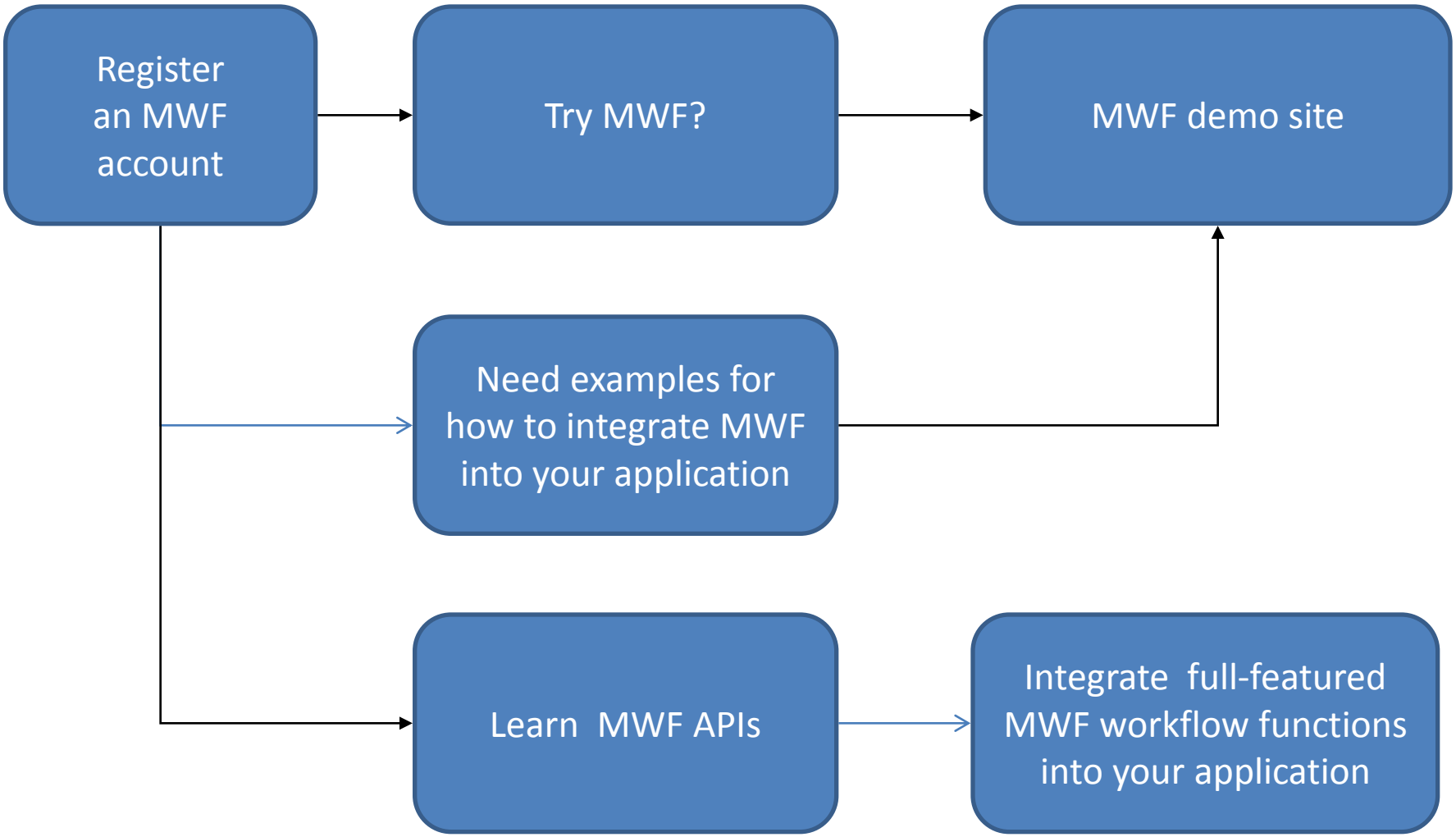
My World Flow

A Workflow Engine

What's MWF

- My World Flow, a workflow engine run as a service in the cloud.
- MWF is a workflow engine
  - that can be used remotely to drive either human-oriented workflow or computer-oriented business process, also in a hybrid mode.
- MWF runs in the cloud
  - no installation is required. Business sensitive data is kept on your side while MWF take care of driving your workflow processes.
- MWF provides Restful API
  - and GUI workflow designer as well as process monitor to programmers who can integrate advanced workflow features in your application with any modern computer language.
- MWF has workflow designer, monitor, full set Restful APIs.
- MWF support very complex workflow logic: sequence, parallel, AND, OR, script, sub-process, vote, timer etc.

How to start


Register an MWF account
Try MWF?
MWF demo site
Need examples for how to integrate MWF into your application
Learn MWF APIs
Integrate full-featured MWF workflow functions into your application

Register MWF account

- Access http://www.myworldflow.com/cflow/reg.jsp

MWF APIs

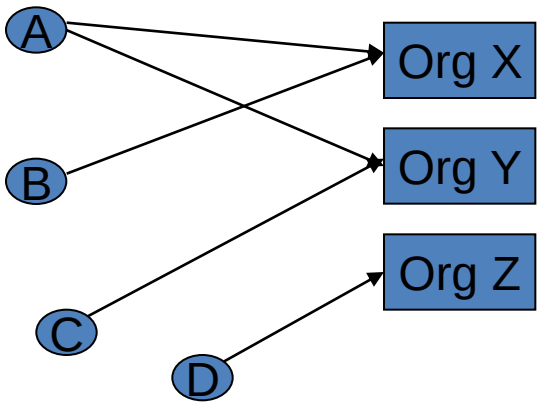
- Restful API:
  - http://www.myworldflow.com/cflow/tutorial/mwf_api.html
  - With Restful API, you can integrate MWF into your programs with any modern languages.
- JAVA SDK:
  - http://www.myworldflow.com/cflow/tutorial/sdk.html

Basic Concept: User Account

- User who participates in workflow collaboration should have an MWF account
- One can register MWF account on MWF website.
- A registered user can create many other user accounts with MWF API.

Basic Concept: Organization

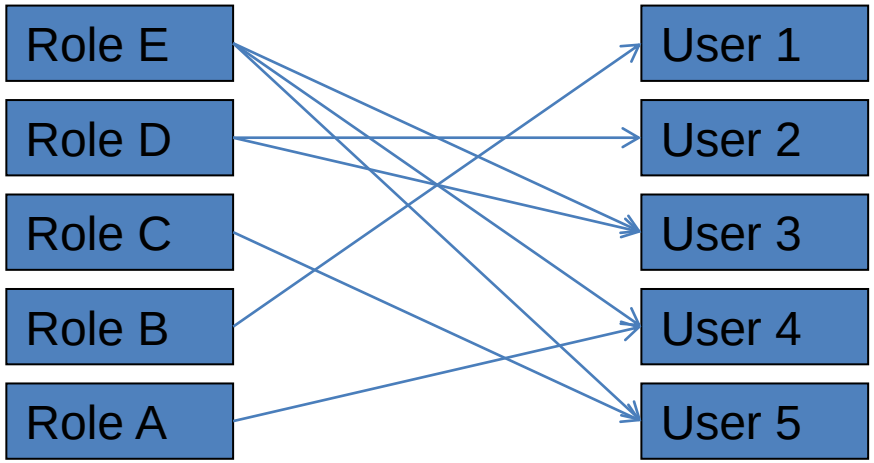
- Organization is a business boundary.
- A user can assign tasks to other users within same organization.
- A user can team-up with other users within same organization.



A can team-up with B since they both belongs to Org X
A can team-up with C since they both belongs to Org Y
C can NOT team-up with D since they don't belong to a common organization

Basic Concept: Team

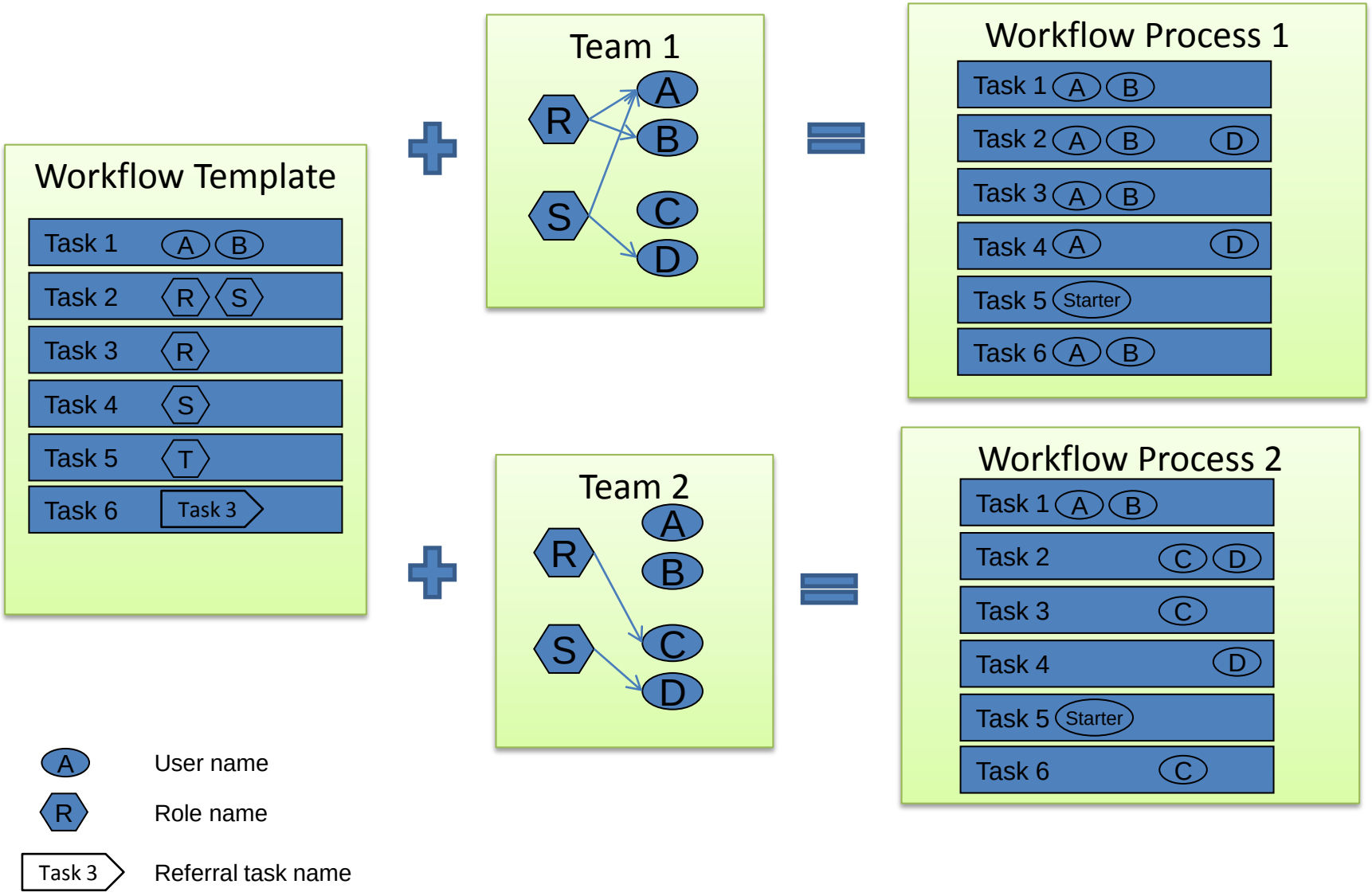
- Team defines role-user map
- Within one team:
  - A role can map to many users
  - A user can take many roles



Basic Concept: Task assignation

- In workflow template, a task can be assigned to one to many users, or one to many roles.
- if a task is assigned to role, and if a team is specified when a workflow is started, then the real task assignation will be determined by role-user map definition is the team.

Role-User Mapping



Workflow Designer

- Use MWF demo site's workflow designer
  - After login, click menu "Template Library" then click "Create a new workflow template"
- Or copy MWF demo site's workflow designer HTML source to your own web server, then modify as necessary.

Workflow Designer: UI

Toolbar

Canvas

Default template loaded, modify it as required.

Status

Workflow Designer: Toolbar

Delete: click a node or a link to delete it

Ground: complete one branch

Round: mark to rewind back to any previous node

OR: pass when any one of previous tasks is completed

AND: pass only when all previous tasks are completed

Script: invoke Javascript, JAVA or remote web services

Sub: include a sub-process

Timer: a timeout timer, process will hold untile timer expired.

Task: a task should be done by people

Link: click first node then click next node to link them up, a link means a workflow routine.

Select: click to select a node or a link

Workflow Designer: Toolbar

Workflow Name: the name of the current template

MAX: maximize the designer window

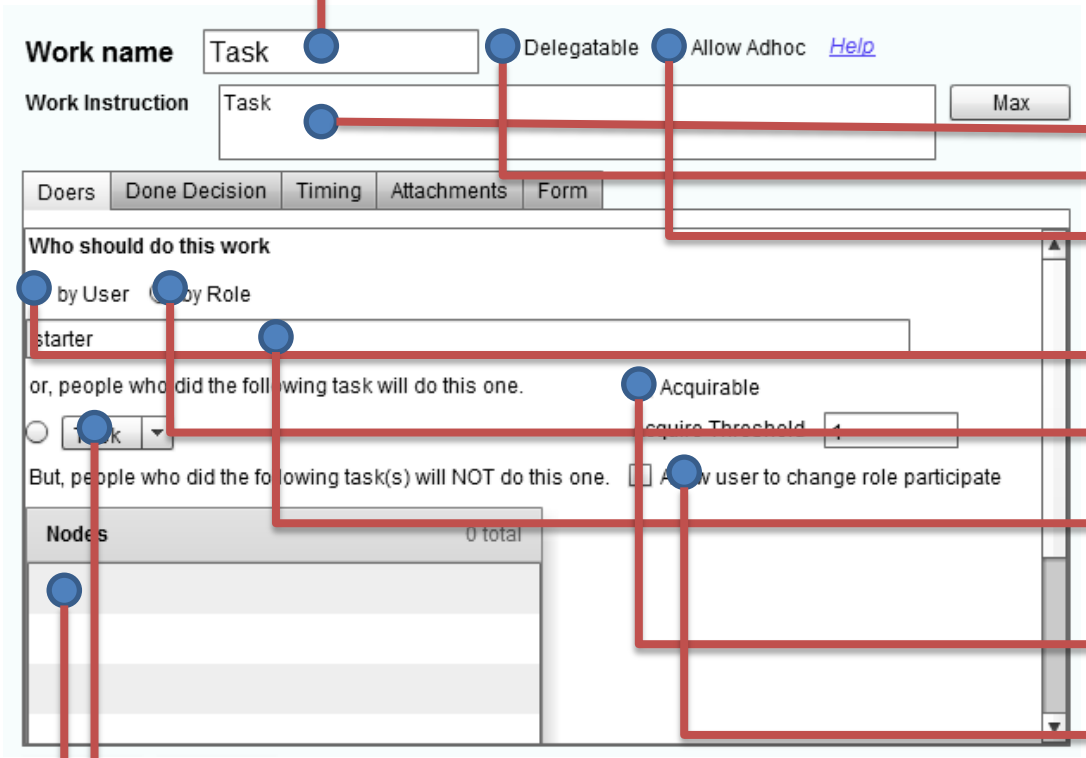
Save: save current workflow template

New: create a new workflow template

Workflow Designer: Link

Click on one node (the preceding task)
Click on another node (the following task),
An arrow will be draw between these two nodes.

Workflow Designer: Task Doers



Work name

Work Description

Allow to delegate?

Allow to add adhoc tasks?

Will be done by user?

Will be done by role?

Comma separated user or role list

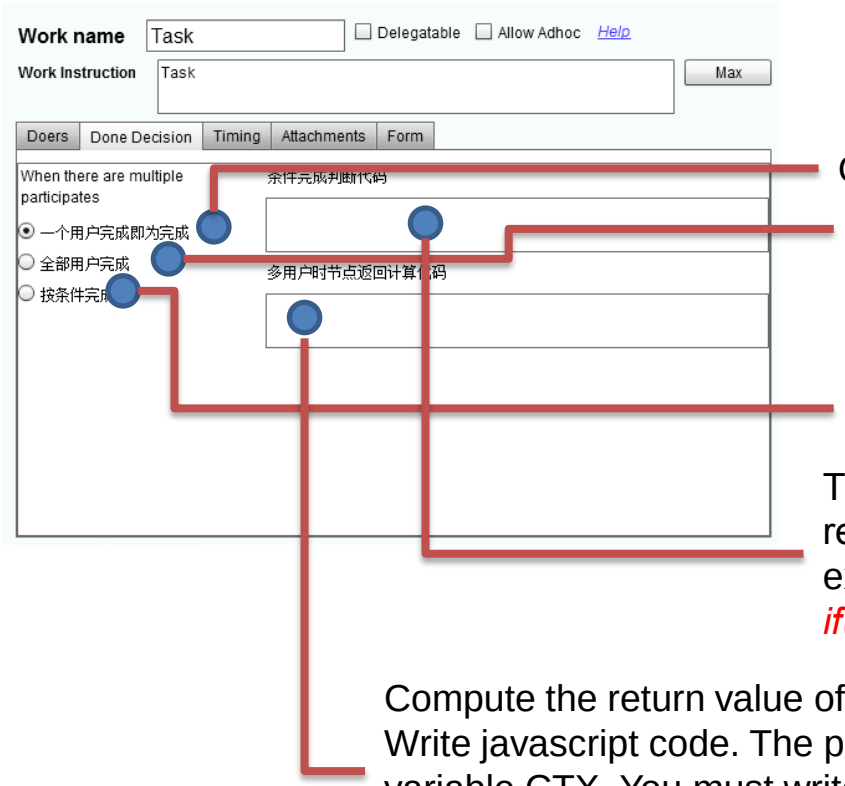
Acquirable?

Allow to change role participation?

Will be done by same DOER(s) as another task?

Except DOER(s) who have done other tasks

Workflow Designer: Done Decision



If the task will be dispatched to many users, how to determine the task is DONE?

One people complete, then DONE

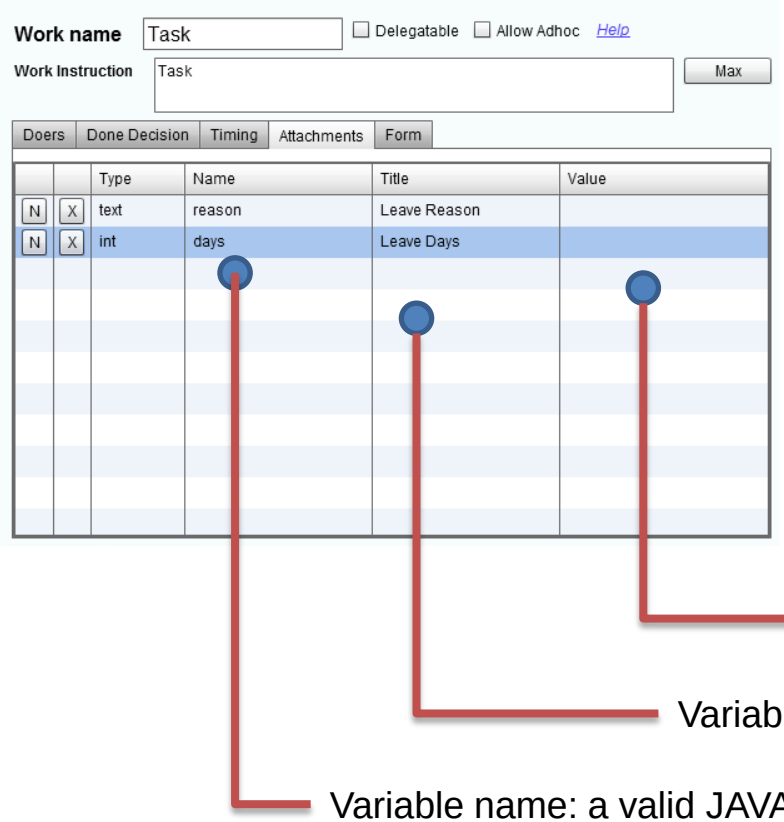
All people complete, then DONE

Determine by condition

The condition script. Put return value to “ret”. If the return value is “true”, then the task is completed. for example, 60 percent people has completed this task:
if(finished/total > 0.6) ret=“true”; else ret=“false”;

Compute the return value of this task when there are many DOERs
Write javascript code. The process contextual values are presented in a JSON variable CTX. You must write the return value to variale “ret”, for example:
if(CTX.leave_days>10) then ret = “A”; else ret = “B”

Workflow Designer: Task variables



Attachments are variables which will be displayed on task form, DOER should input their value.

Developer should display task forms.
Developer call MWF restful API /cflow/rest/task?tid=TASK_ID to get task contextual information, and display the form as will (take demo site as example to display your own form.)

Variable default value.

Variable title: for display

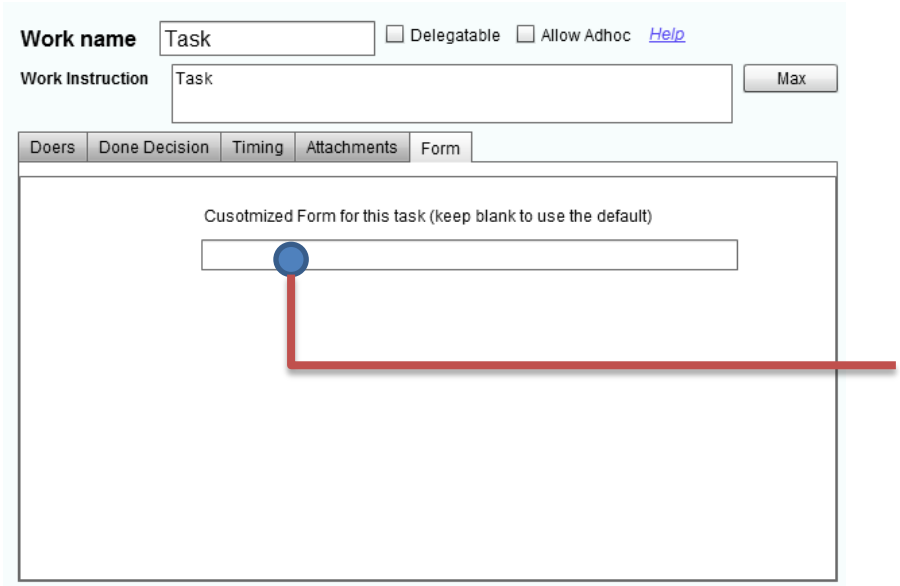
Variable name: a valid JAVA variable name.

Workflow Designer: Task variables

1. Define task varialbes in Workflow Desginer
2. Get task information with Restful API: http://www.hostcflow/rest/task?tid=TID&acsk=ACSK
3. The above API return a JSONObject: task
4. The variables are in task.ATTACHMENTS, which is a JSONArray
5. Show these variables as HTML form input area, you may also use Velocity to customize form per task.
6. When user submit a task, read these input and compose a JSON string: {[variable_name":"variable value"],}
7. Post to http://host/cflow/rest/task

Check /cflow/task.jsp source code to learn more.

Workflow Desinger: form identity



Form ID:

/cflow/rest/task?tid=TASK_ID to get task contextual information which contain this form id.

Developer may use this ID to distinguish forms displayed for different tasks. Developer can also distinguish task forms by task name when every task has different name.

Workflow Designer: SCRIPT

- MWF support three sorts of scripts
  - Javascript
  - Locale JAVA invocation
  - Remote WEB resource call
- A script node’s return value is used to routine the workflow process
- Process contextual variables can be changed within a script.

Workflow Script: JAVASCRIPT

- Write Javascript codes directly.
- Should return a String value.
- The returned value will be treated as this script node's return value, which will be used to determine workflow's following routine.
- For example:
  if(days>10) return 'long'; else return 'short';
  "days" is a process variable which has been define as an attachment of a previous node, use it directly in your script.

Workflow Script: JAVA

- Format
  JAVA:java_class_name
- The java class must be implements “com.lkh.cflow.Linker” and implement “public String linkProxy(JSONObject ctx)”
- See com.lkh.cflow.test.MyLinker source code for example
- The class must return a JSON with keys:
  - “RETURN”: the value will be the tasks’ return value
  - Other keys: if there are process variables with same name, the values of these process variables will be replaced with JSON key’s value.
- Script node’s return value is used to determine following workflow routine.
- The class must be in same CLASS Loader as MWF server.

Workflow Script: call WebService

- Format
  URL:http://.....
- See com.lkh.cflow.test.TestScriptWeb source code for example.
- The URL can be on any remote server.
- The WebService must return a JSON with keys:
  - RETURN: the value will be the tasks' return value
  - Other keys: if there are process variables with same name, the values of these process variables will be replaced with JSON key's value.
- Script node's return value is used to determine following workflow routine.
- Noramlly, the URL refers to a Servlet, a JSP, a Restful API, or a SOA Web service address.
- MWF pass contextual information to remote web service as a request parameter "CTX" which value is a JSON String, in your codes, you should parse it to a JSON Object, and read it's key values which then will be used in your own codes.

Junit Test: Script example

```
<!-- Apply Leaving, should input days and reason -->
<node id="id_apply_leaving" type="task" title="Apply Leaving" name="Apply Leaving">
        <attachment type="int" label="Leave days" attname="days" value=""/>
        <attachment type="String" label="Leave Reason" attname="reason" value=""/>
        <taskto type="role" whom="starter"/>\n <mpcdc/>\n <oec/>
        <next targetID="id_script"/>
</node>

<node id='id_script' type='script'>
        <script>URL:http://REMOTE_SERVER/cflow/TestScriptWeb</script>
        <next option='long' targetID='id_long'/>
        <next option='short' targetID='id_short'/>
</node>

<node id='id_long' type='task' name='LONG'>
        <taskto type="RefertoNode" whom="id_apply_leaving"/>
        <next targetID='id_end'/>
</node>

<node id='id_short' type='task' name='SHORT'>
        <taskto type="RefertoNode" whom="id_apply_leaving"/>
        <next targetID='id_end'/>
</node>
```

Junit Test: Script example

```
String wftid_script_web = client.uploadWft(accessKey, wft_script_web, "testProcess_5");
prcid = client.startWorkflow(accessKey, "U3306", wftid_script_web, teamid, "testProcess_5");
// id_apply_leaving
//Get worklist
wlist = client.getWorklist(accessKey);
assertTrue(wlist.size() > 0);
//A workitem with id: id_apply_leaving should exists.
theWii = client.getWorkitem(wlist, prcid, "id_apply_leaving");
assertTrue(theWii != null);
//Do “apply leaving” task, reason is “go home”.
JSONObject beforeScript = client.getPrcVariables(accessKey, prcid);
client.doTask(accessKey, prcid, (String) theWii.get("NODEID"), (String) theWii.get("SESSID"), null, "{\"days\":\“9\", \"reason\":\"gohome\"}");
// Here, the script node will be executed. com.lkh.cflow.test.MyLinker.
// will write "test value to change back to process attachment" back to
// “reason”
//Get process contextual variables after the script executed.
JSONObject afterScript = client.getPrcVariables(accessKey, prcid);
//The previous value of varialbe reason should be “gohome”
assertTrue(beforeScript.get("reason").equals("gohome"));
//After script execution, the value should be changed.
assertTrue(afterScript.get("reason").equals("test value to change back to process attachment."));
assertTrue(afterScript.get("ignored") == null);

//Process should run to id_short node.
wlist = client.getWorklist(accessKey);
theWii = client.getWorkitem(wlist, prcid, "id_short");
assertTrue(theWii != null);
assertEquals(theWii.get("WORKNAME"), "SHORT");
```

TestScriptWeb code snippest

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        request.setCharacterEncoding("UTF8");
        String ctx_string = request.getParameter("CTX");
        JSONParser parser = new JSONParser();
        JSONObject ctx;
        String ret = "";
        try {
        ctx = (JSONObject) parser.parse(ctx_string);

        JSONObject attachments = (JSONObject) ctx.get("ATTACHMENTS");
        String days = (String) attachments.get("days");
        int idays = Integer.valueOf(days).intValue();
        if (idays > 10)
                ret = "long";
        else
                ret = "short";
        } catch (ParseException e) {
                ret = "error";
                e.printStackTrace();
        }
        JSONObject retObj = new JSONObject();
        retObj.put("RETURN", ret);
        retObj.put("reason", "test value to change back to process attachment.");
        retObj.put("ignored", "this will be ignored.");

        ret = retObj.toJSONString();
        response.getWriter().print(ret);
        response.getWriter().flush();
        response.flushBuffer();
}
```

Restful API: more

- http://www.myworldflow.com/cflow/tutorial/mwf_api.html

SDK

- JAVA SDK
  - http://www.myworldflow.com/cflow/tutorial/programming_with_mwf_sdk.html
- PHP/C# SDK
  - Wrap Restful API as will