

---

## **AstroGrid Job Management v.itn07**

**User Guide**



# Table of Contents

---

<b>1 Introduction &amp; Overview</b>	
1.1 System Architecture .....	1
1.2 FAQ .....	5
1.3 Glossary .....	6
<b>2 Workflow Documents</b>	
2.1 Activities .....	7
2.2 Control Logic, Variables & Scripts .....	12
2.3 JEScript Objects .....	19
2.4 Execution Results .....	85
2.5 Document Schema .....	87
<b>3 Examples</b>	
3.1 Simple Examples .....	88
3.2 Scripted Workflows .....	89
3.3 JEScript Recipes .....	90



## 1.1 System Architecture

---

### Introduction

A workflow aims to accomplish a complex piece of work. In Astrogrid terms this would be an astronomical investigation. If the work could be done in one simple step (eg: by executing a single program from the command line), then it would hardly be a workflow. A prime dimension of a workflow is therefore that the objective will take more than one step to accomplish - it has a degree of complexity.

A simple example of a workflow might consist of the following steps:

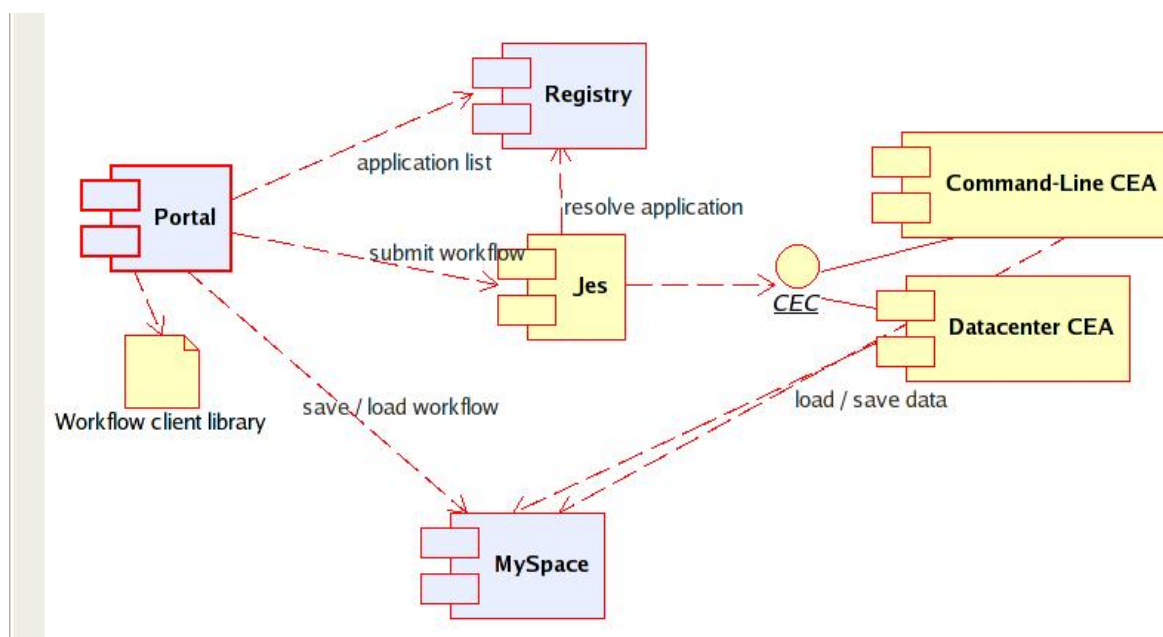
1. Make a query against against one catalog (say a cone search of one area of the sky).
2. Make a similar query against another catalog.
3. Merge the contents.
4. Analyse the resulting data.

Each step is usually the invocation of a separate computer program (a tool), each tool requiring its own inputs and outputs. Some inputs to a step will be astronomical data, and some inputs will be control information for the selected tool. Outputs will normally be processed data destined as a final result, or as intermediate data to be used as inputs for subsequent steps within the workflow. For Astrogrid, the final result of executing a workflow is usually a file held in MySpace. From that viewpoint, Astrogrid workflow is itself an intermediate tool, since we would envisage the final results file being subsequently loaded into another tool, for example, for visualization, or even for input into a different workflow.

Astrogrid tries to deal with the complexity of workflows. But it is not an easy thing to eliminate. The following tries to give the salient points.

### Format and Structure

We hold workflow designs as XML documents. This is a particularly verbose way of describing an individual workflow, but is convenient for its rigour. Examples are shown [here](#). There is no necessity to interact directly with a workflow xml document and therefore there is no necessity to be intimately acquainted with it; the portal design tool hides most of the intricacies. However, some exposure to it is probably useful. Workflow structure is described by its XML schema: every workflow document has to obey the rules embodied in the schema. The details of the workflow schema are described [here](#), but come back to this later.



## Persistence Mechanism

Workflow designs are held as files (as XML documents) and stored in MySpace for Astrogrid purposes. They can be copied or shared. At the end of the day they are just files.

## Design Time and Execution Time

A workflow must be designed by an astronomer with a particular goal in mind. The process of design may be anything from the straightforward to the immensely complex. Even a relatively simple workflow consisting of a small number of steps may involve an astronomer in an iterative process of design, running the workflow against data, examining the results for feedback, then refining the design. A good workflow may therefore be something of a capital investment, and worth sharing with others.

There are two aspects worth noting.

1. Workflows are involved in two processes: a design time process and an execution time process. A single workflow (ie, a design) may be executed many times. Usually it is obvious from the context whether a workflow is meant as a design or as a record of a specific execution of that design. Just be aware of the difference. We try to be clear by using a separate term (ie: 'job') for an executed workflow.
2. Any workflow will refer to data outside of itself (files, catalogs). If you copy, amend or share a workflow, some of these external references may need to be taken into account.

## Metadata

In designing a workflow, we need specialized information regarding each of the steps we design. We tend to refer to data that itself describes astronomical data, or data that describes the working of an astronomical tool, by the term metadata. Most often you hear the terms catalogue metadata and tools

metadata. A workflow design contains metadata or references to metadata within it.

We embody metadata within a workflow in various forms:

- As a query to be submitted against an astronomical archive
- As a reference to a file which itself is to be used as control data (eg, a configuration file for a given tool)
- As settings embedded directly in a workflow document, where the setting is used to control the execution of a tool (eg, an integer value).

But even the name of a tool is derived from access to metadata, so there is no hard and fast line. This control material must be collected together in the design process. Metadata is stored in the Astrogrid registry.

## Job and Job Submission

When a workflow begins execution, we treat it as a job. A workflow becomes a job by being submitted to the Job Execution System. Each submission represents a separate job. If you submit one workflow fifty times, you have fifty jobs, and one workflow. We keep a separate record for each job, a fact which is reflected in the workflow schema. In effect, a job is a specialized workflow document, which contains not only the design (as it was when it was submitted) but run-time information about a specific execution down to the individual step level.

## Job Execution System

The Job Execution System is the part of Astrogrid that controls the execution of a workflow. Otherwise known as JES, it is Astrogrid's workflow engine. JES is an engine that can manage jobs consisting of multiple steps, where individual steps can be run on different computers across a network. JES will attempt to run steps in a controlled fashion (eg: in parallel or sequentially depending upon the workflow definition). Because steps can execute at different locations, both control flow and data flow have to be managed. A partner component, known as the Common Execution Controller (CEC), manages step execution and any data flow required.

All steps in Astrogrid workflow execute asynchronously. That means that a step dispatched by JES is done so on a non-blocking basis.

## CEA and CEC

CEA stands for Common Execution Architecture. It is our standard for formalizing the web-interface provided by all tools, no matter whether they are implemented as Java code, commandline applications, or another SOAP / HTTP-GET based web service. The set of CEA applications may be split into *datacenters* and *processing tools*. Datacenters support complex queries against astronomical archives, while processing tools consume files of data and reduce them in some way. Although these are two very different animals, both can be thought of as tools.

CEC implementations inform JES when a step has completed. This is because steps are dispatched on an asynchronous basis: there is no waiting for a step to finish once it has been dispatched.

## A Unit of Work

If you want to consider things as units of work, the smallest unit of work undertaken in a workflow is embodied in a single step. So a job will contain all the units of work successfully completed by the steps that got executed by JES. However, usually the appearance of a final result's file (eg, a specific VOTable), or a set of such files, within MySpace will be indicative. At the end of the job, or even at strategic points within it, an email can be sent. Whether a job is a formal success or failure can be programmed into the workflow design and interpreted by JES. Certainly from this point of view you can definitely tell when a job has failed. It will tell whether things have gone wrong, but not necessarily whether the results produced in a successful run are what were expected. (At present we also have some problems with steps that never return.)

## 1.2 FAQ

---

### Frequently Asked Questions

#### General

1. [A sample question?](#)

#### General

General

A sample question?

some markup goes here

```
some source code
```

some markup goes here

## 1.3 Glossary

---

### Glossary

#### Workflow

There are a few nuances surrounding the term workflow.

A workflow is a set of pieces of work coordinated to achieve a target of some astronomical investigation. Each piece of work is considered a step in the workflow, and will be undertaken by running a computer program.

A concise and somewhat abstract definition such as the above leaves a lot to be desired and inevitably sounds somewhat pompous. However, in concrete terms a workflow is described in XML and is stored as an XML document, ie: as a file on a file system. When we refer to a workflow, the reference is often to such an XML document. A workflow XML document has an XML schema to describe it, and so the structure of a workflow has an abstract design that all workflows should adhere to.

A workflow consists of a hierarchical structure of parts (steps, tools, flows, sequences). We currently use Castor to generate the Java classes from the XML schema. And a particular instance of a workflow XML file may be deserialized or instantiated into a set of Java objects. So, when we refer to workflow, we can also be referring to the Java class named Workflow, or even more probably to a particular object of type Workflow.

To further complicate matters, in the round a workflow usually refers to the design, whereas a job is the execution-time equivalent, somewhat in the way that a blueprint of a building is not the building. In Astrogrid parlance, a job is a workflow that has been submitted for execution.

Sorry. Anyone familiar with software development will recognize this confusing level of meaning. It's part of the job (no pun intended). Most meanings are clear from the context.

## 2.1 Activities

---

### Introduction

This document describes the basic structures of the workflow document format.

### Definition

Workflows are expressed as XML documents, conforming to the schema  
<http://www.astrogrid.org/schema/AGWorkflow/v1> (and included schema)

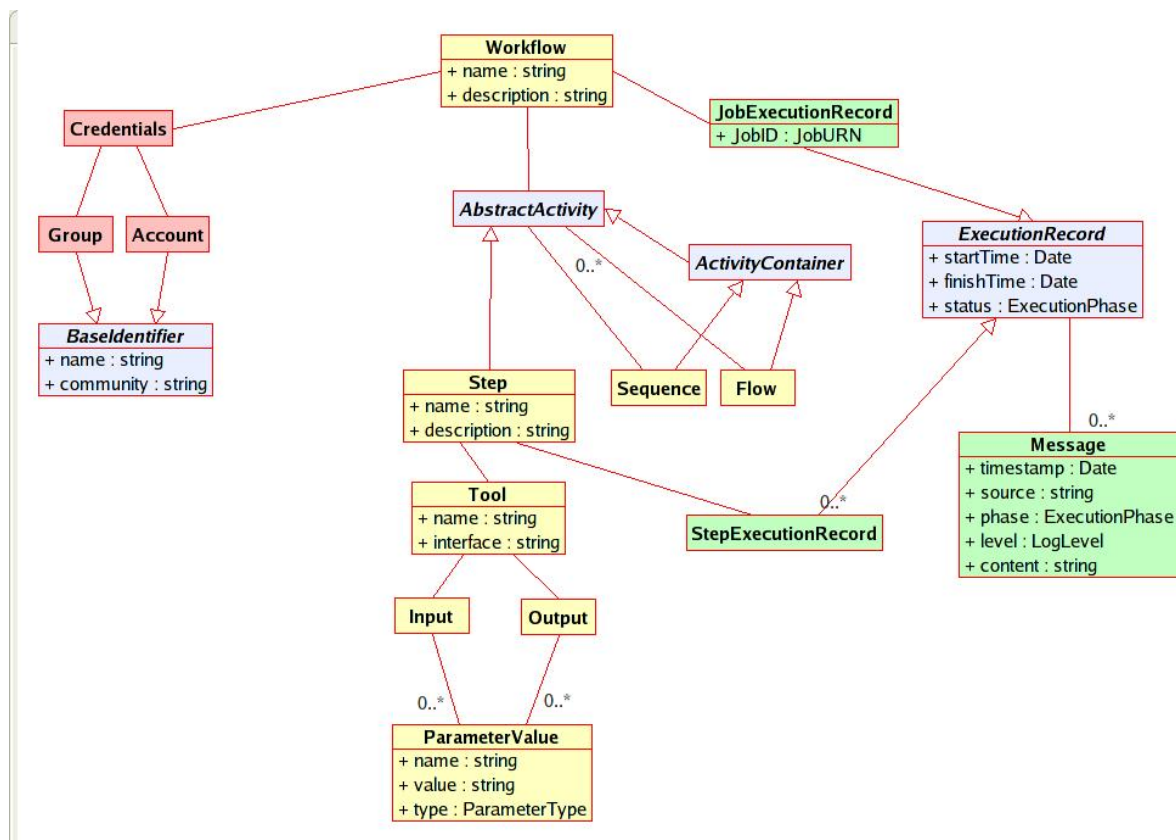
formatted html

<http://www.astrogrid.org/maven/docs/HEAD/astrogrid-workflow-objects/schema/Workflow.html>

schema definitions

<http://www.astrogrid.org/viewcvs/astrogrid/workflow-objects/schema/>

### Schematic View



## Top Level

The root of the workflow document is a `<workflow>` element. It has a required `name` attribute, and an optional `description` element.

Most importantly, a workflow contains a `<sequence>` element, which contains the list of activities to perform in this workflow.

During execution of a workflow, progress messages are added to the workflow document. These are stored in the `<job-execution-record>` child element of workflow. This is described in more detail in a [later section](#).

There's also a `Credentials` element tree, which describes the identity and authentication of the account to run the workflow under. We'll ignore this at the moment.

Finally, there's a `id` attribute that occurs on many of the workflow constructs. This is used within the implementation, and should not be set when writing a workflow.

## Example

In the examples in this document, I've omitted namespace declarations and prefixes for clarity. You'll need these (sadly) if writing real workflow documents by hand..



```
<?xml version="1.0" ?>
<workflow name="a workflow">
  <description>description of the workflow</description>
  <sequence>
    <!-- omitted-->
  </sequence>
  <Credentials>
    <!-- omitted -->
  </Credentials>
  <job-execution-record>
    <!-- omitted -->
  </job-execution-record>
</workflow>
```

## Sequencing

### Sequence

The `<sequence>` element composes a set of activities into a linear structure - they will be executed sequentially in order.

### Flow

The `<flow>` element composes a set of activities into a parallel structure - the activities in the flow may be executed in any order, or even concurrently

### Examples

```
<!-- execute some activities sequentially -->
<sequence>
  <step name="a">
    <!-- omitted -->
  </step>
  <step name="b">
    <!-- omitted -->
  </step>
</sequence>
```

```
<!-- execute same activites concurrently -->
<flow>
  <step name="a">
    <!-- omitted -->
  </step>
  <step name="b">
    <!-- omitted -->
  </step>
</flow>
```

## Step

The `<step>` element is an activity that performs a call to a CEA application. (*link to cea*). This element has a required `name` attribute, and an optional `description` child element.

When a step is executed, the progress of the execution is recorded as a `<step-execution-record>` child element. This is covered in a [later section](#)

## Tool

The `step` element contains a `<tool>` element, which specifies: the name of the CEA application to execute; which interface of the application to call; and the input and output parameter values to the tool call.

## Parameter

Input and Output Parameters to a call are defined by `<parameter>` elements. The `name` attribute defines the name of the parameter, while the content of the `<value>` child element defines the value for this parameter. The parameter value may contain [script expressions](#) delimited by `{ . . }` - these are evaluated into strings before the parameters are passed to CEA.

Finally the `indirect` attribute alters how the value of the parameter is interpreted by CEA. If set to `true`, the parameter value is expected to be a URI that points to a resource that contains the actual value for this parameter. If set to `false` (the default), then the parameter value is expected inline.

For output parameters, if the `indirect` attribute is set to `true` then the value of the parameter specifies the location to store this result. However, if the `indirect` attribute is set to `false`, then this result value will be returned direct to the JES server

## Accessing Results of the Tool Execution

Sometimes the results of a CEA Tool execution are required further on in the workflow - for example, so that they can be passed to another CEA Tool, or to select branches of the workflow to execute.

The `step` element has an optional attribute `result-var`. If set, this attribute defines the name of a workflow variable in which the results computed by the CEA Application will be stored after the application finishes executing.

Once execution completes, the variable named will contain an instance of `java.util.Map`. The map will have entries for all output parameters that were configured to return `direct`. The keys in the map are the output parameter names, while the values are the results returned from the CEA application.

## Example

```
<step name="exampleStep" result-var="exampleStepResults">
  <description>an example</description>
```

```
<tool name="anAuthority/aCeaApplication" interface="simpleInterface">
  <input>
    <!-- constant parameter value-->
    <parameter name="RA"><value>21</value></parameter>
    <!-- script expression referring to previously defined workflow variable
-->
    <parameter name="DEC"><value>${dec}</value></parameter>
    <!-- indirect parameter value -->
    <parameter name="Radius"
indirect="true"><value>ivo://aCommunity/user#user/root/parameters.txt</value></parameter>
  </input>
  <output>
    <!-- write results to handy ftp server, uses script expression to generate
timestamped output filename -->
    <parameter name="results"
indirect="true"><value>ftp://aServer/myResults/results-${new
java.util.Date()}.votable</value></parameter>
  </output>
</tool>
</step>
```

## 2.2 Control Logic, Variables & Scripts

---

### Introduction

The previous section shows how to write basic linear workflows. This chapter describes how to construct workflows containing loops and branches. Looping and conditional control constructs need some way of testing and altering values - so we start by introducing *workflow variables*

*This document is still a bit tangled at present - you may need to read the whole thing once before it starts to make sense*

### Workflow Variables

A workflow may contain *workflow variables* - named variables that contain simple java objects and structures. Variables may be accessed and altered from `<script>` elements within the workflow document, and may also be referenced from script expressions in other workflow activities

### Set

The `<set>` activity defines a new workflow variable, or updates the value of an existing variable. It has two attributes: `var` - the name of the workflow variable (required); `value` - the value to set this variable to (optional).

If no value is provided, the variable is created and set to `null`

The `value` attribute may be a straight literal string, or an script expression

### Unset

The `<unset>` activity deletes a workflow variable. It has one required attribute - `var` - the name of the workflow variable to delete.

If a workflow variable is referenced after it has been deleted, an exception is thrown..

### Scope

`<scope>` is a container for activities that introduces a new nested scope for workflow variables. Any workflow variables defined within a nested scope are not visible outside that scope. Any attempts to reference such variables throws an exception

### Script Expressions

The value of certain attributes and elements may contain references to workflow variables and other scripting language expressions. These **script expressions** are delimited by `${ . . }`. Script expressions are interpreted as follows

- If the entire content of an attribute / element is a script expression (with no further characters or whitespace), then the java object that is the result of evaluating the expression is returned.
- Otherwise, if the attribute / element contains more than one script expression, or an expression plus other characters, each of the expressions is evaluated in turn, the results converted to Strings, and then concatenated together.

## Examples

```
<!-- in the variable 'a' store the value 'hello world' (java.lang.String) -->
<set var="a" value="hello world" />

<!-- in the variable 'b', store the value '1' (java.lang.String) -->
<set var="b" value="1" />

<!-- in the variable 'c', store the value 1 (java.lang.Integer) -->
<set var="c" value="${1}" />

<!-- in the variable 'd' store the value '1 + 1' (java.lang.String) -->
<set var="d" value="${c} + 1" />

<!-- in the variable 'e' store the value 2 (java.lang.Integer) -->
<set var="e" value="${c + 1}" />

<!-- in the variable 'now' store the current date (java.lang.Date) -->
<set var="now" value="${new java.util.Date()}" />

<!-- in the variable 'nowString' store the current date (java.lang.String) - note
trailing space -->
<set var="nowString" value="${new java.util.Date()} " />

<!-- in the variable 'f' store the value '1 - hello world and goodbye.' -->
<set var="f" value="${c} - ${a} and goodbye. " />

<!-- in the variable 'u' store the value 'http://www.astrogrid.org' (java.net.URL)
-->
<set var="u" value="${new java.net.URL('http://www.astrogrid.org')}" />

<!-- in the variable 'scheme' store the value 'http' (java.lang.String) -->
<set var="scheme" value="${u.getScheme()}" />

<!-- in the variable 'scheme' store the value 'http' using concise bean-property
access -->
<set var="scheme" value="${u.scheme}" />
```

## Script

The `<script>` element is an activity that executes some inline scripting code. This is a very versatile activity - some potential uses are

- post-process the results received from a CEA tool (e.g. extracting fields from a VOTABLE)
- manipulating contents of workflow variables - computing values that can't easily be expressed as a simple script expression.
- dynamically adding parameters to subsequent calls to CEA tools
- interrogating astrogrid registries
- moving / copying / deleting files in VOspace
- initiating / tracking progress / aborting other workflow jobs.
- Administrative functions - such as creating user accounts.

The `script` element contains an optional `description` element and a mandatory body element that contains the script text. When it is executed a `step-execution-record` element will be added which records the execution of the script.

A script may reference workflow variables, reading and storing data in them. The changes to the workflow variables are visible further on in the workflow. A script may also define local variables, functions, etc. However, these are only available to the script itself - they are not visible to subsequent scripts or script expressions. Hence any result that is to be accessed later should be stored in a previously-defined workflow variable.

## Scripting Language

The scripting language used within script expressions and `<script>` elements is Groovy - <http://groovy.codehaus.org/> . Groovy describes itself as follows

*Groovy is a new agile dynamic language for the JVM combining lots of great features from languages like Python, Ruby and Smalltalk and making them available to the Java developers using a Java-like syntax.*

*Groovy is designed to help you get things done on the Java platform in a quicker, more concise and fun way - bringing the power of Python and Ruby inside the Java platform.*

Groovy is a superset of Java - Java expressions and statements are valid in Groovy scripts. The java-subset is sufficient for most purposes and should be manageable for anyone who's had experience with Java / C / C++ / JavaScript - the notation is the same.

However, Groovy does provide further language features and sugar, which make it more concise and easy to use - dynamic typing, [native syntax for collections](#) , [closures and internal iterators](#) , [regular expressions](#) , [support for generating xml](#) , [support for consuming xml](#)

There's also a handy reference card to print out -

<http://docs.codehaus.org/download/attachments/2715/groovy-reference-card.pdf>

## Examples

Print out a message (which gets captured into the execution record).

```
<script>
  <body>
    print("hello world");
```

```

</body>
</script>

```

Extract a list of urls from a votable returned by a previous step, store in workflow variable for later use. This example uses methods native to groovy - in the examples chapter we show how to do the same thing more concisely using the STIL library.

```

<step result-var="results">
  <!-- omitted for clarity -->
</step>
<set var="urlList" /> <!-- declare a variable, but don't initialize it-->
<script>
  <body>
    if (results.size() != 1) {
      jes.error("previous step didn't produce expected number of results");
    } else {
      votable = results.get('votable'); // access result of previous step
      parser = new XmlParser(); //create new parser
      nodes = parser.parseText(votable); //parse votable into node tree
      urlList = nodes.depthFirst().findAll{it.name() ==
'STREAM'}.collect{it.value()}.flatten(); // filter node tree on 'STREAM', project
value
      print(urlList); // show what we've found
    }
  </body>
</script>

```

## Conditional

The `<if>` element allows conditional execution. It has a required attribute `test`, which must contain a script expression that evaluates to a boolean.

The `if` element may have either or both a `then` and `else` child elements. Each contains an activity (or sequence of activities) that will be executed depending on the value of the `test` attribute

### Example

```

<set var="x" value="{1}" />
<if test="{x > 0}">
  <then>
    <sequence>
      <!-- some activities to do here -->
    </sequence>
  </then>
  <else>
    <script>
      <body>
        print('test was false');
      </body>
    </script>
  </else>
</if>

```

```

    </script>
  </else>
</if>

```

## While Loop

The `<while>` element expresses a while loop. It has a required attribute `test` which must contain a script expression that evaluates to a boolean.

Its body is an activity (or sequence / flow of activities) that will be executed for every time that the test evaluates to `true`.

### Example

Repeatedly execute a step, until it returns at least one result value.

```

<while test="\${results == null || results.size() < 1}">
  <step result-var="results">
    <!-- omitted -->
  </step>
</while>

```

## For Loop

The `<for>` element expresses a for loop. The structure of the for loop is similar to the `for` in Python (or `for-each` in Javascript) - it iterates over a sequence, rather than using an arithmetic expression like in Java / C / C++.

The `for` element has two required attributes: `items` which must evaluate to a list of items to iterate over; and `var` which provides the name of the loop variable to assign each element of the list to. The body of the `for` element is an activity (or sequence / flow of activities) that will be executed for each item on the list.

Groovy provides native syntactic support for quickly defining numeric sequences -

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

### Examples

Count up to 10.

```

<for var="x" items="\${1...10}"> <!-- start ... finish is groovy syntax for numeric
ranges -->
  <script>
  <body>
    print(x);
  </body>

```

```

</script>
</for>

```

Call a CEA tool for each item in a list of urls (as was created in earlier example)

```

<for var="u" items="{urlList}">
  <sequence>
    <script>
      <body>
        jes.info("calling tool for {u}")
      </body>
    </script>
    <step name="something">
      <tool name="aTool" interface="simple">
        <input>
          <parameter name="input" indirect="true">
            <value>{x}</value><!-- x contains the url of the resouce which
contains this parameter value. -->
          </parameter>
        </input>
        <output>
          <parameter name="result" indirect="true">
            <value>vospace:/myresults/{x.tokenize('/')}.pop()}-resuts.dat</value>
            <!-- use the last part of the input url as part of the output
filename -->
          </parameter>
        </output>
      </tool>
    </step>
  </sequence>
</for>

```

## Parallel For Loop

The `<parfor>` element expresses a **parallel for** loop. It has the same structure as the `for` loop, but executes it's loop body simultaneously for each item in the `values` list.

This construct is useful for starting many CEA application exections in parallel. For example, the previous example could be altered to process each url in the list simulataneously by simply replacing the `for` element with a `parfor` element.

## Error Handling

When an error occurs during the execution of an activity, the normal flow of control is interrupted. The error is recorded, and then propagates upwards. (as with exceptions in other languages). If it reaches the root `workflow` element, then execution of the workflow halts.

The workflow schema defines a `try` element that can be used to wrap activities and intercept errors. There is also a `catch` element, which can be used to define activities to execute only when an error occurs.

**NB:** try and catch are not implemented at the moment (Iteration 7)

## 2.3 JEScript Objects

---

### What's available?

From groovy scripts it is possible to import any class available on the classpath, and then instantiate and use it in the normal manner.

In addition, the JES Server provides an environment for scripts to execute that is configured with some pre-initialized **system objects** - we call scripts for this environment JEScripts. The system objects are preconfigured objects that are always available under fixed names. They can be used to gain information and interact with the current state of the workflow, the Jes Server, and other Astrogrid components.

*link to list of jars bundled with JES - so user can see what's available*

### System Objects

The following objects are available to JEScripts and script expressions:

`astrogrid`

An object that represents the entire Astrogrid system. Implemented by class [org.astrogrid.scripting.Toolbox](#) . Provides

- configured delegates to access other astrogrid services (vospace, registry, cea, jes)
- libraries for constructing workflows
- libraries for parsing and constructing tables
- libraries for input / output
- access to system config

`jes`

An object that represents the JES server that is executing the current workflow. Implemented by class [org.astrogrid.jes.jobscheduler.impl.groovy.JesInterface](#) . Provides

- logging
- access to the document model of the current workflow
- execution control

`account`

object that represents the identity of the user under which this workflow is being executed. An instance of [org.astrogrid.community.beans.v1.Account](#)

`user`

another (deprecated) representation of the current user. An instance of

[org.astrogrid.community.User](#)

`userIvorn`

Ivorn representing the current user. An instance of [org.astrogrid.store.Ivorn](#)

`homeIvorn`

Ivorn pointing to the homespace of the current user. An instance of [org.astrogrid.store.Ivorn](#)

## Toolbox

<a href="#">org.astrogrid.scripting.IOHelper</a>	help with input / output
<a href="#">org.astrogrid.scripting.ObjectBuilder</a>	Class to help construction of various kinds of object
<a href="#">org.astrogrid.scripting.table.MutableScriptStarTable</a>	Extension of the standard starlink read-write table that supports the extra scripting methods.
<a href="#">org.astrogrid.scripting.table.ScriptStarTable</a>	A Scriptable Star Table - extends basic StarLink StarTable.
<a href="#">org.astrogrid.scripting.table.StarTableBuilder</a>	factory for StarTables
<a href="#">org.astrogrid.scripting.TableHelper</a>	Helper object for working with STIL tables See Userguide - <a href="http://www.star.bristol.ac.uk/~mbt/stil/sun252.html">http://www.star.bristol.ac.uk/~mbt/stil/sun252.html</a> See JavaDoc - <a href="http://www.star.bristol.ac.uk/~mbt/stil/javadocs/uk/ac/starlink/table/package-summary.html">http://www.star.bristol.ac.uk/~mbt/stil/javadocs/uk/ac/starlink/table/package-summary.html</a>
<a href="#">org.astrogrid.scripting.Toolbox</a>	root scripting object
<a href="#">org.astrogrid.scripting.XMLHelper</a>	helper methods for working with xml.

The main class for interacting with the astrogrid is [Toolbox](#) . An instance of this class, named `astrogrid` is available in the JEScript environment. Other subsidiary helper objects and delegates are accessible by calling the methods on `Toolbox`

## IOHelper

helper methods for working with streams and external values more..

pipe contents of in to out

```
void bufferedPipe(java.io.Reader in, java.io.Writer out)
```

pipe contents of in to out

```
java.lang.String dateStamp()
```

generate a date-stamp in format 'DD-MM-YY'

```
java.lang.String dateTimeStamp()
```

generate a date/time stamp in format 'DD-MM-YY-HHMMSS'

```
java.lang.String getContents(java.io.InputStream is)
```

read contents of a stream

```
java.lang.String getContents(java.io.Reader r)
```

read contents of a stream

```
java.lang.String getContents( org.astrogrid.applications.parameter.protocol.ExternalValue ext)
```

read contents of an external value

```
org.astrogrid.applications.parameter.protocol.ExternalValue getExternalValue(java.lang.String uri)
```

create an external value that points to a vospace uri

```
org.astrogrid.applications.parameter.protocol.ExternalValue getExternalValue(java.net.URI uri)
```

create an external value that points to a vospace uri

```
org.astrogrid.applications.parameter.protocol.ExternalValue getExternalValue(java.net.URL uri)
```

create an external value that points to a vosapce uri

```
org.astrogrid.applications.parameter.protocol.ProtocolLibrary getProtocolLibrary()
```

access library object that 'knows' about a variety of IO protocols, and can construct {@link ExternalValue} objects to read / write resources via these protocols.

```
void pipe(java.io.InputStream in, java.io.OutputStream out)
```

pipe contents of in to out

```
void pipe(java.io.InputStream in, org.astrogrid.applications.parameter.protocol.ExternalValue out)
```

pipe contents of in to out

```
void pipe(java.io.Reader in, java.io.Writer out)
```

pipe contents of in to out

```
void pipe( org.astrogrid.applications.parameter.protocol.ExternalValue in, java.io.OutputStream out)
```

pipe contents of in to out

```
void pipe( org.astrogrid.applications.parameter.protocol.ExternalValue in,
org.astrogrid.applications.parameter.protocol.ExternalValue out)
```

pipe contents of in to out

```
java.io.Reader readFromString( java.lang.String content)
```

returns a reader of the contents of a string

```
java.io.InputStream streamFromString( java.lang.String content)
```

returns a stream of the contents of a string

```
java.lang.String timeStamp()
```

generate a time-stamp as a long number

---

## ObjectBuilder

Class to help construction of various kinds of object more..

helper method to create an account object

### *Params*

- username name of the user
- community community the user is registered with

```
org.astrogrid.community.beans.v1.Credentials newCredentials( org.astrogrid.community.beans.v1.Account
acc, org.astrogrid.community.beans.v1.Group grp)
```

create a credentials object from an account and a group

```
org.astrogrid.community.beans.v1.Group newGroup( java.lang.String groupName, java.lang.String community)
```

create a group object \*

### *Params*

- groupName name of the group
- community community the group belongs to

```
org.astrogrid.store.Ivorn newIvorn( java.lang.String ivorn)
```

construct an arbitrary ivorn object

*Throws*

- URISyntaxException

---

```
org.astrogrid.store.Ivorn newIvorn(java.lang.String path, java.lang.String fragment)
```

---

construct an arbitrary ivorn object

*Params*

- path the ivorn
- fragment a fragment - i.e. the bit after #

*Returns* an ivorn of form ivo://path#fragment

---

```
org.astrogrid.store.Ivorn
newIvorn(java.lang.String authority, java.lang.String key, java.lang.String fragment)
```

---

construct an arbitrary ivorn object

*Params*

- authority the authority of the ivorn
- key key into the authority
- 

*Returns* an ivorn of form ivo://authority/key#fragment

---

```
org.astrogrid.store.Ivorn newIvorn( org.astrogrid.store.Ivorn root, java.lang.String file)
```

---



---

```
org.astrogrid.store.Ivorn newLocalUserIvorn(java.lang.String account)
```

---

create a user ivorn, from account name and locally-configured community

*Params*

- account the username

*Returns* a user ivorn, where the community name is provided by local configuration

*Throws*

- CommunityServiceException is the community service can't be accessed
- CommunityIdentifierException

---

```
org.astrogrid.community.User
newUser(java.lang.String userid, java.lang.String community, java.lang.String group, java.lang.String token)
```

---

create a user

*Params*

- `userid` username
- `community` community the user is registered with
- `group` group the user belongs to
- `token` not used

---

```
org.astrogrid.store.Ivorn newUserIvorn(java.lang.String community, java.lang.String account)
```

---

create a user ivorn, from community and account name

*Params*

- `community` the name of the community the account is registered with
- `account` the username

*Throws*

- `CommunityIdentifierException` if the community is not known.

---

```
org.astrogrid.store.Ivorn newUserIvorn( org.astrogrid.community.User u)
```

---

convert a user object into an ivorn that represents that user

*Params*

- `u` a user object

*Returns* the equivalent user ivorn.

*Throws*

- `CommunityIdentifierException` if the community is not known

---

```
org.astrogrid.community.User user()
```

---

create the anonymous user

---

## TableHelper

Helper object for working with STIL tables

See Userguide - <http://www.star.bristol.ac.uk/~mbt/stil/sun252.html>

See JavaDoc -

<http://www.star.bristol.ac.uk/~mbt/stil/javadocs/uk/ac/starlink/table/package-summary.html>

more..

access the table builder

---

```
uk.ac.starlink.table.StarTableOutput getOutput()
```

---

access the STIL library object for writing out tables - see

<http://www.star.bristol.ac.uk/~mbt/stil/javadocs/uk/ac/starlink/table/StarTableOutput.html>

---

```
uk.ac.starlink.table.ColumnInfo newColumnInfo(java.lang.String name)
```

---

create a new column info object - see

<http://www.star.bristol.ac.uk/~mbt/stil/javadocs/uk/ac/starlink/table/ColumnInfo.html>

---

```
uk.ac.starlink.table.ColumnInfo newColumnInfo(java.lang.String name, java.lang.Class contentType, java.lang.String description)
```

---

create a new column info object - see

<http://www.star.bristol.ac.uk/~mbt/stil/javadocs/uk/ac/starlink/table/ColumnInfo.html>

---

```
org.astrogrid.scripting.table.MutableScriptStarTable
newMutableTable(uk.ac.starlink.table.ColumnInfo[] info)
```

---

create a new empty mutable table

---

```
org.astrogrid.scripting.table.MutableScriptStarTable
newMutableTableFromTemplate(uk.ac.starlink.table.StarTable s)
```

---

create a new empty mutable table, with the same structure as a template table

---

```
java.io.InputStream toInputStream(uk.ac.starlink.table.StarTable table, java.lang.String format)
```

---

access contents of a table as a stream

---

```
java.lang.String toString(uk.ac.starlink.table.StarTable table, java.lang.String format)
```

---

access contents of table as a string

*Throws*

- IOException

---

```
void writeTable( org.astrogrid.applications.parameter.protocol.ExternalValue
val, uk.ac.starlink.table.StarTable table, java.lang.String format)
```

---

write star table to an external location

*Throws*

- InaccessibleExternalValueException
- IOException

## Toolbox

this is the *main* scripting object more..

create a CEA client connected to the specified service endpoint

---

```
org.astrogrid.registry.client.admin.RegistryAdminService createRegistryAdminClient()
```

---

client to administer the registry

---

```
org.astrogrid.registry.client.query.RegistryService createRegistryClient()
```

---

client to query the registry

---

```
org.astrogrid.store.tree.TreeClient createTreeClient( org.astrogrid.store.Ivorn
acc, java.lang.String password)
```

---

create a client to work with the vospace tree-model

### Params

- `acc` the ivorn of the account to connect to
- `password` the password for this account

*Returns* a logged-in treeclient for this account

### Throws

- `TreeClientLoginException` if login failed
- `TreeClientServiceException` if communication failed

---

```
org.astrogrid.scripting.ScriptVoSpaceClient createVoSpaceClient( org.astrogrid.community.User u)
```

---

create a client to access vospace

### Params

- `u` object representing the user for whom to create the client for

*Returns* a vospace client which has the permissions of user `u`

---

```
org.astrogrid.scripting.IOHelper getIoHelper()
```

---

helper object for working with IO

---

```
org.apache.commons.logging.Log getLogger()
```

---

access a standard logging object

---

---

```
org.astrogrid.scripting.ObjectBuilder getObjectBuilder()
```

---

helper object for building account-type objects

*Returns* object that assists in building {@link User} objects, etc.

---

```
org.astrogrid.applications.parameter.protocol.ProtocolLibrary getProtocolLibrary()
```

---

access library object that 'knows' about a variety of IO protocols, and can construct {@link ExternalValue} objects to read / write resources via these protocols.

---

```
org.astrogrid.scripting.table.StarTableBuilder getStarTableBuilder()
```

---

access an object for building star tables from strngs, URLs and ExternalValues

---

```
org.astrogrid.config.Config getSystemConfig()
```

---

acces the system configuration object

*Returns* the system configuration object

---

```
java.lang.String getSystemInfo()
```

---

returns system information for the toolbox

---

```
org.astrogrid.scripting.TableHelper getTableHelper()
```

---

helper object for building, manipulating and writing tables

---

```
java.lang.String getVersion()
```

---

returns version information for the toolbox

---

```
org.astrogrid.portal.workflow.intf.WorkflowManager getWorkflowManager()
```

---

helper oboect for building, saving, submitting and inspecting workflows.

*Throws*

- WorkflowInterfaceException
- 

```
org.astrogrid.scripting.XMLHelper getXmlHelper()
```

---

helper object for working with io

*Returns* object that assists with constructing and manipulatingn xml.

---

```
java.lang.String toString()
```

---

## XMLHelper

helper methods for working with xml. more..

convert document to string

```
java.lang.String elementToString(org.w3c.dom.Element el)
```

convert element to string

```
org.w3c.dom.Document newDocument()
```

create new DOM Document

```
org.w3c.dom.Document newDocument(java.io.InputStream is)
```

parse contents of input stream into document

```
org.w3c.dom.Document newDocument(java.lang.String s)
```

parse contents of string into document

---

## MutableScriptStarTable

Extends uk.ac.starlink.table.AbstractStarTable, uk.ac.starlink.table.RandomStarTable, uk.ac.starlink.table.RowListStarTable

Extension of the standard starlink read-write table that supports the extra scripting methods. See RowListStarTable javadoc

<http://www.star.bristol.ac.uk/~mbt/stil/javadocs/uk/ac/starlink/table/RowListStarTable.html> more..

```
org.astrogrid.scripting.table.MutableScriptStarTable asMutableTable()
```

```
java.util.Iterator columnIterator(int col)
```

```
java.util.Iterator iterator()
```

```
org.astrogrid.scripting.table.ScriptStarTable removeColumn(int index)
```

---

## ScriptStarTable

A Scriptable Star Table - extends basic StarLink StarTable. See StarTable Javadoc - <http://www.star.bristol.ac.uk/~mbt/stil/javadocs/uk/ac/starlink/table/StarTable.html> more..

Return a new table, based on this one with an additional column

### Params

- `meta` metadata for the new column
- `colValue` a value that defines the column value. May either be a static {@link java.lang.reflect.Method}, of type Object method(Object[] arr). The method will be passed the values of the other cells in the row, and the result used as the value of the new cellan instance of {@link ColumnFunction} whose method will be called to define the value of the new cella groovy closure - which will be passed a single parameter - a list containing the values of the other cells in the row. The result returned by the closure will be used as the value of the new cell.any other object - in which case it is used as a constant value for every cell in this column.

Returns a new table, same as this one, with an extra column at the end.

---

```
org.astrogrid.scripting.table.MutableScriptStarTable asMutableTable()
```

---

Return a mutable copy of this table.

Returns a mutable table that has the same structure and contents as the current table.

### Throws

- IOException

---

```
java.util.Iterator columnIterator(int col)
```

---

iterate over each cell in a column

### Params

- `col` column to iterate over

Returns an iteratore that will return the value of each cell in this column.

### Throws

- IOException

---

```
java.util.Iterator iterator()
```

---

iterate over each row in the table.

*Returns* an iterator that will return a {@link java.util.List} of the cells in each row

*Throws*

- IOException

---

```
org.astrogrid.scripting.table.ScriptStarTable removeColumn(int index)
```

---

Return a new table, based on this table, but with one column removed

*Params*

- `index` index of the column to remove.

*Returns* a new table, based on this one, with one less column.

---

## StarTableBuilder

Extends uk.ac.starlink.table.StarTableFactory

factory for ScriptStarTables - extends StarLink's StarTableFactory, but gurantees all tables returned will be instances of {@link org.astrogrid.scripting.table.ScriptStarTable}

StarTableFactory Javadoc -

<http://www.star.bristol.ac.uk/~mbt/stil/javadocs/uk/ac/starlink/table/StarTableFactory.html> more..

---

```
uk.ac.starlink.table.StarTable makeStarTable(java.lang.String location)
```

---

load the contents of a file / url into a scriptable star table

---

```
uk.ac.starlink.table.StarTable makeStarTable(java.lang.String location, java.lang.String format)
```

---



---

```
uk.ac.starlink.table.StarTable makeStarTable(java.net.URL location)
```

---

load the contents of a URL into a scriptable star table

---

```
uk.ac.starlink.table.StarTable makeStarTable(java.net.URL location, java.lang.String format)
```

---

load the contents of a URL into a scriptable star table

---

```
uk.ac.starlink.table.StarTable makeStarTable( org.astrogrid.applications.parameter.protocol.ExternalValue
externalValue)
```

---

construct a star table from an external value

*Params*

- `externalValue` reference to a remote location

*Returns* a star table

*Throws*

- `TableFormatException` if table data is not in a known format
- `IOException` if remote location cannot be accessed

---

```
uk.ac.starlink.table.StarTable makeStarTable( org.astrogrid.applications.parameter.protocol.ExternalValue
externalValue, java.lang.String format)
```

---

construct a star table from an external value

*Params*

- `externalValue` reference to a remote location
- `format` expected table format

*Returns* a star table

*Throws*

- `TableFormatException` if table data is not in expected format
- `IOException` if remote location cannot be accessed

---

```
uk.ac.starlink.table.StarTable makeStarTable(uk.ac.starlink.util.DataSource arg0)
```

---



---

```
uk.ac.starlink.table.StarTable makeStarTable(uk.ac.starlink.util.DataSource arg0, java.lang.String arg1)
```

---



---

```
uk.ac.starlink.table.StarTable makeStarTableFromString(java.lang.String tableContent)
```

---

construct a starTable from the contents of a string

---

```
uk.ac.starlink.table.StarTable makeStarTableFromString(java.lang.String tableContent, java.lang.String format)
```

---

construct a starTable from the contents of a string

---

```
uk.ac.starlink.table.StarTable randomTable(uk.ac.starlink.table.StarTable table)
```

---

return a guaranteed random-access scriptable star table

---

## Config

Defines the methods that a Configurator must implement. Also provides many of the convenience methods, such as `getUrl()`.

There is no store yet but we could add this later

NB - Have deliberately NOT included a `loadStream(Stream)` method - although this might be generically useful, it makes it difficult to track through the Config package where properties have come from. If you need to do this, write your stream of properties to a file and then load from `File.toURL()`

more..

Writes out the configuration keys and values to the given Writer. Used for site debugging. Remember that passwords should be hidden...

---

```
boolean getBoolean(java.lang.String key)
```

---

Typed `getProperty` - returns boolean.

---

```
boolean getBoolean(java.lang.String key, boolean defaultValue)
```

---

Typed `getProperty` - returns boolean.

---

```
java.lang.String getDeploymentId()
```

---

Attempt to resolve deployment-specific name. In this case it looks at the url of this file

---

```
org.w3c.dom.Document getDom(java.lang.String key)
```

---

Indirect typed `getProperty` - returns a DOM loaded from a file at the url specified. ie, looks up the given key to get a url, then attempts to read a DOM from the file at that URL.

---

```
org.w3c.dom.Document getDom(java.lang.String key, java.net.URL defaultUrl)
```

---

Indirect typed `getProperty` - returns a DOM loaded from a file at the url specified. ie, looks up the given key to get a url, then attempts to read a DOM from the file at that URL.

---

```
org.w3c.dom.Document getDom(java.lang.String key, org.w3c.dom.Document defaultDom)
```

---

Indirect typed `getProperty` - returns a DOM loaded from a file at the url specified. ie, looks up the given key to get a url, then attempts to read a DOM from the file at that URL.

---

```
int getInt(java.lang.String key)
```

---

Typed `getProperty` - returns integer. If property is not a valid int, throws a wrapping `ConfigException` as a runtime error

---

```
int getInt(java.lang.String key, int defaultValue)
```

---

Typed `getProperty` - returns integer. If property is not a valid int, throws a wrapping `ConfigException` as a runtime error

---

```
java.lang.Object[] getProperties(java.lang.String key)
```

---

Returns the list of properties identified by the given key. If no matching properties are found, throws a `PropertyNotFoundException`

---

```
java.lang.Object getProperty(java.lang.String key)
```

---

Returns the property identified by the given key. If the property is not found, throws a `PropertyNotFoundException`

---

```
java.lang.Object getProperty(java.lang.String key, java.lang.Object defaultValue)
```

---

Returns the property identified by the given key. If the property is not found, returns the given `defaultValue`

---

```
java.lang.String getString(java.lang.String key)
```

---

Convenience string of `getProperty`

---

```
java.lang.String getString(java.lang.String key, java.lang.String defaultValue)
```

---

Convenience string of `getProperty`

---

```
java.net.URL getUrl(java.lang.String key)
```

---

Typed `getProperty` - returns URL. If property is not a valid url, throws a wrapping `ConfigException` as a runtime error

---

```
java.net.URL getUrl(java.lang.String key, java.net.URL defaultValue)
```

---

Typed `getProperty` - returns URL. If property is not a valid url, throws a wrapping `ConfigException` as a runtime error

---

```
java.util.Set keySet()
```

---

Returns a list of the keys

---

```
void loadFromUrl(java.net.URL url)
```

---

Loads the properties from the file at the given URL

---

```
java.lang.String loadedFrom()
```

---

Returns information about where the values are being found

---

```
java.lang.String resolveEnvironmentVariables(java.lang.String givenSource)
```

---

Resolves environment variables. Looks for `${xxxx}` strings and replace with whatever `xxxx` is set to in the system environment properties

---

```
java.net.URL resolveFilename(java.lang.String givenFilename)
```

---

There are several occasions when an application needs a complete file. For example, a metadata file. This method provides a way of locating that file in different environments. If the filename includes a `${..}` then the contents of the brackets are resolved using the system properties. This allows us to make use of Tomcat's locations for example. If the filename is then absolute, the file is resolved normally. If the filename is relative, the file is searched for first in the classpath then in the working directory. If the path is not found, a `FileNotFoundException` is thrown listing the places looked. If the path is found, a url to it is returned. Hmm not sure if this is really a Configuration thing.... NB this resolves to a URL so it won't find files in jar files

---

```
void setProperty(java.lang.String key, java.lang.Object[] values)
```

---

Sets the property to the given list of values

---

```
void setProperty(java.lang.String key, java.lang.Object value)
```

---

Set property. Stores in cache so it overrides all other properties with the same key.

---

## VoSpace

<a href="#">org.astrogrid.applications.parameter.protocol.Protocol</a>	Factory interface for creating <code>org.astrogrid.applications.parameter.protocol.ExternalValue</code> instances.
<a href="#">org.astrogrid.applications.parameter.protocol.ProtocolLibrary</a>	A library of protocol-handling code, for working with External Values
<a href="#">org.astrogrid.scripting.ScriptVoSpaceClient</a>	Wrapper of standard vospace client that has string-friendly scripting methods.
<a href="#">org.astrogrid.store.Ivorn</a>	International Virtual Observatory Resource Name.
<a href="#">org.astrogrid.store.tree.Container</a>	Representation of a container in an Astrogrid Store.
<a href="#">org.astrogrid.store.tree.File</a>	Representation of a File in an Astrogrid Store
<a href="#">org.astrogrid.store.tree.Node</a>	A wrapper for the AstroGrid StoreFile to make it easier to integrate into Aladin.
<a href="#">org.astrogrid.store.tree.TreeClient</a>	An adapter to enable Aladin to access files in AstroGrid MySpace.

[org.astrogrid.store.VoSpaceClient](#)

This delegate provides methods for operating on files in VoSpace - that is, files that are on store points (accessible through StoreClient implementations) that are Registered in IVO Registries and/or Communities.

---

There's three interfaces for interacting with VoSpace - the vospace client, the tree, and external values

### ExternalValue

Interface for working with a value that is 'external' - ie probably not in this JVM. May be in local storage, may be remote.

Because of this vagueness, the interface provides the bare minimum for working with the external value.  
more..

access a stream to read the contents of the external value from

*Returns* an input stream containing the content of the external value

*Throws*

- InaccessibleExternalValueException

---

java.io.OutputStream **write()**

---

access a stream to write the contents of the external value to

*Returns* an output stream.

*Throws*

- InaccessibleExternalValueException
- 

### Protocol

Factory interface for creating `{@link org.astrogrid.applications.parameter.protocol.ExternalValue}` instances. more..

create a new `indirectParameterValue` for the passed in URI

*Params*

- `reference` the uri to build an instance for.

*Returns* a handler for this uri.

*Throws*

- `InaccessibleExternalValueException`

---

```
java.lang.String getProtocolName()
```

---

access the name of the protocol this object provides support for

*Returns* name of the protocol this factory can build instances for.

---

## ProtocolLibrary

A library of protocol-handling code, for working with External Values more..

build an external value, direct from a URI String

*Params*

- `location` String representation of URI location to build external value for

*Returns* an external value for this location

*Throws*

- `InaccessibleExternalValueException` - if the external value cannot be accessed - e.g. cannot resolve,
- `UnrecognizedProtocolException` - if the uri protocol / scheme is not recognized

---

```
org.astrogrid.applications.parameter.protocol.ExternalValue getExternalValue(java.net.URI location)
```

---

build an external value, direct from a URI

*Params*

- `location` location to build external value for

*Returns* an external value for this location

*Throws*

- `InaccessibleExternalValueException` - if the external value cannot be accessed - e.g. cannot resolve,
  - `UnrecognizedProtocolException` - if the uri protocol / scheme is not recognized
-

```
org.astrogrid.applications.parameter.protocol.ExternalValue  getExternalValue(
org.astrogrid.applications.beans.v1.parameters.ParameterValue  pval)
```

build an external value for a particular indirect parameter value

#### *Params*

- `pval` parameter to build external value for

*Returns* an external value for this parameter.

#### *Throws*

- `InaccessibleExternalValueException` - if the external value cannot be gained - e.g. cannot resolve, no path
- `UnrecognizedProtocolException` - if the indirection uri format is not recognized.

```
boolean  isProtocolSupported(java.lang.String protocol)
```

return true if this library supports the protocol in question

```
java.lang.String[]  listSupportedProtocols()
```

list the protocols supported in this library

## ScriptVoSpaceClient

Extends [org.astrogrid.store.VoSpaceClient](#)

Wrapper of standard vospace client that has string-friendly scripting methods. more..

```
org.astrogrid.store.Ivorn  createUser(java.lang.String arg0, java.lang.String arg1)
```

```
void  delete(java.lang.String arg0)
```

```
void  deleteUser(java.lang.String arg0, java.lang.String arg1)
```

```
org.astrogrid.store.delegate.StoreClient  getDelegate(java.lang.String arg0)
```

```
java.io.InputStream  getStream(java.lang.String arg0)
```

```
void  move(java.lang.String arg0, java.lang.String arg1)
```

```
.....
void newFolder(java.lang.String arg0)
.....
```

```
.....
void putBytes(byte[] arg0, int arg1, int arg2, java.lang.String arg3, boolean arg4)
.....
```

```
.....
java.io.OutputStream putStream(java.lang.String arg0)
.....
```

```
.....
void putUrl(java.net.URL arg0, java.lang.String arg1)
.....
```

```
.....
void putUrl(java.net.URL arg0, java.lang.String arg1, boolean arg2)
.....
```

---

## Ivorn

International Virtual Observatory Resource Name. Used to name specific IVO resources.

They act as keys to VO Registries; give the registry an IVORN and it will return the associated VOResource document

Ivorns are of the form:

```
ivo://something/anything/athing/etc#somethingwithmeaningtothiscontext
```

For example:

```
ivo://roe.ac.uk/storee#path/to/file.ext
```

*might* resolve to an FTP server, on which the path 'path/to/file.ext' would locate a document.

Ivorns are immutable - ie once created, they cannot be changed. You can make new ones out of old ones.

This is a badly named package. IVORNs can be used for things other than stores...

more..

Returns specific-to-service - ie fragment

```
.....
java.lang.String getPath()
.....
```

Returns ivo id without the scheme. eg 'ivo://roe.ac.uk/mch/myspace' returns 'roe.ac.uk/mch/myspace'

---

```
java.lang.String getScheme()
```

---

Returns identifier scheme

---

```
boolean isIvorn(java.lang.String aString)
```

---

Returns true if the given string is likely to be an ivorn - ie if it starts with ivo://

---

```
java.lang.String toRegistryString()
```

---

Representation to be used when submitting the IVORN to a registry to be resolved

---

```
java.lang.String toString()
```

---

String representation

---

```
java.net.URI toUri()
```

---

Returns the IVORN in URI form

---

## VoSpaceClient

This delegate provides methods for operating on files in VoSpace - that is, files that are on store points (accessible through `StoreClient` implementations) that are Registered in IVO Registries and/or Communities.

Note that there is a difference between the user (termed **operator** here) of this class, and the owner of the files that might be being browsed.

more..

Copy a file

---

```
org.astrogrid.store.Ivorn createUser( org.astrogrid.store.Ivorn targetVoSpace,
org.astrogrid.store.Ivorn user)
```

---

Creates a new user (identified by the given user ivorn) on the storepoint given by the 'target' Ivorn.

---

```
void delete( org.astrogrid.store.Ivorn toDelete)
```

---

Delete a file

---

```
void deleteUser( org.astrogrid.store.Ivorn target, org.astrogrid.store.Ivorn user)
```

---

---

```
org.astrogrid.store.delegate.StoreClient getDelegate( org.astrogrid.store.Ivorn storeIvorn)
```

---

Returns a StoreClient delegate for the given ivorn for direct file manipulation

```
org.astrogrid.store.delegate.StoreFile getFile( org.astrogrid.store.Ivorn fileIvorn)
```

---

Returns the StoreFile representation of the file at the given AGSL

```
org.astrogrid.community.User getOperator()
```

---

Returns the user of this delegate - ie the account it is being used by

```
java.io.InputStream getStream( org.astrogrid.store.Ivorn source)
```

---

Gets a file's contents as a stream

```
void move( org.astrogrid.store.Ivorn source, org.astrogrid.store.Ivorn target)
```

---

Moves/Renames a file

```
void newFolder( org.astrogrid.store.Ivorn target)
```

---

Create a container

```
void putBytes(byte[] bytes, int offset, int length, org.astrogrid.store.Ivorn
targetIvorn, boolean append)
```

---

Puts the given byte buffer from offset of length bytes, to the given target

```
java.io.OutputStream putStream( org.astrogrid.store.Ivorn target)
```

---

Streaming output - returns a stream that can be used to output to the given location

```
void putUrl(java.net.URL source, org.astrogrid.store.Ivorn targetIvorn, boolean append)
```

---

Copies the contents of the file at the given source url to the given location

---

## Container

Representation of a container in an Astrogrid Store. more..

Add a child to a container.

*Params*

- `name` The container name.

*Throws*

- `TreeClientDuplicateException` If the container already exists.
- `TreeClientServiceException` If the service is unable to handle the request.

---

```
org.astrogrid.store.tree.File addFile(java.lang.String name)
```

---

Add a file to this container.

*Params*

- `name` The file name.

*Throws*

- `TreeClientDuplicateException` If the file already exists.
- `TreeClientServiceException` If the service is unable to handle the request.

---

```
java.util.Collection getChildNodes()
```

---

Get a list (collection) of the current child nodes. Note, you cannot add a child node by adding a node to the collection.

*Returns* An unmodifiable collection of Node(s) for the child nodes.

---

## File

Representation of a File in an Astrogrid Store more..

Get an InputStream to read data from the file.

*Throws*

- `TreeClientServiceException` If the service is unable to handle the request.

---

```
java.lang.String getMimeType()
```

---

Get the mime type for the file.

*Returns* The mime type for the file contents, or null if is not set..

*Throws*

- `TreeClientServiceException` If the service is unable to handle the request.

---

```
java.io.OutputStream getOutputStream()
```

---

Get an `OutputStream` to send data to the file. Opening a new stream to an existing file will over-write the file contents. The client **MUST** close the output stream to force the transfer to complete.

*Throws*

- `TreeClientServiceException` If the service is unable to handle the request.
- 

## Node

A wrapper for the AstroGrid StoreFile to make it easier to integrate into Aladin. more..

Access to the node name.

*Returns* The node name.

---

```
boolean isContainer()
```

---

Check if this represents a container.

*Returns* true if this represents a container.

---

```
boolean isFile()
```

---

Check if this represents a file.

*Returns* true if this represents a file.

---

## TreeClient

An adapter to enable Aladin to access files in AstroGrid MySpace. more..

Get the root node of the account home space.

*Returns* An `AladinAdapterContainer` representing the root of the account myspace.

*Throws*

- `TreeClientSecurityException` If the adapter is not logged in.
- `TreeClientServiceException` If unable to handle the request.

---

`org.astrogrid.community.common.security.data.SecurityToken` `getToken()`

---

Access to the current security token.

*Returns* The current security token, or null if not logged in.

---

`void login( org.astrogrid.store.Ivorn ivorn, java.lang.String password)`

---

Login to the AstroGrid community.

*Params*

- `ivorn` The 'ivo://...' identifier for the AstroGrid account.
- `password` The account password.

*Throws*

- `TreeClientLoginException` If the login fails.
- `TreeClientServiceException` If unable to handle the request.

---

`void logout()`

---

Logout from the AstroGrid community.

*Throws*

- `TreeClientServiceException` If unable to handle the request.

---

## JES Server

`org.astrogrid.jes.jobscheduler.impl.groovy.WorkflowLogger`

allows workflow scripts to emit log messages

The `JesInterface` class gives access into some of the internals of the current JES server. An instance of this class named `jes` is available in the JEScript environment. The `JesInterface` class in turn extends `WorkflowLogger`, and so inherits it's methods too.

### JesInterface

Extends `org.astrogrid.jes.jobscheduler.impl.groovy.WorkflowLogger`

this object allows access to some of the internals of the jes server the workflow script is executing on more..

access an element in the workflow document by it's unique id.

```
java.util.List getSteps()
```

access the list of steps in the current workflow

```
java.lang.String getVersion()
```

inspect version information for this server

```
org.astrogrid.workflow.beans.v1.Workflow getWorkflow()
```

access the object model of the current workflow

```
org.astrogrid.applications.beans.v1.parameters.ParameterValue newParameter()
```

create a new tool-parameter object, initialized, but unattached to any tool

## WorkflowLogger

methods in this class allow a script to add messages to the execution record for the workflow, at different log levels. At moment, debug and info messages are both logged at 'info' level. Error and fatal messages are both logged at 'error' level more..

```
void debug(java.lang.Object arg0, java.lang.Throwable arg1)
```

```
void error(java.lang.Object arg0)
```

```
void error(java.lang.Object arg0, java.lang.Throwable arg1)
```

```
void fatal(java.lang.Object arg0)
```

```
void fatal(java.lang.Object arg0, java.lang.Throwable arg1)
```

```
void info(java.lang.Object arg0)
```

---

```
void info(java.lang.Object arg0, java.lang.Throwable arg1)
```

---

```
void warn(java.lang.Object arg0)
```

---

```
void warn(java.lang.Object arg0, java.lang.Throwable arg1)
```

---

## Workflow Delegate

<a href="#">org.astrogrid.portal.workflow.intf.ApplicationDescription</a>	Description of an application
<a href="#">org.astrogrid.portal.workflow.intf.ApplicationDescriptionSummary</a>	a summary of an application
<a href="#">org.astrogrid.portal.workflow.intf.ApplicationRegistry</a>	query registry for applications
<a href="#">org.astrogrid.portal.workflow.intf.JobExecutionService</a>	A component that can execute and manage jobs
<a href="#">org.astrogrid.portal.workflow.intf.WorkflowBuilder</a>	A component of methods to help build workflow documents
<a href="#">org.astrogrid.portal.workflow.intf.WorkflowManager</a>	root object of the jes client interface
<a href="#">org.astrogrid.portal.workflow.intf.WorkflowStore</a>	A component that can store and retrieve workflow documents from VoSpace.

---

These classes provide a high-level interface to the Astrogrid workflow system. They provide methods for building workflows, loading and saving to VoSpace, querying registry for cea applications, and submitting workflows to JES for execution.

### ActivityKey

Implementation of activity keys for the new workflow object model more..

create a key for the current position in a workflow

*Params*

- `root` document root object
- `current` object to create activity key for

*Returns* activity key that points to the current object.

*Throws*

- `IllegalArgumentException` - if `root` or `current` is null, or `current` is not within the workflow object tree

```
boolean equals(java.lang.Object o)
```

Returns true if this `ActivityKey` is the same as the `o` argument.

*Returns* true if this `ActivityKey` is the same as the `o` argument.

```
org.astrogrid.common.bean.BaseBean extractFrom(org.astrogrid.workflow.beans.v1.Workflow root)
```

apply the activity key to a document, to access the object it refers to

*Params*

- `root` workflow document

*Returns* the object in the tree that the activity key pointed to, or null if no object was found

```
int hashCode()
```

Override `hashCode`.

*Returns* the Objects `hashCode`.

```
java.lang.String toString()
```

## ApplicationDescription

Description of an application

This class provides access to the descriptor of an application - by wrapping a `{@link org.astrogrid.applications.beans.v1.ApplicationBase}` object.

- `{@link #getInterfaces}`
- `{@link #getName}`
- `{@link #getParameters}`

It also provides methods to create a new workflow `{@link org.astrogrid.workflow.beans.v1.Tool}` object that can be used to call the application this class describes

- `{@link #createToolFromDefaultInterface}`
- `{@link #createToolFromInterface}`

ParameterValues can be added to a `Tool` by hand, with the assistance of these methods:

- `{@link #createValueFromDefinition}`
- `{@link #getDefinitionForValue}`
- `{@link #getReferenceForValue}`
- `{@link #getDefinitionForReference}`

Finally, a `Tool` object can be checked that it conforms to the application description by calling the `{@link #validate}` method

more..

construct a new, initialized tool from the default (first) interface in the application descriptor

---

```
org.astrogrid.workflow.beans.v1.Tool createToolFromInterface(
org.astrogrid.applications.beans.v1.Interface intf)
```

---

construct a new, initialized tool from an interface in this application descriptor

*Params*

- `intf` the interface to create a tool instance from

*Returns* a populated tool object

*Throws*

- `IllegalArgumentException` if the interface does not belong to this application description.

---

```
org.astrogrid.applications.beans.v1.parameters.ParameterValue createValueFromDefinition(
org.astrogrid.applications.beans.v1.parameters.BaseParameterDefinition defn)
```

---

from a definition of a parameter, create a new, initialized parameter value.

*Params*

- `defn` definition of the parameter

*Returns* a parameter value with name, type initialized. if `defn` provides a `defaultValue`, this is set as content

---

```
org.astrogrid.applications.beans.v1.parameters.BaseParameterDefinition getDefinitionForReference(
org.astrogrid.applications.beans.v1.ParameterRef parameterRef)
```

---

interfaces contain a bunch of 'ParameterRef' objects, that reference 'BaseParameterDefinitions' that are defined elsewhere, and give the type, user interface description, etc of the paramter. This method will get a definition for a parameter reference

#### *Params*

- `parameterRef` the reference to look up

*Returns* the associated parameter definition.

#### *Throws*

- `IllegalArgumentException` when asked to find definition for a reference that is not in the application description.

---

```
org.astrogrid.applications.beans.v1.parameters.BaseParameterDefinition getDefinitionForValue(
org.astrogrid.applications.beans.v1.parameters.ParameterValue paramVal,
org.astrogrid.applications.beans.v1.Interface intf)
```

---

convenience method to get a parameter definition for a parameter value. equivalent to `getDefinitionForReference(getReferenceForValue(p,i))`

#### *Params*

- 
- 

#### *Throws*

- `IllegalArgumentException`

---

```
org.astrogrid.applications.beans.v1.InterfacesType getInterfaces()
```

---



---

```
java.lang.String getName()
```

---



---

```
org.w3c.dom.Element getOriginalVODescription()
```

---

access the xml of the original registry entry that defines this Application

*Returns* dom of registry entry. may be null; if not, root element will be `<VODescription>`

---

```
org.astrogrid.applications.beans.v1.Parameters getParameters()
```

---



---

```
org.astrogrid.applications.beans.v1.ParameterRef getReferenceForValue(
org.astrogrid.applications.beans.v1.parameters.ParameterValue paramVal,
org.astrogrid.applications.beans.v1.Interface intf)
```

---

find the parameter reference associated with a parameter value in a particular interface

---

```
void validate( org.astrogrid.workflow.beans.v1.Tool t)
```

---

verify that a tool object matches what is expected by the application (as defined by this descriptor) at moment, checks against information in the application description. in future, could access web services to resolve ucds, or check url parameters, etc.

---

### ApplicationDescriptionSummary

a summary of an application. more..

array of the interface names for this application

```
java.lang.String getName()
```

---

system name of the application

```
java.lang.String getUIName()
```

---

user-friendly name of the application

---

### ApplicationRegistry

high-level interface to the registry, returning information relevant to building workflow documents more..

query registry for description of a named application

#### *Params*

- `applicationName` name of the application.

*Returns* an description of the application

#### *Throws*

- `WorkflowInterfaceException` if an error occurs - i.e. if the application is not found

```
java.lang.String[] listApplications()
```

---

list names of known tools

```
org.astrogrid.portal.workflow.intf.ApplicationDescriptionSummary[] listUIApplications()
```

---

list system names and user-friendly names of applications.

---

### JobExecutionService

A component that can execute and manage jobs more..

cancel a job - halt the execution of it does nothing at the moment

#### *Params*

- `jobURN` unique identifier of the job

#### *Throws*

- `WorkflowInterfaceException`

---

```
void deleteJob( org.astrogrid.workflow.beans.v1.execution.JobURN jobURN)
```

---

delete a job - remove all record of it from the jes store does nothing at the moment

#### *Params*

- `jobURN` unique identifier of the job to delete

---

```
org.astrogrid.workflow.beans.v1.execution.WorkflowSummaryType[] listJobs(
org.astrogrid.community.beans.v1.Account account)
```

---

list jobs for a particular user.

#### *Params*

- `account` identifies the user

*Returns* array of workflow summaries

#### *Throws*

- `WorkflowInterfaceException`

---

```
org.astrogrid.workflow.beans.v1.Workflow readJob( org.astrogrid.workflow.beans.v1.execution.JobURN
jobURN)
```

---

Retrieve an annotated workflow document from the jes store

*Params*

- `jobURN` unique job id

*Returns* workflow document, plus annotations. never null

*Throws*

- `WorkflowInterfaceException` if document cannot be retrieved.

---

```
org.astrogrid.jes.delegate.JobSummary[] readJobList( org.astrogrid.community.beans.v1.Account account)
```

---

retrieve list of jobs for a particular user

*Params*

- `account` identifies a user

*Returns* array of job summaries

*Throws*

- `WorkflowInterfaceException`

---

```
org.astrogrid.workflow.beans.v1.execution.JobURN submitWorkflow(  
org.astrogrid.workflow.beans.v1.Workflow workflow)
```

---

submit a workflow to the job controller

*Params*

- `workflow` workflow document to submit (contains user credentials to execute under)

*Returns* new unique identifier for the job. never null

*Throws*

- `WorkflowInterfaceException` if job cannot be submitted
- 

## **WorkflowBuilder**

A component of methods to help build workflow documents more..

create a workflow object

#### *Params*

- `creds` credentials for this workflow - contains account, group and authentication token for the user this workflow is going to run as.
- `name` name of the new workflow
- `description` a textual description of the workflow

*Returns* a new workflow document object, that is schema-valid. never null

---

### **WorkflowManager**

root of object tree for constructing workflows, saving and loading to myspace, querying registry for application information, and submitting workflows for execution. more..

get a component to manage jobs - executions of workflows. never null

.....  
`org.astrogrid.portal.workflow.intf.ApplicationRegistry` `getToolRegistry()`  
.....

get a component to query a registry. never null

.....  
`org.astrogrid.portal.workflow.intf.WorkflowBuilder` `getWorkflowBuilder()`  
.....

get a component to build workflows. never null

.....  
`org.astrogrid.portal.workflow.intf.WorkflowStore` `getWorkflowStore()`  
.....

get a component to access a store. never null

---

### **WorkflowStore**

A component that can store and retrieve workflow documents from VoSpace. (i.e. myspace) more..

read workflow from the store

*Params*

- `user` account details for the owner of the workflow document
- `locationToReadFrom` ivorn location to read the workflow from.

*Returns* workflow document object. will never return null

*Throws*

- `WorkflowInterfaceException` if document can't be loaded

---

```
void saveWorkflow( org.astrogrid.community.User user, org.astrogrid.store.Ivorn locationToSaveTo,
org.astrogrid.workflow.beans.v1.Workflow workflow)
```

---

save workflow to myspace

*Params*

- `user` account details for the owner of the workflow
- `locationToSaveTo` ivorn location to save the document
- `workflow` workflow document to save

*Throws*

- `WorkflowInterfaceException`
- 

## Registry Delegate

[org.astrogrid.registry.client.admin.UpdateRegistry](#)

Class Name: `RegistryAdminService` Description: This class represents the client webservice delegate to the Administration piece of the web service.

[org.astrogrid.registry.client.query.QueryRegistry](#)

The `QueryRegistry` class is a delegate to a web service that submits an XML formatted registry query to the to the server side web service also named the same `RegistryService`.

[org.astrogrid.registry.client.query.RegistryService](#)

The `RegistryService` class is a delegate to a web service that submits an XML formatted registry query to the to the server side web service also named the same `RegistryService`.

[org.astrogrid.registry.client.query.ServiceData](#)

---

Delegates to query and adminisister a registry

### RegistryAdminService

Class Name: `RegistryAdminService` Description: This class represents the client webservice delegate to the Administration piece of the web service. It uses the same Interface as the server side webservice so they both implement and handle the same web service method names. The primary goal of this class is to

establish a Axis-Message style webservice call to the server. more..

```
.....
org.w3c.dom.Document getStatus()
.....
```

```
.....
void harvestResource(org.w3c.dom.Document harvestDoc)
.....
```

```
.....
org.w3c.dom.Document update(org.w3c.dom.Document update)
.....
```

Takes an XML Document to send to the update server side web service call. Establishes a service and a call to the web service and call it's update method, using an Axis-Message style. Then updates this document onto the registry.

#### *Params*

- *query* Document a XML document dom object to be updated on the registry.

*Returns* the document updated on the registry is returned.

```
.....
org.w3c.dom.Document updateFromFile(java.io.File fi)
.....
```

```
.....
org.w3c.dom.Document updateFromURL(java.net.URL location)
.....
```

## UpdateRegistry

Class Name: RegistryAdminService Description: This class represents the client webservice delegate to the Administration piece of the web service. It uses the same Interface as the server side webservice so they both implement and handle the same web service method names. The primary goal of this class is to establish a Axis-Message style webservice call to the server. more..

```
.....
org.w3c.dom.Document getStatus()
.....
```

```
.....
void harvestResource(org.w3c.dom.Document harvestDoc)
.....
```

```
.....
org.w3c.dom.Document update(org.w3c.dom.Document update)
.....
```

Takes an XML Document to send to the update server side web service call. Establishes a service and a call to the web service and call it's update method, using an Axis-Message style. Then updates this document onto the registry.

*Params*

- `query` Document a XML document dom object to be updated on the registry.

*Returns* the document updated on the registry is returned.

---

```
org.w3c.dom.Document updateFromFile(java.io.File fi)
```

---

```
org.w3c.dom.Document updateFromString(java.lang.String voresources)
```

---

```
org.w3c.dom.Document updateFromURL(java.net.URL location)
```

---

## QueryRegistry

The QueryRegistry class is a delegate to a web service that submits an XML formatted registry query to the to the server side web service also named the same RegistryService. This delegate helps the user browse the registry and also the OAI. more..

Query for a specific resource in the Registry based on its identifier element(s), Then extracts out the AccessURL element to find the endpoint. If the endpoint is a web service and has a "?wsdl" ending then attempts to parse the wsdl to obtain certain information such as endpoints, and port names.

*Params*

- `string` identifier of the resource.

*Returns* String of a url.

---

```
org.astrogrid.registry.common.WSDLBasicInformation getBasicWSDLInformation(org.w3c.dom.Document voDoc)
```

---

Query for a specific resource in the Registry based on its identifier element(s), Then extracts out the AccessURL element to find the endpoint. If the endpoint is a web service and has a "?wsdl" ending then attempts to parse the wsdl to obtain certain information such as endpoints, and port names.

*Params*

- `string` identifier of the resource.

*Returns* String of a url.

---

```
java.lang.String getEndPointByIdentifier(java.lang.String ident)
```

---

Query for a specific resource in the Registry based on its identifier element(s), Then extracts out the AccessURL element to find the endpoint. If the endpoint is a web service and has a "?wsdl" ending then attempts to parse the wsdl for the end point of the service.

*Params*

- string identifier of the resource.

*Returns* String of a url.

---

```
java.lang.String getEndPointByIdentifier( org.astrogrid.store.Ivorn ident)
```

---

Query for a specific resource in the Registry based on its identifier element(s), Then extracts out the AccessURL element to find the endpoint. If the endpoint is a web service and has a "?wsdl" ending then attempts to parse the wsdl for the end point of the service.

*Params*

- ivorn object.

*Returns* String of a url.

---

```
java.net.URL[] getEndPointByInterfaceType( org.astrogrid.registry.common.InterfaceType interfaceType)
```

---

```
org.w3c.dom.Document getRecord(java.lang.String identifier)
```

---

OAI - Get a specific record from OAI given an identifier. Defaults the metadataPrefix to ivo\_vor.

*Params*

- identifier for a particular record ex: ivo\_vor://astrogrid.org/Registry

*Returns* XML DOM of an OAI-PMH for the GetRecord.

---

```
org.w3c.dom.Document getRecord(java.lang.String identifier, java.lang.String metadataPrefix)
```

---

OAI - Get a specific record for an identifier and metadata prefix

*Params*

- identifier for a particular record ex: ivo\_vor://astrogrid.org/Registry
- metadataPrefix is the oai prefix/id to be used, currently only ivo\_vor and oai\_dc.

*Returns* XML DOM of an OAI-PMH for the GetRecord.

---

```
org.w3c.dom.Document getRegistries()
```

---

Performs a query to return all Resources of a type of Registry.

Returns XML DOM of Resources queried from the registry.

Throws

- RegistryException problem during the query server or client side.

---

```
org.w3c.dom.Document getResourceByIdentifier(java.lang.String ident)
```

---

Query for a specific resource in the Registry based on its identifier element(s). Essentially creates a search query "old style astrogrid" for now. Based on the identifier.

Params

- identifier string.

Returns XML DOM of Resource queried from the registry.

---

```
org.w3c.dom.Document getResourceByIdentifier(org.astrogrid.store.Ivorn ident)
```

---

Query for a specific resource in the Registry based on its identifier element(s). Essentially creates a search query "old style astrogrid" for now. Based on the identifier.

Params

- identifier IVORN object.

Returns XML DOM of Resource queried from the registry.

---

```
org.astrogrid.registry.client.query.ServiceData[] getResourcesByInterfaceType(
org.astrogrid.registry.common.InterfaceType interfaceType)
```

---



---

```
org.w3c.dom.Document identify()
```

---

Identify - Queries based on OAI-Identify verb, identifying the repository.

Returns XML DOM of an OAI-PMH for the Identify.

---

```
org.w3c.dom.Document listIdentifiers()
```

---

OAI - ListIdentifiers call, similar to ListRecords but only returns the identifiers (unique ids) for the records. Defaults the metadataPrefix to ivo\_vor.

Returns XML DOM of an OAI-PMH for the ListIdentifiers.

---

```
org.w3c.dom.Document listIdentifiers(java.lang.String metadataPrefix, java.util.Date fromDate, java.util.Date untilDate)
```

---

OAI - ListIdentifiers call, similar to ListRecords but only returns the identifiers (unique ids) for the records. Defaults the metadataPrefix to ivo\_vor.

*Params*

- metadataPrefix the oai prefix; normally ivo\_vor. Also available is oai\_dc.
- fromDate - A from date for returning Resources from a date till now.
- untilDate - Returning resources from the beginning till a date.

*Returns* XML DOM of an OAI-PMH for the ListIdentifiers.

---

```
org.w3c.dom.Document listMetadataFormats(java.lang.String identifier)
```

---

ListMetadataFormats - OAI ListMetadataFormats verb call. With an optional identifier string to list the metadata formats for a particular id.

*Returns* XML DOM of an OAI-PMH for the ListMetadataFormats.

---

```
org.w3c.dom.Document listRecords()
```

---

ListRecords - OAI ListRecords query, the Registry server will default the metadataPrefix to ivo\_vor.

*Returns* XML DOM of an OAI-PMH for the ListRecords.

---

```
org.w3c.dom.Document listRecords(java.lang.String metadataPrefix, java.util.Date fromDate, java.util.Date untilDate)
```

---

ListRecords - OAI ListRecords query. This will be the most used OAI verb for harvesting.

*Params*

- metadataPrefix - oai metadataPrefix string normally ivo\_vor or oai\_dc. A null will let the Registry server default it to ivo\_vor.
- fromDate - A from date for returning Resources from a date till now.
- untilDate - Returning resources from the beginning till a date.

*Returns* XML DOM of an OAI-PMH for the ListRecords.

---

```
org.w3c.dom.Document listRecords(java.util.Date fromDate)
```

---

ListRecords - OAI ListRecords query based on a fromDate, the records changed from that date The Registry server will default the metadataPrefix to ivo\_vor

*Params*

- fromDate - A from date for returning Resources from a date till now.

*Returns* XML DOM of an OAI-PMH for the ListRecords.

---

```
org.w3c.dom.Document loadRegistry()
```

---

Loads this registry type resource for the registry. Essentially querying for one resource that defines the Registry.

*Returns* XML DOM of Resources queried from the registry.

---

```
java.util.HashMap managedAuthorities()
```

---

Queries for all the authorities managed by this registry. By loading this registries main registry resource type and looking for the ManagedAuthority elements, currently does not work with version 0.10. But is slowly being factored out of use.

*Returns* a hashmap of all the managed authority id's.

---

```
org.w3c.dom.Document search(java.lang.String adql)
```

---

To perform a query with ADQL, using adql string.

*Params*

- adql string form of adql (xml)

*Returns* XML DOM of Resources queried from the registry.

*Throws*

- RegistryException problem during the query server or client side.

---

```
org.w3c.dom.Document search(org.w3c.dom.Document adql)
```

---

To perform a query on the Registry using a DOM conforming of ADQL. Uses a Axis-Message type style so wrap the DOM in the method name conforming to the WSDL.

*Params*

- adql string form of adqls

*Returns* XML DOM of Resources queried from the registry.

*Throws*

- RegistryException problem during the query server or client side.

---

```
org.w3c.dom.Document searchFromSADQL(java.lang.String adql)
```

---

To perform a query with ADQL, using adqls.

*Params*

- adql string form of adqls

*Returns* XML DOM of Resources queried from the registry.

*Throws*

- RegistryException problem during the query server or client side.

---

```
org.w3c.dom.Document submitQuery(java.lang.String query)
```

---

Old style xml in string form to perform a query. To be deprecated soon, but currently other astrogrid components use this method. Created before the standard of ADQL.

*Params*

- the xml string version of the old style astrogrid query language..

*Returns* XML DOM of Resources queried from the registry.

---

```
org.w3c.dom.Document submitQuery(org.w3c.dom.Document query)
```

---

Old style form to perform a query.

*Params*

- xml version of the old style astrogrid query language.

*Returns* XML DOM of Resources queried from the registry.

---

## RegistryService

The RegistryService class is a delegate to a web service that submits an XML formatted registry query to the to the server side web service also named the same RegistryService. This delegate helps the user browse the registry. Queries should be formatted according to the schema at IVOA schema version 0.9. This class also uses the common RegistryInterface for knowing the web service methods to call on the server side. more..

---

```
org.astrogrid.registry.common.WSDLBasicInformation getBasicWSDLInformation(org.w3c.dom.Document voDoc)
```

---

```
java.lang.String getEndPointByIdentifier(java.lang.String ident)
```

---

```
java.lang.String getEndPointByIdentifier( org.astrogrid.store.Ivorn ident)
.....

java.net.URL[] getEndPointByInterfaceType( org.astrogrid.registry.common.InterfaceType interfaceType)
.....

org.w3c.dom.Document getRecord(java.lang.String identifier)
.....

org.w3c.dom.Document getRecord(java.lang.String identifier, java.lang.String metadataPrefix)
.....

org.w3c.dom.Document getRegistries()
.....

org.w3c.dom.Document getResourceByIdentifier(java.lang.String ident)
.....

org.w3c.dom.Document getResourceByIdentifier( org.astrogrid.store.Ivorn ident)
.....

org.astrogrid.registry.client.query.ServiceData[] getResourcesByInterfaceType(
org.astrogrid.registry.common.InterfaceType interfaceType)
.....

org.w3c.dom.Document identify()
.....

org.w3c.dom.Document listIdentifiers()
.....

org.w3c.dom.Document listIdentifiers(java.lang.String metadataPrefix, java.util.Date fromDate, java.util.Date untilDate)
.....

org.w3c.dom.Document listMetadataFormats(java.lang.String identifier)
.....

org.w3c.dom.Document listRecords()
.....

org.w3c.dom.Document listRecords(java.lang.String metadataPrefix, java.util.Date fromDate, java.util.Date untilDate)
.....

org.w3c.dom.Document listRecords(java.util.Date fromDate)
.....

org.w3c.dom.Document loadRegistry()
.....
```

```
java.util.HashMap managedAuthorities()
```

```
org.w3c.dom.Document search(java.lang.String xadql)
```

```
org.w3c.dom.Document search(org.w3c.dom.Document adql)
```

```
org.w3c.dom.Document submitQuery(java.lang.String query)
```

```
org.w3c.dom.Document submitQuery(org.w3c.dom.Document query)
```

---

## ServiceData

more..

```
java.net.URL getAccessURL()
```

```
java.lang.String getDescription()
```

```
java.util.Collection getInterface()
```

```
org.astrogrid.store.Ivorn getIvorn()
```

```
java.lang.String getTitle()
```

```
void setAccessURL(java.net.URL accessURL)
```

```
void setDescription(java.lang.String description)
```

```
void setIvorn( org.astrogrid.store.Ivorn ivorn)
```

```
void setTitle(java.lang.String title)
```

---

```
java.lang.String toString()
```

---

## CEA Delegate

[org.astrogrid.applications.delegate.impl.CommonExecutionConnectorDelegate](#)

[org.astrogrid.applications.delegate.impl.CommonExecutionConnectorDelegateImpl](#)

---

These classes provide a low-level interface to interact with CEA servers.

### CommonExecutionConnectorClient

A client side interface for the `{@link CommonExecutionConnector}`. This interface uses castor objects derived from the schema instead of the axis ones. more..

Abort an running application.

#### *Params*

- `executionId` the identifier for the execution instance that is to be aborted.

*Returns* true if aborted successfully.

#### *Throws*

- `RemoteException`
- `CeaFault`

---

```
boolean execute(java.lang.String executionId)
```

---

execute a previously-initialized application

#### *Params*

- `executionId` the id of the application to start

*Returns* true if execution was started successfully

#### *Throws*

- `CEADelegateException`

---

```
org.astrogrid.applications.beans.v1.cea.castor.ExecutionSummaryType  
getExecutionSummary(java.lang.String executionId)
```

---

retrieve summary information for an applicatin execution - this structure contains input parameters, results, plus execution status.

#### *Params*

- `executionId` the identifier for the instance to query.

#### *Throws*

- `CEADelegateException`

---

```
org.astrogrid.applications.beans.v1.cea.castor.ResultListType getResults(java.lang.String executionId)
```

---

access the results of an application execution

#### *Params*

- `executionId` the identifier for the execution instance to query

---

```
java.lang.String init( org.astrogrid.workflow.beans.v1.Tool tool,
org.astrogrid.jes.types.v1.cea.axis.JobIdentifierType jobstepID)
```

---

Initialize an application asynchronously. The application may be a command line application, a data query or another web service depending on the type of Common Execution Controller that is being contacted.

#### *Params*

- `tool` This is the specification of the application that will be run.
- `jobstepID` An identifier that the caller can use to keep track of this particular execution instance.
- `jobMonitorURL` The endpoint of a JobMonitor service that should be called back when the execution has finished.

*Returns* An identifier that the CommonExecutionConnector service uses to track this instance of the application execution. This is used as an argument to some of the other methods in this interface.

#### *Throws*

- `RemoteException`
- `CeaFault`

---

```
org.astrogrid.applications.beans.v1.cea.castor.MessageType
queryExecutionStatus(java.lang.String executionId)
```

---

query execution status of an application

*Params*

- `executionId` identifier of application to query

*Returns* message, containing various metadata, including status code.

*Throws*

- `CEADelegateException`

```
void registerProgressListener(java.lang.String executionId, java.net.URI listenerEndpoint)
```

register a listener to the progress of an application. the web interface of the listener will be called to notify it of execution events of this application

*Params*

- `executionId` the id of the application to listen to
- `listenerEndpoint` endpoint of the web service (must implement the `{@link org.astrogrid.jes.delegate.v1.JobMonitor}` interface)

*Throws*

- `CEADelegateException`

```
void registerResultsListener(java.lang.String executionId, java.net.URI listenerEndpoint)
```

register a listener / consumer for the results of this application execution.

*Params*

- `executionId` the id of the application to consume results from.
- `listenerEndpoint` endpoint of the web service that will consume results (must implement the `{@link org.astrogrid.jes.service.v1.ceareresults.ResultsLIstener}` interface)

*Throws*

- `CEADelegateException`

```
java.lang.String returnRegistryEntry()
```

return the registry entry for this common execution controller.

*Throws*

- `RemoteException`

## CommonExecutionConnectorDelegate

Extends [org.astrogrid.common.delegate.AbstractDelegate](#)

more..

---

```
org.astrogrid.applications.delegate.impl.CommonExecutionConnectorDelegate
  buildDelegate(java.lang.String targetEndPoint, int timeout)
```

---

### CommonExecutionConnectorDelegateImpl

Extends [org.astrogrid.common.delegate.AbstractDelegate](#) ,  
[org.astrogrid.applications.delegate.impl.CommonExecutionConnectorDelegate](#)

more..

---

```
boolean execute(java.lang.String executionId)
```

---

```
org.astrogrid.applications.beans.v1.cea.castor.ExecutionSummaryType
  getExecutionSummary(java.lang.String executionId)
```

---

```
org.astrogrid.applications.beans.v1.cea.castor.ResultListType  getResults(java.lang.String executionId)
```

---

```
java.lang.String init( org.astrogrid.workflow.beans.v1.Tool tool,
  org.astrogrid.jes.types.v1.cea.axis.JobIdentifierType jobstepID)
```

---

```
org.astrogrid.applications.beans.v1.cea.castor.MessageType
  queryExecutionStatus(java.lang.String executionId)
```

---

```
void registerProgressListener(java.lang.String executionId, java.net.URI listenerEndpoint)
```

---

```
void registerResultsListener(java.lang.String executionId, java.net.URI listenerEndpoint)
```

---

```
java.lang.String returnRegistryEntry()
```

---

## User Objects

<a href="#">org.astrogrid.community.beans.v1.BaseIdentifier</a>	The identifier base class.
<a href="#">org.astrogrid.community.beans.v1.Credentials</a>	The full authorization and authentication credentials.
<a href="#">org.astrogrid.community.beans.v1.Group</a>	A security group used in authorization.
<a href="#">org.astrogrid.community.User</a>	A bean to hold what is passed in the "community snippet".

## Representations of user and account details

### User

A bean to hold what is passed in the "community snippet". Note that the current interpretation of this is that the account contains an identifier of the form `user@community` - The group relates to the group that the user wishes to use for its credentials, this may be a cross community group. note that this should really be merged with the [@link org.astrogrid.community.common.util.CommunityMessage](#) class, which effectively does xml deserialization (should be deprecated From iteration 5 - use [@link org.astrogrid.community.beans.v1.Credentials](#), but may still be needed) more..

```

java.lang.String getAccount()
.....
java.lang.String getCommunity()
.....
java.lang.String getGroup()
.....
java.lang.String getToken()
.....
java.lang.String ()
.....
void setAccount(java.lang.String string)
.....
void setGroup(java.lang.String string)
.....
void setToken(java.lang.String string)
.....
java.lang.String toSnippet()
.....
java.lang.String toString()
.....

```

### Account

Extends [org.astrogrid.common.bean.BaseBean](#) , [org.astrogrid.community.beans.v1.BaseIdentifier](#)

The user account. more..

```

boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
org.astrogrid.community.beans.v1.Account unmarshalAccount(java.io.Reader reader)
.....

```

```
void validate()
```

---

## BaseIdentifier

Extends [org.astrogrid.common.bean.BaseBean](#)

The identifier base class. Contains the name and the community more..

```
java.lang.String getCommunity()
.....
java.lang.String getName()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
void setCommunity(java.lang.String community)
.....
void setName(java.lang.String name)
.....
org.astrogrid.community.beans.v1.BaseIdentifier unmarshalBaseIdentifier(java.io.Reader reader)
.....
void validate()
```

---

## Credentials

Extends [org.astrogrid.common.bean.BaseBean](#)

The full authorization and authentication credentials. more..

```
org.astrogrid.community.beans.v1.Account getAccount()
.....
org.astrogrid.community.beans.v1.Group getGroup()
.....
java.lang.String getSecurityToken()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
void setAccount( org.astrogrid.community.beans.v1.Account account)
.....
void setGroup( org.astrogrid.community.beans.v1.Group group)
.....
void setSecurityToken(java.lang.String securityToken)
.....
org.astrogrid.community.beans.v1.Credentials unmarshalCredentials(java.io.Reader reader)
.....
void validate()
```

---

## Group

Extends [org.astrogrid.common.bean.BaseBean](#) , [org.astrogrid.community.beans.v1.BaseIdentifier](#)

A security group used in authorization. more..

```
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
org.astrogrid.community.beans.v1.Group unmarshalGroup(java.io.Reader reader)
.....
void validate()
```

---

## Workflow Document Model

<a href="#">org.astrogrid.workflow.beans.v1.ActivityContainer</a>	Abstract base class of activities that contain other activities
<a href="#">org.astrogrid.workflow.beans.v1.Catch</a>	Action to take when an error occurs in the wrapped try block variable specified by 'var' attribute will contain details of the error.
<a href="#">org.astrogrid.workflow.beans.v1.Else</a>	Branch to take when if condition evaluates to false
<a href="#">org.astrogrid.workflow.beans.v1.execution.ExecutionRecordType</a>	basic type for execution records
<a href="#">org.astrogrid.workflow.beans.v1.execution.Extension</a>	A string 'buffer' for holding further information, keyed by attribute - so execution record becomes a map.
<a href="#">org.astrogrid.workflow.beans.v1.execution.JobExecutionRecord</a>	A record of a single execution of a job
<a href="#">org.astrogrid.workflow.beans.v1.execution.JobExecutionRecordType</a>	Class JobExecutionRecordType.
<a href="#">org.astrogrid.workflow.beans.v1.execution.JobURN</a>	unique identifier for job executions
<a href="#">org.astrogrid.workflow.beans.v1.execution.StepExecutionRecord</a>	A record of a single execution of a job step
<a href="#">org.astrogrid.workflow.beans.v1.execution.WorkflowSummaryType</a>	summary record for a single execution of a job
<a href="#">org.astrogrid.workflow.beans.v1.Flow</a>	a collection of activities to be performed potentially in parallel
<a href="#">org.astrogrid.workflow.beans.v1.For</a>	A for loop construct - iterated over a sequence of items
<a href="#">org.astrogrid.workflow.beans.v1.If</a>	An if statement
<a href="#">org.astrogrid.workflow.beans.v1.Input</a>	the list of input parameters
<a href="#">org.astrogrid.workflow.beans.v1.Output</a>	the list of output paramters
<a href="#">org.astrogrid.workflow.beans.v1.Parfor</a>	A parallel-for loop construct - executes the loop body on each element of items in parallel
<a href="#">org.astrogrid.workflow.beans.v1.Scope</a>	create a new workflow-variable scope for its body.
<a href="#">org.astrogrid.workflow.beans.v1.Script</a>	a step in the workflow - execute some script statements.
<a href="#">org.astrogrid.workflow.beans.v1.Sequence</a>	a collection of activities to be performed sequentially
<a href="#">org.astrogrid.workflow.beans.v1.Set</a>	Declare (and optionally initialize) a new workflow variable, or update value of existing variable
<a href="#">org.astrogrid.workflow.beans.v1.Step</a>	a step of the workflow - call to an external CEA application.

<a href="#">org.astrogrid.workflow.beans.v1.Then</a>	Branch to take when if condition evaluates to true
<a href="#">org.astrogrid.workflow.beans.v1.Tool</a>	represents a call to a CEA application
<a href="#">org.astrogrid.workflow.beans.v1.Try</a>	Error-handling construct.
<a href="#">org.astrogrid.workflow.beans.v1.types.JoinType</a>	
<a href="#">org.astrogrid.workflow.beans.v1.Unset</a>	forget a previously-declared workflow variable
<a href="#">org.astrogrid.workflow.beans.v1.While</a>	A while loop construct
<a href="#">org.astrogrid.workflow.beans.v1.Workflow</a>	Base element of an Astrogird workflow document, conforming to the schema defined for namespace <a href="http://www.astrogrid.org/schema/AGWorkflow/v1">http://www.astrogrid.org/schema/AGWorkflow/v1</a>

Workflow Documents are represented using the following objects

### AbstractActivity

Extends [org.astrogrid.common.bean.BaseBean](#)

The abstract base class of all activities that can be performed in a workflow more..

```

java.lang.String getId()
boolean isValid()
void marshal(java.io.Writer out)
void marshal(org.xml.sax.ContentHandler handler)
void setId(java.lang.String id)
org.astrogrid.workflow.beans.v1.AbstractActivity unmarshalAbstractActivity(java.io.Reader reader)
void validate()

```

### ActivityContainer

Extends [org.astrogrid.common.bean.BaseBean](#) , [org.astrogrid.workflow.beans.v1.AbstractActivity](#)

Abstract base class of activities that contain other activities more..

```

void addActivity( org.astrogrid.workflow.beans.v1.AbstractActivity vActivity)
void clearActivity()
java.util.Enumeration enumerateActivity()
boolean equals(java.lang.Object obj)
org.astrogrid.workflow.beans.v1.AbstractActivity[] getActivity()
org.astrogrid.workflow.beans.v1.AbstractActivity getActivity(int index)

```

```

int getActivityCount()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
boolean removeActivity( org.astrogrid.workflow.beans.v1.AbstractActivity vActivity)
.....
void setActivity(int index, org.astrogrid.workflow.beans.v1.AbstractActivity vActivity)
.....
void setActivity( org.astrogrid.workflow.beans.v1.AbstractActivity[] activityArray)
.....
org.astrogrid.workflow.beans.v1.ActivityContainer unmarshalActivityContainer(java.io.Reader reader)
.....
void validate()
.....

```

---

## Catch

Extends [org.astrogrid.common.bean.BaseBean](#)

Action to take when an error occurs in the wrapped try block variable specified by 'var' attribute will contain details of the error. more..

```

org.astrogrid.workflow.beans.v1.AbstractActivity getActivity()
.....
java.lang.String getVar()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
void setActivity( org.astrogrid.workflow.beans.v1.AbstractActivity activity)
.....
void setVar(java.lang.String var)
.....
org.astrogrid.workflow.beans.v1.Catch unmarshalCatch(java.io.Reader reader)
.....
void validate()
.....

```

---

## Else

Extends [org.astrogrid.common.bean.BaseBean](#)

Branch to take when if condition evaluates to false more..

```

org.astrogrid.workflow.beans.v1.AbstractActivity getActivity()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....

```

```

void setActivity( org.astrogrid.workflow.beans.v1.AbstractActivity activity)
.....
org.astrogrid.workflow.beans.v1.Else unmarshalElse(java.io.Reader reader)
.....
void validate()

```

---

## Flow

Extends [org.astrogrid.common.bean.BaseBean](#) , [org.astrogrid.workflow.beans.v1.AbstractActivity](#) , [org.astrogrid.workflow.beans.v1.ActivityContainer](#)

a collection of activities to be performed potentially in parallel more..

```

boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
org.astrogrid.workflow.beans.v1.Flow unmarshalFlow(java.io.Reader reader)
.....
void validate()

```

---

## For

Extends [org.astrogrid.common.bean.BaseBean](#) , [org.astrogrid.workflow.beans.v1.AbstractActivity](#)

A for loop construct - iterated over a sequence of items more..

```

org.astrogrid.workflow.beans.v1.AbstractActivity getActivity()
.....
java.lang.String getItems()
.....
java.lang.String getVar()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
void setActivity( org.astrogrid.workflow.beans.v1.AbstractActivity activity)
.....
void setItems(java.lang.String items)
.....
void setVar(java.lang.String var)
.....
org.astrogrid.workflow.beans.v1.For unmarshalFor(java.io.Reader reader)
.....
void validate()

```

---

## If

Extends [org.astrogrid.common.bean.BaseBean](#) , [org.astrogrid.workflow.beans.v1.AbstractActivity](#)

An if statement more..

```

org.astrogrid.workflow.beans.v1.Else  getElse()
-----
java.lang.String  getTest()
-----
org.astrogrid.workflow.beans.v1.Then  getThen()
-----
boolean  isValid()
-----
void  marshal(java.io.Writer out)
-----
void  marshal(org.xml.sax.ContentHandler handler)
-----
void  setElse( org.astrogrid.workflow.beans.v1.Else  _else)
-----
void  setTest(java.lang.String test)
-----
void  setThen( org.astrogrid.workflow.beans.v1.Then  then)
-----
org.astrogrid.workflow.beans.v1.If  unmarshalIf(java.io.Reader reader)
-----
void  validate()
-----

```

## Input

Extends [org.astrogrid.common.bean.BaseBean](#)

the list of input parameters more..

```

void  addParameter( org.astrogrid.applications.beans.v1.parameters.ParameterValue  vParameter)
-----
void  clearParameter()
-----
java.util.Enumeration  enumerateParameter()
-----
boolean  equals(java.lang.Object obj)
-----
org.astrogrid.applications.beans.v1.parameters.ParameterValue[]  getParameter()
-----
org.astrogrid.applications.beans.v1.parameters.ParameterValue  getParameter(int index)
-----
int  getParameterCount()
-----
boolean  isValid()
-----
void  marshal(java.io.Writer out)
-----
void  marshal(org.xml.sax.ContentHandler handler)
-----
boolean  removeParameter( org.astrogrid.applications.beans.v1.parameters.ParameterValue  vParameter)
-----
void  setParameter(int index, org.astrogrid.applications.beans.v1.parameters.ParameterValue  vParameter)
-----
void  setParameter( org.astrogrid.applications.beans.v1.parameters.ParameterValue[]  parameterArray)
-----
org.astrogrid.workflow.beans.v1.Input  unmarshalInput(java.io.Reader reader)
-----
void  validate()
-----

```

## Output

Extends [org.astrogrid.common.bean.BaseBean](#)

the list of output paramters more..

```

void addParameter( org.astrogrid.applications.beans.v1.parameters.ParameterValue vParameter)
.....
void clearParameter()
.....
java.util.Enumeration enumerateParameter()
.....
boolean equals(java.lang.Object obj)
.....
org.astrogrid.applications.beans.v1.parameters.ParameterValue[] getParameter()
.....
org.astrogrid.applications.beans.v1.parameters.ParameterValue getParameter(int index)
.....
int getParameterCount()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
boolean removeParameter( org.astrogrid.applications.beans.v1.parameters.ParameterValue vParameter)
.....
void setParameter(int index, org.astrogrid.applications.beans.v1.parameters.ParameterValue vParameter)
.....
void setParameter( org.astrogrid.applications.beans.v1.parameters.ParameterValue[] parameterArray)
.....
org.astrogrid.workflow.beans.v1.Output unmarshalOutput(java.io.Reader reader)
.....
void validate()
.....

```

---

## Parfor

Extends [org.astrogrid.common.bean.BaseBean](#) , [org.astrogrid.workflow.beans.v1.AbstractActivity](#)

A parallel-for loop construct - executes the loop body on each element of items in parallel more..

```

org.astrogrid.workflow.beans.v1.AbstractActivity getActivity()
.....
java.lang.String getItems()
.....
java.lang.String getVar()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
void setActivity( org.astrogrid.workflow.beans.v1.AbstractActivity activity)
.....
void setItems(java.lang.String items)
.....
void setVar(java.lang.String var)
.....

```

```

org.astrogrid.workflow.beans.v1.Parfor unmarshalParfor(java.io.Reader reader)
.....
void validate()

```

---

## Scope

Extends [org.astrogrid.common.bean.BaseBean](#) , [org.astrogrid.workflow.beans.v1.AbstractActivity](#)

create a new workflow-variable scope for its body. any variables defined within its body will no longer be in scope after the end tag. more..

```

org.astrogrid.workflow.beans.v1.AbstractActivity getActivity()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
void setActivity( org.astrogrid.workflow.beans.v1.AbstractActivity activity)
.....
org.astrogrid.workflow.beans.v1.Scope unmarshalScope(java.io.Reader reader)
.....
void validate()

```

---

## Script

Extends [org.astrogrid.common.bean.BaseBean](#) , [org.astrogrid.workflow.beans.v1.AbstractActivity](#)

a step in the workflow - execute some script statements. more..

```

void addStepExecutionRecord( org.astrogrid.workflow.beans.v1.execution.StepExecutionRecord
vStepExecutionRecord)
.....
void clearStepExecutionRecord()
.....
java.util.Enumeration enumerateStepExecutionRecord()
.....
boolean equals(java.lang.Object obj)
.....
java.lang.String getBody()
.....
java.lang.String getDescription()
.....
org.astrogrid.workflow.beans.v1.execution.StepExecutionRecord[] getStepExecutionRecord()
.....
org.astrogrid.workflow.beans.v1.execution.StepExecutionRecord getStepExecutionRecord(int index)
.....
int getStepExecutionRecordCount()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....

```

---

```

boolean removeStepExecutionRecord( org.astrogrid.workflow.beans.v1.execution.StepExecutionRecord
    vStepExecutionRecord)
.....
void setBody(java.lang.String body)
.....
void setDescription(java.lang.String description)
.....
void setStepExecutionRecord(int index, org.astrogrid.workflow.beans.v1.execution.StepExecutionRecord
    vStepExecutionRecord)
.....
void setStepExecutionRecord( org.astrogrid.workflow.beans.v1.execution.StepExecutionRecord[]
    stepExecutionRecordArray)
.....
org.astrogrid.workflow.beans.v1.Script unmarshalScript(java.io.Reader reader)
.....
void validate()

```

---

## Sequence

Extends [org.astrogrid.common.bean.BaseBean](#) , [org.astrogrid.workflow.beans.v1.AbstractActivity](#) ,  
[org.astrogrid.workflow.beans.v1.ActivityContainer](#)

a collection of activities to be performed sequentially more..

```

boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
org.astrogrid.workflow.beans.v1.Sequence unmarshalSequence(java.io.Reader reader)
.....
void validate()

```

---

## Set

Extends [org.astrogrid.common.bean.BaseBean](#) , [org.astrogrid.workflow.beans.v1.AbstractActivity](#)

Declare (and optionally initialize) a new workflow variable, or update value of existing variable more..

```

java.lang.String getValue()
.....
java.lang.String getVar()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
void setValue(java.lang.String value)
.....
void setVar(java.lang.String var)
.....
org.astrogrid.workflow.beans.v1.Set unmarshalSet(java.io.Reader reader)
.....

```

```
void validate()
```

---

## Step

Extends [org.astrogrid.common.bean.BaseBean](#) , [org.astrogrid.workflow.beans.v1.AbstractActivity](#)

a step of the workflow - call to an external CEA application. more..

```
void addStepExecutionRecord( org.astrogrid.workflow.beans.v1.execution.StepExecutionRecord
                             vStepExecutionRecord)
.....
void clearStepExecutionRecord()
.....
void deleteSequenceNumber()
.....
void deleteStepNumber()
.....
java.util.Enumeration enumerateStepExecutionRecord()
.....
boolean equals(java.lang.Object obj)
.....
java.lang.String getDescription()
.....
org.astrogrid.workflow.beans.v1.types.JoinType getJoinCondition()
.....
java.lang.String getName()
.....
java.lang.String getResultVar()
.....
int getSequenceNumber()
.....
org.astrogrid.workflow.beans.v1.execution.StepExecutionRecord[] getStepExecutionRecord()
.....
org.astrogrid.workflow.beans.v1.execution.StepExecutionRecord getStepExecutionRecord(int index)
.....
int getStepExecutionRecordCount()
.....
int getStepNumber()
.....
org.astrogrid.workflow.beans.v1.Tool getTool()
.....
boolean hasSequenceNumber()
.....
boolean hasStepNumber()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
boolean removeStepExecutionRecord( org.astrogrid.workflow.beans.v1.execution.StepExecutionRecord
                                   vStepExecutionRecord)
.....
void setDescription(java.lang.String description)
.....
void setJoinCondition( org.astrogrid.workflow.beans.v1.types.JoinType joinCondition)
.....
void setName(java.lang.String name)
.....
void setResultVar(java.lang.String resultVar)
.....
```

```

void setSequenceNumber(int sequenceNumber)
.....
void setStepExecutionRecord(int index, org.astrogrid.workflow.beans.v1.execution.StepExecutionRecord
vStepExecutionRecord)
.....
void setStepExecutionRecord( org.astrogrid.workflow.beans.v1.execution.StepExecutionRecord[]
stepExecutionRecordArray)
.....
void setStepNumber(int stepNumber)
.....
void setTool( org.astrogrid.workflow.beans.v1.Tool tool)
.....
org.astrogrid.workflow.beans.v1.Step unmarshalStep(java.io.Reader reader)
.....
void validate()

```

---

## Then

Extends [org.astrogrid.common.bean.BaseBean](#)

Branch to take when if condition evaluates to true more..

```

org.astrogrid.workflow.beans.v1.AbstractActivity getActivity()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
void setActivity( org.astrogrid.workflow.beans.v1.AbstractActivity activity)
.....
org.astrogrid.workflow.beans.v1.Then unmarshalThen(java.io.Reader reader)
.....
void validate()

```

---

## Tool

Extends [org.astrogrid.common.bean.BaseBean](#)

represents a call to a CEA application more..

```

org.astrogrid.workflow.beans.v1.Input getInput()
.....
java.lang.String getInterface()
.....
java.lang.String getName()
.....
org.astrogrid.workflow.beans.v1.Output getOutput()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
void setInput( org.astrogrid.workflow.beans.v1.Input input)
.....

```

```

void setInterface(java.lang.String _interface)
.....
void setName(java.lang.String name)
.....
void setOutput( org.astrogrid.workflow.beans.v1.Output output)
.....
org.astrogrid.workflow.beans.v1.Tool unmarshalTool(java.io.Reader reader)
.....
void validate()

```

---

## Try

Extends [org.astrogrid.common.bean.BaseBean](#) , [org.astrogrid.workflow.beans.v1.AbstractActivity](#)

Error-handling construct. if an error occurs during execution of the wrapped activity, the activity in the 'catch' block is executed. more..

```

org.astrogrid.workflow.beans.v1.AbstractActivity getActivity()
.....
org.astrogrid.workflow.beans.v1.Catch getCatch()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
void setActivity( org.astrogrid.workflow.beans.v1.AbstractActivity activity)
.....
void setCatch( org.astrogrid.workflow.beans.v1.Catch _catch)
.....
org.astrogrid.workflow.beans.v1.Try unmarshalTry(java.io.Reader reader)
.....
void validate()

```

---

## Unset

Extends [org.astrogrid.common.bean.BaseBean](#) , [org.astrogrid.workflow.beans.v1.AbstractActivity](#)

forget a previously-declared workflow variable more..

```

java.lang.String getVar()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
void setVar(java.lang.String var)
.....
org.astrogrid.workflow.beans.v1.Unset unmarshalUnset(java.io.Reader reader)
.....
void validate()

```

---

## While

Extends [org.astrogrid.common.bean.BaseBean](#) , [org.astrogrid.workflow.beans.v1.AbstractActivity](#)

A while loop construct more..

```

org.astrogrid.workflow.beans.v1.AbstractActivity getActivity()
.....
java.lang.String getTest()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
void setActivity( org.astrogrid.workflow.beans.v1.AbstractActivity activity)
.....
void setTest(java.lang.String test)
.....
org.astrogrid.workflow.beans.v1.While unmarshalWhile(java.io.Reader reader)
.....
void validate()
.....

```

---

## Workflow

Extends [org.astrogrid.common.bean.BaseBean](#)

Base element of an Astrogrid workflow document, conforming to the schema defined for namespace <http://www.astrogrid.org/schema/AGWorkflow/v1> more..

```

org.astrogrid.community.beans.v1.Credentials getCredentials()
.....
java.lang.String getDescription()
.....
java.lang.String getId()
.....
org.astrogrid.workflow.beans.v1.execution.JobExecutionRecord getJobExecutionRecord()
.....
java.lang.String getName()
.....
org.astrogrid.workflow.beans.v1.Sequence getSequence()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
void setCredentials( org.astrogrid.community.beans.v1.Credentials credentials)
.....
void setDescription(java.lang.String description)
.....
void setId(java.lang.String id)
.....
void setJobExecutionRecord( org.astrogrid.workflow.beans.v1.execution.JobExecutionRecord
jobExecutionRecord)
.....

```

```

void setName(java.lang.String name)
.....
void setSequence( org.astrogrid.workflow.beans.v1.Sequence sequence)
.....
org.astrogrid.workflow.beans.v1.Workflow unmarshalWorkflow(java.io.Reader reader)
.....
void validate()
.....

```

---

## ExecutionRecordType

Extends [org.astrogrid.common.bean.BaseBean](#)

basic type for execution records more..

```

void addExtension( org.astrogrid.workflow.beans.v1.execution.Extension vExtension)
.....
void addMessage(int index, org.astrogrid.applications.beans.v1.cea.castor.MessageType vMessage)
.....
void addMessage( org.astrogrid.applications.beans.v1.cea.castor.MessageType vMessage)
.....
void clearExtension()
.....
void clearMessage()
.....
java.util.Enumeration enumerateExtension()
.....
java.util.Enumeration enumerateMessage()
.....
boolean equals(java.lang.Object obj)
.....
org.astrogrid.workflow.beans.v1.execution.Extension[] getExtension()
.....
org.astrogrid.workflow.beans.v1.execution.Extension getExtension(int index)
.....
int getExtensionCount()
.....
java.util.Date getFinishTime()
.....
org.astrogrid.applications.beans.v1.cea.castor.MessageType[] getMessage()
.....
org.astrogrid.applications.beans.v1.cea.castor.MessageType getMessage(int index)
.....
int getMessageCount()
.....
java.util.Date getStartTime()
.....
org.astrogrid.applications.beans.v1.cea.castor.types.ExecutionPhase getStatus()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
boolean removeExtension( org.astrogrid.workflow.beans.v1.execution.Extension vExtension)
.....
boolean removeMessage( org.astrogrid.applications.beans.v1.cea.castor.MessageType vMessage)
.....
void setExtension(int index, org.astrogrid.workflow.beans.v1.execution.Extension vExtension)
.....
void setExtension( org.astrogrid.workflow.beans.v1.execution.Extension[] extensionArray)
.....

```

```

void setFinishTime(java.util.Date finishTime)
.....
void setMessage(int index, org.astrogrid.applications.beans.v1.cea.castor.MessageType vMessage)
.....
void setMessage( org.astrogrid.applications.beans.v1.cea.castor.MessageType\[\] messageArray)
.....
void setStartTime(java.util.Date startTime)
.....
void setStatus( org.astrogrid.applications.beans.v1.cea.castor.types.ExecutionPhase status)
.....
org.astrogrid.workflow.beans.v1.execution.ExecutionRecordType
unmarshalExecutionRecordType(java.io.Reader reader)
.....
void validate()

```

---

## Extension

Extends [org.astrogrid.common.bean.BaseBean](#)

A string 'buffer' for holding further information, keyed by attribute - so execution record becomes a map.  
more..

```

java.lang.String getContent()
.....
java.lang.String getKey()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
void setContent(java.lang.String content)
.....
void setKey(java.lang.String key)
.....
org.astrogrid.workflow.beans.v1.execution.Extension unmarshalExtension(java.io.Reader reader)
.....
void validate()

```

---

## JobExecutionRecord

Extends [org.astrogrid.common.bean.BaseBean](#) ,  
[org.astrogrid.workflow.beans.v1.execution.ExecutionRecordType](#) ,  
[org.astrogrid.workflow.beans.v1.execution.JobExecutionRecordType](#)

A record of a single execution of a job more..

```

boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....

```

```

org.astrogrid.workflow.beans.v1.execution.JobExecutionRecord
  unmarshalJobExecutionRecord(java.io.Reader reader)
.....
void validate()

```

---

### JobExecutionRecordType

Extends [org.astrogrid.common.bean.BaseBean](#) ,  
[org.astrogrid.workflow.beans.v1.execution.ExecutionRecordType](#)

Class JobExecutionRecordType. more..

```

org.astrogrid.workflow.beans.v1.execution.JobURN  getJobId()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
void setJobId( org.astrogrid.workflow.beans.v1.execution.JobURN  jobId)
.....
org.astrogrid.workflow.beans.v1.execution.JobExecutionRecordType
  unmarshalJobExecutionRecordType(java.io.Reader reader)
.....
void validate()

```

---

### JobURN

Extends [org.astrogrid.common.bean.BaseBean](#)

unique identifier for job executions more..

```

java.lang.String getContent()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
void setContent(java.lang.String content)
.....
org.astrogrid.workflow.beans.v1.execution.JobURN  unmarshalJobURN(java.io.Reader reader)
.....
void validate()

```

---

### StepExecutionRecord

Extends [org.astrogrid.common.bean.BaseBean](#) ,  
[org.astrogrid.workflow.beans.v1.execution.ExecutionRecordType](#)

A record of a single execution of a job step more..

```

boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
org.astrogrid.workflow.beans.v1.execution.StepExecutionRecord
  unmarshalStepExecutionRecord(java.io.Reader reader)
.....
void validate()

```

---

## WorkflowSummaryType

Extends [org.astrogrid.common.bean.BaseBean](#) ,  
[org.astrogrid.workflow.beans.v1.execution.ExecutionRecordType](#) ,  
[org.astrogrid.workflow.beans.v1.execution.JobExecutionRecordType](#)

summary record for a single execution of a job more..

```

java.lang.String getDescription()
.....
java.lang.String getWorkflowName()
.....
boolean isValid()
.....
void marshal(java.io.Writer out)
.....
void marshal(org.xml.sax.ContentHandler handler)
.....
void setDescription(java.lang.String description)
.....
void setWorkflowName(java.lang.String workflowName)
.....
org.astrogrid.workflow.beans.v1.execution.WorkflowSummaryType
  unmarshalWorkflowSummaryType(java.io.Reader reader)
.....
void validate()

```

---

## JoinType

more..

```

int getType()
.....
java.lang.String toString()
.....
org.astrogrid.workflow.beans.v1.types.JoinType valueOf(java.lang.String string)

```

---

## 2.4 Execution Results

---

### Running a workflow

This section walks through the process of executing a workflow. It assumes you've already got a workflow document written.

### Submitting a Workflow

Workflows have to be submitted to a JES server to be executed. This can be done:

- via the portal
- via the JES servers html interface (e.g. see <http://twmbarlwm.astrogrid.org:8888/astrogrid-jes-SNAPSHOT/query-form.html> )
- programmatically using the Jes Delegate (javadoc: <http://www.astrogrid.org/maven/docs/HEAD/jes/apidocs/org/astrogrid/jes/delegate/package-frame.html> )

### Tracking Progress

Once a workflow has been submitted to Jes, a unique identifier for the job is assigned. This identifier (*JobURN*) can then be used to key the Jes Server and retrieve information about the workflow document.

Alternatively, a Jes Server can be queried for the list of jobs that were submitted to it by a particular user.

Both functions are available via the portal, the JES html interface, and programmatically through delegates. The portal returns an interactive display of a workflow, the html interface returns a html-formatted representation of workflow, while the delegates return the raw xml document.

### Examples

- [HTML Formatted report](#)
- [XML Formatted report](#)

### Inspecting Results

During execution of a job, the Jes Server records progress and result information as annotations to the submitted workflow document. The following annotation elements are used to record execution information.

### Step Execution Record

The `<step-execution-record>` element records a single execution of a `step` or `script` element. A `step` or `script` may be annotated with an unbounded number of `step-execution-records` - one for each time the activity is executed. (its possible for an activity to be executed more than once, due to loops).

A `step-execution-record` records the time the execution started and finished and the current execution phase (one of `PENDING`, `RUNNING`, `COMPLETED`, `ERROR`).

A `step-execution-record` element may also contain a number of `message` elements, which provide more information on the execution of the activity.

It may also contain a number of `extension` elements - this provides an extension point of the workflow document schema where implementation data / further functionality can be stored. This possibilities of this extension point, and its use in the current implementation of JES need to be documented, but it's not important from a user point-of-view

### **Job Execution Record**

A `<job-execution-record>` element records the execution of the entire workflow. The root workflow element may have a single `job-execution-record`.

This element has the same structure as `step-execution-record` but also has an additional child element - `<jobId>` which records the unique identifier (obURN) for this job execution.

### **Message**

A `<message>` element records a message generated during execution. It may have been generated by the `JesServer`, a `CEA` server that was called to execute a tool, or via a script call to one of the `jes` logging functions. Each message provides:

- `content` the text of the message
- `timestamp`
- `level` severity / importance of the message
- `source` originator of the message
- `phase` current execution phase when message was generated.

## 2.5 Document Schema

---

### Workflow Document Schema

- [HTML Formatted](#)
- [XSD Schema](#)

## 3.1 Simple Examples

---

### Introduction

## 3.2 Scripted Workflows

---

### Introduction

Will have some whole workflow documents here, showing how scripting features can be integrated into workflows.

### Defining Constants

### Control flow based on results of previous step

### Iteration with variable tool parameters

### Parallel iteration with variables

### Self-Modifying Workflows

workflow scripts may add dynamically parameter values to tool elements

```
// access result of previous step
votable = source.result;
//create new parser
parser = new XmlParser();
//parse votable into node tree
nodes = parser.parseText(votable);
// filter node tree on 'TD', extract value attribute
urls = nodes.depthFirst().findAll{it.name() ==
'TD'}.collect{it.value()}.flatten();
// show what we've found
print(urls);
// find next step in workflow
sinkStep = jes.getSteps().find {it.name == 'sink-step'};
// get to set of input parameters
inputs = sinkStep.tool.input;
// for each url, create new parameter, populate, add to inputs.
urls.each { p = jes.newParameter(name:'url', value:it); inputs.addParameter(p);}
```

## 3.3 JEScript Recipes

---

### Introduction

This document describes snippets of JEScript for performing common tasks.

### Working with tables.

JEScript integrates the [STIL library](#) for reading and writing astronomical tables in many common formats.

#### Construct a table from contents of a String

the following example constructs a star table from data held in a string `votableString`. If possible, the optional second parameter should be used to tell the library what format table to expect

```
table = astrogrid.tableHelper.builder.makeStarTableFromString(votableString,
'votable' )
```

#### Construct a table from contents of a URI

The following example constructs a star table from data held at a remote location, pointed to by a string `uri`. The following example will work for references to `myspace`, `http`, `file` and `ftp` - ie for `uri`'s starting with `ivo://,file://,http://,ftp://`

```
ev = astrogrid.ioHelper.getExternalValue(uri)
table = astrogrid.tableHelper.builder.makeStarTable(ev)
```

There are also variants of `makeStarTable()` that accept a URL or String directly - but these will only work for conventional `http://` and `ftp://` references.

#### Write out a table

Contents of a table object can be written out to a `uri` location (for `uri`'s starting with `ivo://,file://,http://,ftp://`) as follows

```

ev = astrogrid.ioHelper.getExternalValue(uri);
astrogrid.tableHelper.writeTable(ev,table,'votable')
// and you can check it's there be doing something like..
table1 = astrogrid.tableHelper.builder.makeStarTable(ev)
assert table.rowCount == table1.rowCount
assert table.columnCount == table1.columnCount

```

### Processing the rows of a table

As well as the row and cell access methods defined by the classes of the STIL library, JEScript also provides iterators over the rows and cells of a table. This allows Groovy's list-processing features to be used to filter and process table data

The following sums all the cells in column 3 of the table

```

colSum = table.columnIterator(3).inject(0) {acc, item | acc + item}

```

The following example sums each row of a table, returning a list of sums

```

sumList = table.iterator().collect{ it.inject(0) {acc, item | acc + item}}

```

The following returns all values of column 5 in rows where column 3 is less than column 4

```

vals = table.iterator().findAll{ it[3] < it[4]}.collect{it[5]}

```

### Extract a column by UCD

The following example parses the metadata of a votable, to identify which column has the UCD `VOX:Image_AccessReference`. It then extracts the contents of that column of the table into a list. Most of the methods used in this example are part of the STIL table library

The script assumes that a `table` object already exists.

Results is a list of strings - which in this example happen to be urls

```

col = ( 0 ... table.columnCount ).find{ table.getColumnInfo( it ).getUCD() ==
'VOX:Image_AccessReference' }
urls = table.columnIterator(col).collect{it}

```

This example uses the internal-iterator style of Groovy programming to achieve very concise code. The equivalent using external looping would be

```
col = 0
for (x in 0 .. table.columnCount) {
    if (table.getColumnInfo(x).getUCD() == 'VOX:Image_AccessReference') {
        col = x
    }
}
urls = []
for (x in table.columnIterator(col)) {
    urls.add(x);
}
```

### Extract a column by name

Variation of the previous example - look for a column based on its name

```
col = ( 0 .. table.columnCount ).find{ table.getColumnInfo( it ).name == 'dec' }
decs= table.columnIterator(col).collect{it}
```

### Removing a column from a table

To define a new table that is the same as an existing table `t`, but with column 3 removed, say:

```
newTable = t.removeColumn(3)
```

### Adding a constant column to a table

To define a new table, based on an existing table, but with a new column of a constant value '5'

```
meta = astrogrid.tableHelper.newColumnInfo("column-name")
newTable = table.addColumn(meta,5);
```

### Adding a computed column to a table

To define a new table, based on an existing table, but with a new column whose values are computed from other cells in the row:

```
meta = astrogrid.tableHelper.newColumnInfo("column-name")
newTable = table.addColumn(info,{ it[1] * 3 + it[4]})
```

To define a compute column, the value passed in is a groovy closure. This will be passed a list containing the values of the other cells in the row - the value it returns is used as the value for the computed cell

### Building a table from scratch

If you need to build up a table bit by bit, it's best to create a mutable table, as follows

```
cols = [
    astrogrid.tableHelper.newColumnInfo("ra"),
    astrogrid.tableHelper.newColumnInfo("dec"),
    astrogrid.tableHelper.newColumnInfo("radius")
]
mutableTable = astrogrid.tableHelper.newMutableTable(cols);
```

The mutable table provides methods for adding rows, and setting cell values

It's also possible to create a mutable table that has the same structure as an existing table (although it will not contain the table's data)

```
mutableTable = astrogrid.tableHelper.newMutableTableFromTemplate(table)
```

### Fine-grain table modification

If you need to do table manipulation that can't be expressed as new columns, then it is possible to create a mutable copy of the table, as follows. The copy have the same structure and contain the same data as the original

```
mutableCopy = table.asMutableTable()
```

## Working with VOspace

### Using VOspaceClient

The following example creates a new date-stamped folder in vospace, and then loads the contents of a URL into a vospace file in this new directory.

```
vospace = astrogrid.createVoSpaceClient(user)
newDirIvorn = astrogrid.objectBuilder.newIvorn(homeIvorn, "example-" +
astrogrid.ioHelper.dateStamp())
vospace.newFolder(newDirIvorn)
target = astrogrid.objectBuilder.newIvorn(newDirIvorn, 'slashdot.html')
vospace.putUrl('http://www.slashdot.org', target, false)
```

The first line creates a new vospace client, which will act under the permissions of the current user - user is a predefined JEScript object that represents the owner of the current workflow.

An ivorn pointing to a new directory is then created. This is relative to the homeIvorn, which is another predefined JEScript object that is the Ivorn of the homespace of the owner of the current workflow

The new folder is created, and then a file ivorn is computed relative to this. Finally, the vospace client is told to fetch the contents of a url and save it to this target ivorn.

## System Information

### Check server version

```
try {
  jes.info("Scripting version : " + astrogrid.version)
} catch (Exception e) {
  jes.warn(e)
}
try {
  jes.info("JES version: " + jes.version)
} catch (Exception e) {
  jes.warn(e)
}
```

This script will send two messages, with version info for the scripting library, and then the jes server. The version information is at present a space-separated list of bugzilla numbers that this component implements. The version can be checked programmatically by searching for a particular substring with these version reports.

## Groovy Programming

This section gives examples of how to do common things in groovy

### Manipulating Dates

```
import java.util.Calendar
interval = ['start':'2002-03-14 01:38:00' , 'end': '2002-03-14 01:45:00' ]
cal = Calendar.getInstance()
df = new java.text.SimpleDateFormat('yyyy-MM-dd HH:mm:ss')
cal.setTime(df.parse(interval.start))
cal.set(Calendar.MINUTE,0)
refstarttime = df.format(cal.time)

cal.setTime(df.parse(interval.end))
cal.set(Calendar.MINUTE,0)
cal.add(Calendar.HOUR,1)
refendtime = df.format(cal.time)
```

This script shows how to parse a string into a date object, alter values (in this case round up and down to the hour), and then write out to another string. Time calculations should be done with care, using the appropriate classes, rather than straight string manipulation - otherwise periods that cross midnight will fail.

Date patterns are documented at: <http://java.sun.com/j2se/1.4.2/docs/api/java/text/DateFormat.html>

## Templates

```
import groovy.text.SimpleTemplateEngine
// set up the query template - although this could also be read from a file
queryTemplate = 'query with ${var1} < ${var2} \nwhere ${foo}'
// set up values to substitute
binding = ['var1':'23', 'var2':'RA', 'foo':'x = 3']
// create template engine
engine = new SimpleTemplateEngine()
template = engine.createTemplate(queryTemplate)
query = template.make(binding)
```

The above example illustrates how to construct templates, and then how to apply a template to a set of bindings.