TaskListProperties properties, Integer maxRows, Integer beginIndex, String timezone) – TaskListData

---

See also:

JavaScript Package - tw.system.step.*

(as of 8.5.5)



id - int
guid - String
name - String
counter - int
processInstance - TWProcessInstance
task - TWTask
timer - TWTimerInstance
error - XMLElement
isConditionalActivitySelected - boolean
links - *TWLink
getTypeName() - String

See also:

JavaScript Package – tw.system.org.*

(as of 8.5.7)

Attribute	
team	TeamNamespace

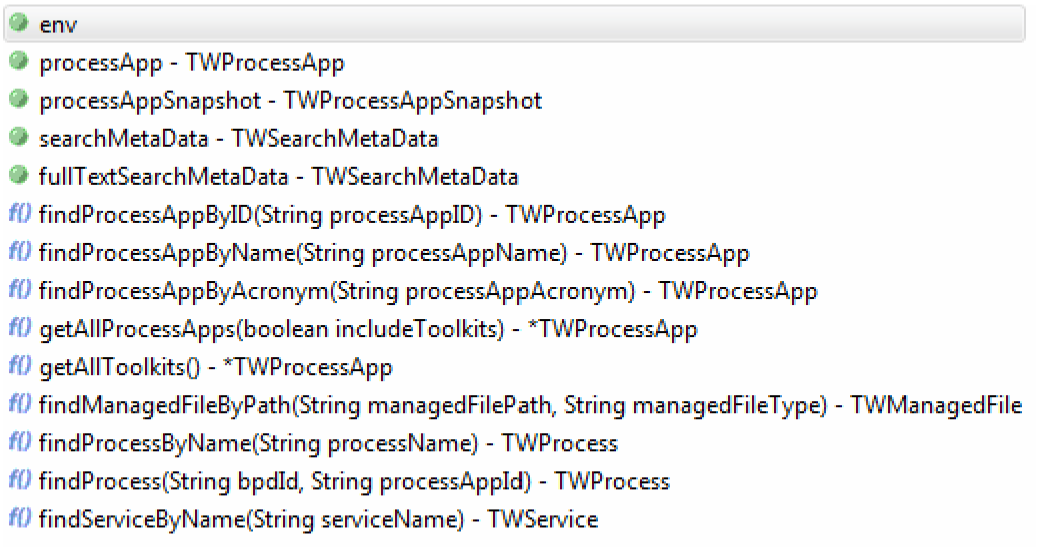
Method
TWRole createRole(

String roleName, String roleDisplayName, String roleDescription, Boolean checkAuthorization)| |TWUser[] findAllUsersForRoles(TWRole roles[])| |TWUser[] findCommonUsersForRoles(TWRole roles[])| |TWParticipantGroup findParticipantGroupByID(String participantId)| |TWParticipantGroup findParticipantGroupByName(String roleName)| |TWRole findRoleById(Integer roleId)| |TWRole findRoleByName(String roleName)| |TWTeam findTeam(String teamId, String processAppId)| |TWTeam findTeamById(String teamId)| |TWTeam findTeamByName(String teamName)| |TWUser findUserById(Integer userId)| |TWUser findUserByName(String userName)| |TWParticipantGroup[] getAllParticipantGroups()| |TWRole[] getAllRoles()| |TWTeam[] getAllTeams()| |TWUser[] getAllUsers()| |void removeRole(String roleName, Boolean checkAuthorization)|

See also:

JavaScript Package – tw.system.model.*

As of 8.5.



env
processApp - TWProcessApp
processAppSnapshot - TWProcessAppSnapshot
searchMetaData - TWSearchMetaData
fullTextSearchMetaData - TWSearchMetaData
findProcessAppByID(String processAppID) - TWProcessApp
findProcessAppByName(String processAppName) - TWProcessApp
findProcessAppByAcronym(String processAppAcronym) - TWProcessApp
getAllProcessApps(boolean includeToolkits) - *TWProcessApp
getAllToolkits() - *TWProcessApp
findManagedFileByPath(String managedFilePath, String managedFileType) - TWManagedFile
findProcessByName(String processName) - TWProcess
findProcess(String bpdId, String processAppId) - TWProcess
findServiceByName(String serviceName) - TWService

This is a particularly interesting package. From within here we can interrogate the meta data of our environment.

See also:

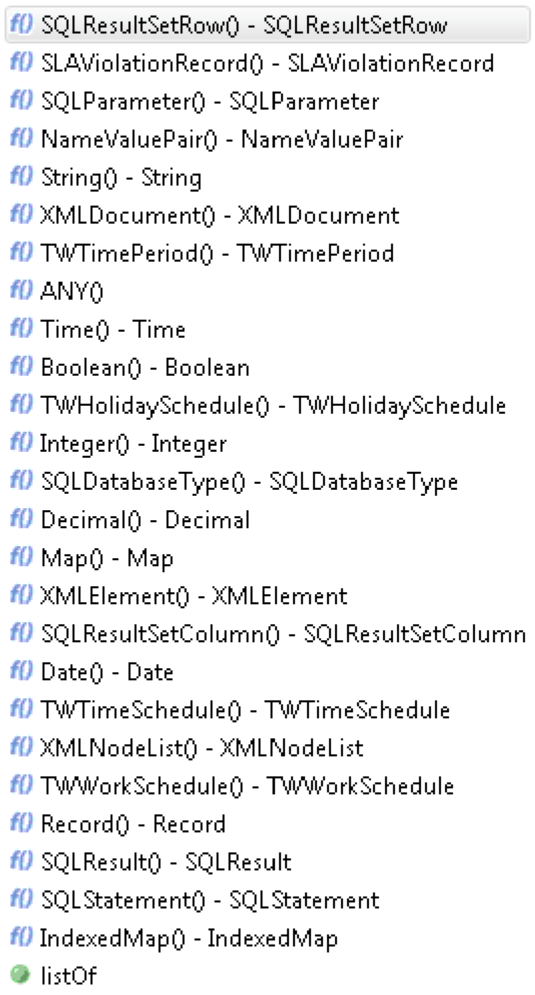
Getting a list of Process Apps

In the following snippet, we see us getting the list of all process apps known to the system and logging their names. Each process app is described by a `TWProcessApp` object which has many other properties beyond name.

var listOfProcessApps = tw.system.model.getAllProcessApps(); for (var i=0; i<listOfProcessApps.length; i++) { log.info(listOfProcessApps[i].name); }

See also:

JavaScript Package - tw.object.*



SQLResultSetRow() - SQLResultSetRow
SLAViolationRecord() - SLAViolationRecord
SQLParameter() - SQLParameter
NameValuePair() - NameValuePair
String() - String
XMLDocument() - XMLDocument
TWTimePeriod() - TWTimePeriod
ANY()
Time() - Time
Boolean() - Boolean
TWHolidaySchedule() - TWHolidaySchedule
Integer() - Integer
SQLDatabaseType() - SQLDatabaseType
Decimal() - Decimal
Map() - Map
XMLElement() - XMLElement
SQLResultSetColumn() - SQLResultSetColumn
Date() - Date
TWTimeSchedule() - TWTimeSchedule
XMLNodeList() - XMLNodeList
TWWorkSchedule() - TWWorkSchedule
Record() - Record
SQLResult() - SQLResult
SQLStatement() - SQLStatement
IndexedMap() - IndexedMap
listOf

JavaScript Package - tw.system.model

Methods:

All methods part of the `tw.system.model` package

Method	**Description**
`findManagedFileByPath`	Returns an instance of the managed file represented by the call `TWManagedFile`
`findProcessAppByAcronym`	Returns a `TWProcessApp` based on the acronym of the Process Application or Toolkit
`findProcessAppByID`	Returns a `TWProcessApp` based on the identifier of the Process Application or Toolkit
`findProcessAppByName`	Return a `TWProcessApp` based on its name.
`findProcessByName`	Returns a `TWProcess` based on the name.

Method	Description
`findServiceByName`	Returns a `TWService` based on the name.
`findParticipantGroupByName`	
`findPartticipantGroupByID`	
`getAllProcessApps`	Returns an array of `TWProcessApp` for all the Process Apps and Toolkits
`getAllToolkits`	Returns an array of `TWProcessApp` for all the Toolkits

tw.system

Methods for tw.system

Method	Description
`rescheduleTimer`	
`variableTypeForVariable`	
`createFromSql`	
`createFromXmlElement`	
`cancelStep`	
`touchVariable`	
`addTimeSchedule`	
`findTimeScheduleByName`	
`removeTimeSchedule`	
`addHolidaySchedule`	
`findHolidayScheduleByName`	
`removeHolidaySchedule`	
`startProcessByName`	Start an instance of a named process. Returns a TWProcessInstance.
`executeServiceByName`	
`convertIDToDB`	
`findHelpRequestById`	
`calculateBusinessDate`	

TWProcess attributes

id

String

ro

The ID of the BPD

name

String

ro

The name of the BPD

description

String

ro

The description of the BPD

searchMetaData

TWSearchMetaData

ro

processApp

TWProcessApp

ro

snapshot

TWProcessAppSnapshot

ro

|| |TWProcess methods|| |getTypeName|| |startNew||

Creating Business Object instances in JavaScript

In IBPM PD we can create custom Business Object data type definitions. When a Business Object type variable is created, it must be initialized. This can be done through JavaScript with the code:

tw.local.myVariable = new tw.object.*ComplexType*();

A similar story is also true for data of type list:

tw.local.myListOfStrings = new tw.object.listOf.String();

Variables in a service

Variables in a service are defined in the variables section of that service. Three types of variables can be defined. These three types are input, output and private.

A JavaScript function called `tw.local.allVars()` returns a list of string names of the locally defined variables in the current service.

A function called `tw.local.toXML()` returns an XML Element object containing all the locally defined variables.

It is important to note that even though these look like JavaScript variables, the complex ones are **not**. They are in fact Java classes accessed through JavaScript syntax. This has an important ramification if JavaScript methods are used against them thinking that they are JavaScript objects. Many such functions imply won't work.

Data Type	JavaScript "typeof" result
String	string
Boolean	boolean
Date	object
Decimal	number

Data Type	JavaScript "typeof" result
Integer	number
Time	object

Complex data types can be serialized to XML with the following:

The Dojo Toolkit for JavaScript

Primarily of value to client-side JavaScript coding which executes in the browser, we may wish to use the services of the Dojo Toolkit. The Dojo Toolkit is supplied and loaded by Coach pages. Dojo is an open source set of JavaScript libraries that enhance the capabilities of native JavaScript. Dojo supplies both basic and advanced JavaScript functions. For details on the Dojo toolkit see:

Because Dojo is included with the IBPM coaches, the Dojo Toolkit functions can be used with custom solutions. Here are some of the more common Dojo patterns:

var x = dojo.byId("name of control");

This returns the DOM node of the control by name.

Commonly, Dojo is thought of as a visual language for building rich web pages. Although this is an important component of the Dojo Toolkit, it is not its sole function. However, if you are -approaching Dojo for building rich user interfaces, strongly consider using IBM's Rational Application Developer v8 (RAD) for development of the HTML and JavaScript for those functions. RAD includes a rich UI designer for Dojo development.

Searching for processes and tasks from JavaScript

The ability to search within IBPM's internal data is all about asking IBPM for information about the current operating environment. Examples of a search may include:

At a high level, the JavaScript object of type `TWSearch` can be used to execute a search. Think of this as an object having both state and methods. The state of an instance of TWSearch defines the columns to return. These are defined in the TWSearch.columns array. At least one column must be defined (otherwise we aren't asking for any results).

It has three primary methods:

The `TWSearch` object has a number of properties that are used to govern the data queried for and returned.

A call to `TWSearch` using the `execute()` method can return a `TWSearchResults` object.

For reference, the list of columns is shown below:

|**Tasks**|**Instances**|**Process**|**Business Data**| |-|-|-| |Activity|CaseFolderID|Name|| |AssignedToRole|CaseFolderServerName||| |AssignedToRoleDisplayName|DueDate||| |AssignedToUser|ID||| |ClosedBy|Name||| |ClosedDate|ProcessApp||| |DueDate|Snapshot||| |ID|StartingDocumentID||| |Priority|StartingDocumentServerName||| |ReadDate|Status||| |ReceivedDate|||| |ReceivedFrom|||| |SentDate|||| |Status|||| |Subject||||

Example:

```
log.info("Starting to find other tasks ...."); log.info("This process: " +
tw.system.currentProcessInstance.id); var col1 = new TWSearchColumn(); col1.name =
TWSearchColumn.ProcessInstanceColumns.ID; col1.type =
TWSearchColumn.Types.ProcessInstance; var search = new TWSearch(); search.columns =
[col1]; var condition = new TWSearchCondition(); condition.column = new TWSearchColumn();
condition.column.name = TWSearchColumn.ProcessInstanceColumns.ID; condition.column.type
= TWSearchColumn.Types.ProcessInstance; condition.operator =
TWSearchCondition.Operations.Equals; condition.value = "270"; search.conditions = new
Array(condition); var order1 = new TWSearchOrdering(); order1.column = col1; order1.order =
TWSearchOrdering.Orders.Descending; search.orderBy = new Array(order1); search.organizedBy
= TWSearch.OrganizeByTypes.ProcessInstance; var results = search.execute();
log.info("Result.rows.length = " + results.rows.length); for (var i=0; i<results.rows.length; i++) {
}
```

Notice that the `TWSearch` execute methods have an optional user/role filter. This allows the list of tasks/processes to be retrieved as though they were for a specific user. If this parameter is not specified, then a list of **all** Tasks/Instances are returned for all users.

Here is an example of some JavaScript which will retrieve all the tasks for all users:

```
log.info("Starting to find other tasks ...."); var col1 = new TWSearchColumn(); col1.name =
TWSearchColumn.ProcessInstanceColumns.ID; col1.type =
TWSearchColumn.Types.ProcessInstance; var col2 = new TWSearchColumn(); col2.name =
TWSearchColumn.TaskColumns.ID; col2.type = TWSearchColumn.Types.Task; var col3 = new
TWSearchColumn(); col3.name = TWSearchColumn.TaskColumns.Subject; col3.type =
TWSearchColumn.Types.Task; var search = new TWSearch(); search.columns = [col1, col2, col3];
search.organizedBy = TWSearch.OrganizeByTypes.Task; var order1 = new TWSearchOrdering();
order1.column = col2; order1.order = TWSearchOrdering.Orders.Descending; search.orderBy =
new Array(order1); var results = search.execute(); log.info("Result.rows.length = " +
results.rows.length); for (var i=0; i<results.rows.length; i++) { //log.info(results.rows[i].values);
var currentTaskInstance = new tw.object.TaskInstance(); currentTaskInstance.taskId =
results.rows[i].values[1]; currentTaskInstance.processId = results.rows[i].values[0];
currentTaskInstance.taskSubject = results.rows[i].values[2];
tw.local.taskInstance[tw.local.taskInstance.listLength] = currentTaskInstance; }
```

Here is another example. In this one, we are looking for all tasks associated with the process called "BAM1" which are not completed. From this, we want to know how many tasks are claimed and how many are not claimed.

```
var col1 = new TWSearchColumn(); col1.name = TWSearchColumn.ProcessColumns.Name;
col1.type = TWSearchColumn.Types.Process; var col2 = new TWSearchColumn(); col2.name =
TWSearchColumn.TaskColumns.Activity; col2.type = TWSearchColumn.Types.Task; var col3 =
new TWSearchColumn(); col3.name = TWSearchColumn.TaskColumns.Status; col3.type =
TWSearchColumn.Types.Task; var col4 = new TWSearchColumn(); col4.name =
TWSearchColumn.TaskColumns.AssignedToUser; col4.type = TWSearchColumn.Types.Task; var
search = new TWSearch(); search.columns = [col1, col2, col3, col4]; var condition = new
TWSearchCondition(); condition.column = new TWSearchColumn(); condition.column.name =
TWSearchColumn.ProcessColumns.Name; condition.column.type =
TWSearchColumn.Types.Process; condition.operator = TWSearchCondition.Operations.Equals;
condition.value = "BAM1"; var condition2 = new TWSearchCondition(); condition2.column = new
TWSearchColumn(); condition2.column.name = TWSearchColumn.TaskColumns.Status;
condition2.column.type = TWSearchColumn.Types.Task; condition2.operator =
TWSearchCondition.Operations.Equals; condition2.value = "Received"; search.conditions =
[condition, condition2]; var order1 = new TWSearchOrdering(); order1.column = col1;
order1.order = TWSearchOrdering.Orders.Descending; search.orderBy = new Array(order1);
search.organizedBy = TWSearch.OrganizeByTypes.Task; var results = search.execute();
log.info("Result.rows.length = " + results.rows.length); var claimedCount = 0; var
notClaimedCount = 0; for (var i=0; i<results.rows.length; i++) { if (results.rows[i].values[3] ==
null) { notClaimedCount++; } else { claimedCount++; } log.info(results.rows[i].values[0] + ":" +
results.rows[i].values[1] + ":" + results.rows[i].values[2] + ":" + results.rows[i].values[3]); }
log.info("Claimed = " + claimedCount + ", Not Claimed = " + notClaimedCount);
```

|| |**Note:**There is an unconfirmed forum post that claims that more than 14 columns requested to be returned breaks the search.|

If some data you are looking for is not part of the data available in a search, remember that a search may return a process instance or task instance id that can then be used to retrieve a lot more details on that process or task instance that may include the information you are looking for.

See also:

Calling Java through LiveScript

JavaScript in IBPM can invoke Java through the Live Connect technology (see: (ext) LiveConnect). This is not a performance optimal solution but can provide critical functions/features when needed.

Examples of using this include:

Getting the current host-name:

tw.local.hostname = Packages.java.net.InetAddress.getLocalHost().getHostName();

Working with XML in JavaScript

The IBPM supplied framework provides capabilities to work with XML in the JavaScript environment. These capabilities are based on three supplied classes:

The `XMLDocument` is the representation of the DOM model of an XML document. An instance of the `XMLDocument` can be created from its constructor with:

var xmlString = "<A><B>123</B></A>"; var myDocument = **new** tw.object.XMLDocument(xmlString);

An XPath expression interpreter is supplied which can walk into the document and return a list of nodes in the document corresponding to the pattern:

var myNodeList = myDocument.xpath("/A/B");

Once we have a node list, we can determine how many (if any) members are in it with its `length` property. We can retrieve an `XMLElement` object with the `item(index)` method.

var myElement = myNodeList.item(0);

We can also get the value of elements with the `getText()` method:

var myText = myElement.getText();

We can walk the tree using “dot” notation. So if `myVar` is an `XMLElement`, we can code:

`myVar.A.B.getText()` to get the value of `<A><B>`....

See also:

Working with document attachments in JavaScript

Document attachments can be associated with Process Instances. These can be added through Coach Controls or through programmatic additions. The relevant JavaScript components are:

The `addDocument()` function has the following parameters:

`type:String`	This is either `TWDocument.Types.URL` or `TWDocument.Types.File`.
`name:String`	This is the name of the document.
`fileLocation:String`	
`hideInPortal:Boolean`	This is a flag which describes whether or not the document is to be visible in the Process Portal.
`createdBy:TWUser`	
`properties:Map`	

Note: Experimentation seems to show that the `addDocument()` function will **not** work when called from JavaScript locally coded in a BPD activity but seems to work just fine when coded in a General Service.

Here is an example of calling addDocument():

```
var myMap = new tw.object.Map();
 var hide = false;
 var user = tw.system.user;
 tw.system.currentProcessInstance.addDocument(
 TWDocument.Types.File,
 "name",
 "C:\\\\Projects\\\\WLE\\\\Images\\\\ibm.jpg",
 hide,
 user,
 myMap);
```

Interestingly, there is no exposed API to obtain the content of a document or set the content of an attached document from within JavaScript. We can write the content of the document to a local file but that is about it.

See also:

Working with JSON in IBPM

JSON is a string representation of a JavaScript object that can be transmitted over the network. It is primarily used for parameter passing. An Open Source JavaScript implementation is available to both parse and construct JSON strings.

http://www.json.org/js.html

Part of this package is a JavaScript source file called "`json2.js`". This file should be added as a Server managed file to your toolkit or process application. This will provide some new global methods/objects. The core of these is:

JSON.stringify(*JavaScript Object*);

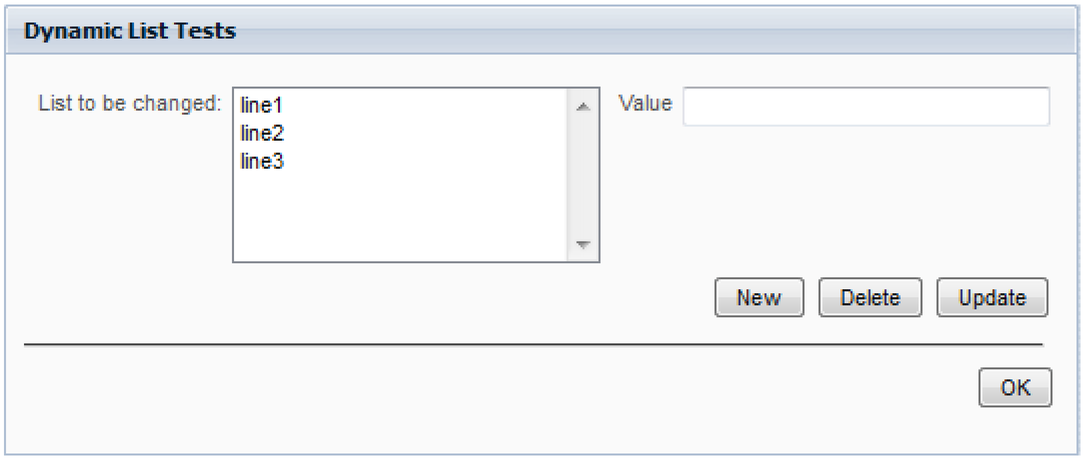
When invoked, this will return a String representation of the JavaScript object parameter. This string is a JSON string.

Unfortunately, even though many data structures within IBPM behaves in many ways like JavaScript objects, under the covers they are not JavaScript objects and don't always work but we can often code work-arounds. Here is an example of taking a List of Complex data structures and building a JSON representation:

```
var newArray = new Array();
 for (var j=0; j\<tw.local.myPurchase.listLength; j++)
 {
 var newObj = new Object();
 for (var property in tw.local.myPurchase[j].propertyNames)
 {
 var name = tw.local.myPurchase[j].propertyNames[property]; newObj[name] =
tw.local.myPurchase[j][name];
 }
 newArray.push(newObj);
 }
 var jsonText = JSON.stringify(newArray);
 log.error("jsonText = " + jsonText);
```

Within a browser/Coach, we can use the Dojo classes to work with JSON. The method `dojo.fromJson(string)` will parse a JSON string to return a JavaScript object while `dojo.toJson(Object)` will return a JSON string given a JavaScript object.

JSON can also be used in conjunction with Coaches. For example, imagine a Coach that looks as follows:



In this story, we have a list presented. We want to be able to perform add, delete and update operations on the list. This is not normally possible with the basic Coach controls as they are usually display only. The solution to this puzzle is to allow the list to be modified at the DOM/HTML level but, when the list is to be sent back to the server, we encode the list as a JSON string and return that. At the server, the JSON is parsed and a corresponding String list is rebuilt at the JavaScript level.

The way that a JSON string is returned from the Coach is to include a text field in the coach but flag it as hidden. It will not appear on the screen but can be set at the HTML level in the browser. This value will be returned back to the server. The following code fragment can then be executed at the server. This expects a JSON encoded array of strings to be returned into the variable `tw.local.returnText`. The result will be a populated `tw.local.myData` array of strings.

```
// Decode the returned text in case it is HTML encoded var jsonText =
Packages.org.apache.commons.lang3.StringEscapeUtils.unescapeHtml4(tw.local.returnText); //
Parse the text as a JSON string var localList =
JSON.parse(Packages.org.apache.commons.lang3.StringEscapeUtils.unescapeHtml4(jsonText)); //
It is expected that the JSON returned is an array of strings // now we want to convert the
JSON string into a BPM array of strings
 tw.local.myData = new tw.object.listOf.String();
 for (var i=0; i\<localList.length; i++)
 {
 tw.local.myData[i] = localList[i];
 }
```

One interesting area to note is the use of the `org.apache.commons.lang3.StringEscapeUtils` package. This is a JAR file added to the server managed files. The JAR comes from the Apache Commons Lang environment. It is used to decode HTML returned data. For example, the data returned:

"hello"

would be returned as

"hello"

The `unescapeHtml4()` method converts it back to the expected format. Although this example illustrates the use of simple data types, more complex structures can also be worked upon.

A really useful tool for working with raw JSON data is the web page found here:

http://www.jsoneditoronline.org/

Into this page, one can post JSON data and have it reformatted/parsed for readability.

JavaScript Fragments

When implementing a Process Application, it is likely that JavaScript coding will be required in some small or large part. This section provides a series of commonly occurring fragments of JavaScript code and recipes that can be reused in those solutions.

Starting a new process

A new instance of a process can be started by using the `tw.system.startProcessByName()` function. This function takes two parameters. The first is the name of the Process to be started. The second is a Map containing the input parameters for the process. An instance `TWProcessInstance` is returned. For example, this will start a new instance of a process with data:

```
var inputs = new tw.object.Map();

inputs.put("parm1", "parm1 value");inputs.put("parm2", "parm2 value");

tw.system.startProcessByName("StartProcess2", inputs);
```

Note that the process will start executing concurrently.

While this technique works great for starting a new process, that process has to exist in the same Process App as the caller. If we want to start a new process from a separate Process App, things are a little trickier and we have to know the identity of the Process App that contains the Process template we wish to start.

Getting the current process instance

The current process instance can be retrieved through the variable called:

tw.system.currentProcessInstance

The data type of that object is `TWProcessInstance`.

Getting the current userid

The current userid is available as a `TWUser` object from:

`tw.system.user`

Realize that at the BPD level, JavaScript based BPD activities do not run as a particular user so `tw.system.user` will return '`undefined`'.

Starting an external application

If we wish to start an application that is external to IBPM such as a batch file or shell script, we can perform that task by using the JavaScript LiveScript mechanism. The following fragment will execute a batch file on a Windows system and can be placed in a JavaScript fragment:

```
var runtime = java.lang.Runtime.getRuntime();
runtime.exec("C:\\\\RunMe.bat");
```

Returning the owner of a task

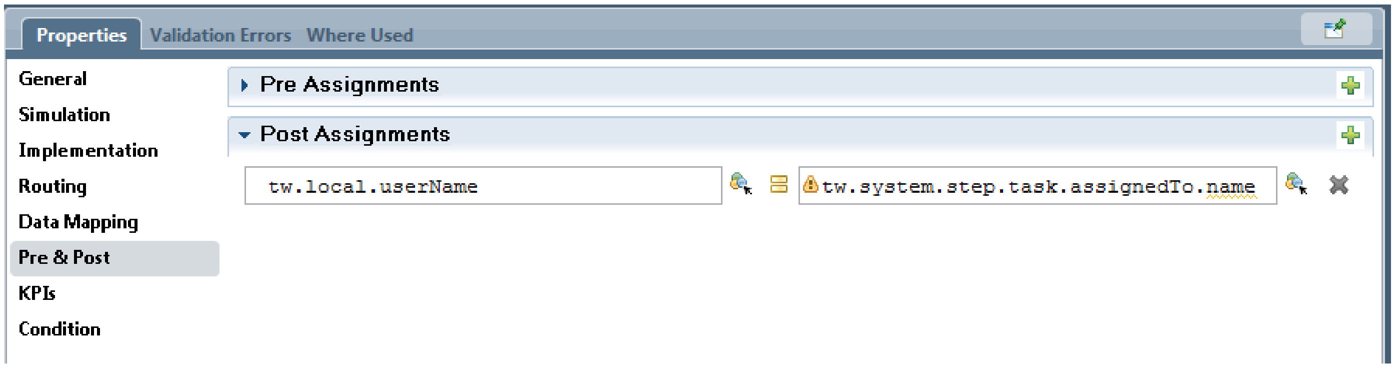
When a task is completed, we may wish to know the identity of the user that completed the task. One way to achieve this is to define an output parameter for the Human Service that is the name of the user that completed the task. Within the Human Service, prior to its termination, this identity can be determined with the following code:

var user = tw.system.currentTask.assignedTo;

The data type returned is an instance of `TWUser`.

The property called `name` of this data type can be used to determine the userid of the user that claimed the task.

As an alternative to returning the userid of the task from within the Human Service, a post-assignment can be performed on the Human Service. In the Post Assignment, the variable `tw.system.step.task.assignedTo` is again a `TWUser` that holds the `TWUser` that owned the Human Task.



tw.local.userName

tw.system.step.task.assignedTo.name

Note: As of 2014-01, there appears to be a bug in the entry assist for this variable. When we ask for entry assist for `tw.system.step.task.assignedTo` , the PD tool seems to want to think of this as a "list" when it is in fact a straight `TWUser` .

Extracting a managed file

If a Process App or a Toolkit contains a managed file, on occasion we may want to extract that from the run-time to see what it contains. One way to achieve this is to execute the following JavaScript:

```
var managedFile = tw.system.model.findManagedFileByPath("integration.jar",
TWManagedFile.Types.Server);log.error("Got the managed file!: " + managedFile);
managedFile.writeDataToFile("C:\\\\integration.jar");
```

This will retrieve the named managed file and write its contents of a local file.

Generating a Random Number or string

There are times, especially during development and testing, where we might want a random number in our solution. This could perhaps be used to determine which path to take in a process or some similar notion. An easy solution is to leverage the `java.util.Random` class.

```
var rand = new java.util.Random();
var value = rand.nextInt(10);
```

This will generate a random integer between 0 and 9 (inclusive).

If we want a random string of a certain length, there are excellent JavaScript fragments on the Internet to achieve that.

Revisión #4

Creado el hace 1 año por Admin

Actualizado el hace 1 año por Neil Kolban