Mini-Language Guide (Version 2 - Deprecated)

Originally written By: David E. Jones, jonesde@ofbiz.org

This is deprecated since revisions 1360689 and 1361073 in trunk and r1361651 in R12.04. Older releases continue to use the previous version here documented.
The new documentation is at Mini Language - minilang - simple-method - Reference. There is still no documentation for Map processor though, the one below can be used

---

Table of Contents

Introduction

The Mini-Language concept in Open For Business is similar to the Gang of Four Interpreter pattern, or the Mark Grand Little Language pattern. This is also the central theme of the Building Parsers with Java book by Steven John Metsker which the OFBiz Rule Engine is based on. The idea is to create simple languages that simplify complex or frequently performed tasks.

In business software there are things that are done hundreds or thousands of times in a single application that a Mini-Language can simplify to the point of cutting implementation and maintenance times not by just 50%, but often as much as 70-90%. Yes, certain tasks will take only 10% of the effort and knowledge to perform. This makes it easier people not familiar with the software to manipulate existing or build new functionality.

Mini-Languages tend to have instructions that are more like method calls in a general purpose language. They are meant to solve a specific problem in a specific context, and are generally worthless or need to be modified for other contexts or problems.

Often this idea is implemented using a free form (BNF) or english-like syntax. In Open For Business the Mini-Languages are expressed as XML files to simplify the learning and manipulation of the syntax, in addition to making the Mini-Languages easier to write and extend.

The XML schema (XSD) for the simple-map-processor and simple-method XML files is in the distribution in **framework/minilang/dtd/simple-methods.xsd** . These are combined into a single file to make it easy to use inlined simple-map-processors inside a simple-method.

Note that the simple-method.xsd file is also available at http://ofbiz.apache.org/dtds/simple-methods.xsd .

To specify the XML schema for a simple-methods or simple-map-processors XML file use the following:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://ofbiz.apache.org
/dtds/simple-methods.xsd">
```

The Simple Map Processor Mini-Language

Simple Map Processor Overview

The Simple Map Processor Mini-Language performs two primary tasks: validation and conversion. It does this in a context of moving values from one Map to another. The input map will commonly contain Strings, but can contain other object types like Integer, Long, Float, Double, java.sql.Date, Time, and Timestamp.

NOTE: The reference information for the simple-map-processor has been moved to annotations in the http://ofbiz.apache.org/dtds/simple-methods.xsd file. The idea is to now to use an XML completion tool in development. **If you are interested by this issue take a look at https://issues.apache.org/jira/browse/OFBIZ-571**

As static documentation you may find a simple-methods.html generated file in attachments. This file may be out of date but with the attached simple-methods.xsl file you are able to generate an updated file. In such case please feel free to put a comment below in order to let us know, thanks.

Simple Map Processors Example

Note that fail-message is used here for the purpose of demonstration but in general it's better to use fail-property because fail-message is not localisable

```
<simple-map-processors xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:noNamespaceSchemaLocation="http://ofbiz.apache.org/dtds/simple-methods.xsd">
        <simple-map-processor name="update">
                <make-in-string field="estimatedStartDate">
                        <in-field field="estimatedStartYear"><constant>-</constant>
                        <in-field field="estimatedStartMonth"><constant>-</constant>
                        <in-field field="estimatedStartDay"><constant>T</constant>
                        <in-field field="estimatedStartHour"><constant>:</constant>
                        <in-field field="estimatedStartMinute"><constant>:</constant>
                        <in-field field="estimatedStartSecond">
                </make-in-string>
                <process field="workEffortId"><copy replace="false"/></process>
                <process field="scopeEnumId"><copy/></process>
                <process field="currentStatusId">
                        <copy/>
                        <not-empty>
                <!-- Note that fail-message is used here for the purpose of demonstration but in general it's
better to use fail-property because fail-message is not localisable -->
                                <fail-message message="Status is missing."/>
                                <!-- Note that fail-message is used here for the purpose of demonstration;
                                but in general it's better to use fail-property because fail-message is not
localisable -->
                        </not-empty>
                </process>
                <process field="priority">
                        <convert type="Long">
                                <fail-message message="Priority is not a valid whole number."/>
                        </convert>
                </process>
                <process field="estimatedStartDate">
                        <compare-field operator="less" field="estimatedCompletionDate" type="Timestamp" format="
yyyy-MM-dd'T'HH:mm:ss">
                                <fail-message message="Estimated Start date/time must be BEFORE End date/time."
/>
                        </compare-field>
                        <convert type="Timestamp" format="yyyy-MM-dd'T'HH:mm:ss">
                                <fail-message message="Estimated Start Date is not a valid Date-Time."/>
                        </convert>
                </process>
                <process field="estimatedCompletionDate">
                        <convert type="Timestamp">
                                <fail-message message="Estimated Completion Date is not a valid Date-Time."/>
                        </convert>
                </process>
                <process field="estimatedMilliSeconds">
                        <convert type="Double">
                                <fail-message message="Estimated Milli-seconds is not a valid number."/>
                        </convert>
                </process>
        </simple-map-processor>
        <simple-map-processor name="delete">
                <process field="workEffortId">
                        <copy/>
                        <not-empty>
                                <fail-message message="Work Effort ID is missing."/>
                        </not-empty>
                </process>
        </simple-map-processor>
</simple-map-processors>
```

The Simple Method Mini-Language

Simple Method Overview

The Simple Method Mini-Language is a simple way to implement an event that is invoked by the Control Servlet or a service that is invoked by the Service Engine. A Simple Method can be invoked through the static methods on the SimpleMethod class, or as an event through an entry in the controller configuration XML file like the following:

```
<event type="simple" path="org/ofbiz/commonapp/workeffort/workeffort/WorkEffortSimpleEvents.xml" invoke="update"
/>
```

or as a service through an entry in a services.xml file like the following:

```
<service name="createPartyRole" engine="simple" location="org/ofbiz/commonapp/party/party/PartyRoleServices.
xml" invoke="createPartyRole" auth="true">
        <description>Create a Party Role (add a Role to a Party)</description>
        <attribute name="partyId" type="String" mode="IN" optional="true"/>
        <attribute name="roleTypeId" type="String" mode="IN" optional="false"/>
</service>
```

The path or location for a Simple Method is the classpath and filename of the XML file.

In this Mini-Language you can invoke Simple Map Processors, Services and bsh scripts, perform entity related operations, and create messages to return to the caller. Specific operations can be enclosed in if blocks to execute conditionally and values or fields can be copied around in the maps, lists and method environment.
There are a number of tags which can be used to get and set attributes to/from a request or session object when called as an event or to set attributes in the result when called as a service. These operations are only applied when applicable. In other words if you include an **env-to-request** operation it will only be invoked when the simple-method is called as an event and an **env-to-result** operation will only be invoked when the simple-method is called as a service. Everything else is the same when called as an event or a service which makes it easy to write flexible logic that can be mounted/applied in various ways.
There are a number of objects that exist in the method environment when a simple-method starts or that are used as it executes to keep track of certain information. Some will exist when called as an event or a service, these are marked in the XSD. Each name can be overridden using an attribute on the **simple-method** tag. The defaults are listed below in the XSD.

NOTE: The reference information for simple-method has been moved to annotations in the http://ofbiz.apache.org/dtds/simple-methods.xsd file. The idea is to now to use an XML completion tool in development. **If you are interested by this issue take a look at https://issues.apache.org/jira/browse/OFBIZ-571**

As static documentation you may find a simple-methods.html generated file in attachments with reference information. This file may be out of date but with the also attached simple-methods.xsl file you are able to generate an updated file. In such case please feel free to put a comment below in order to let us know about that, thanks.

Special Context Access Syntax

In strings and field names a special syntax is supported to flexibly access Map member, List elements and to insert environment values into string constants.

The ${} (dollar-sign-curly-brace) syntax can be used to insert an environment variable value in pretty much any string constant in a simple-method file. Not only can it be used to reference top-level environment variables, the syntax elements described below can be used to access values in sub-structures.

You should use the "." (dot) syntax to access Map members (the old map/field syntax is deprecated since 2009). For example if you specify the attribute field="product.productName" it will reference the productName member of the productMap. This would be the same as specifying map="product" field="productName". Note that this is, of course, more flexible than a field/map combination because the Map structure can be multiple levels deep. For example if you have used the attribute field="products.widget.productName" it will reference the productName in the widget Map which is in the products Map.

The "[]" (square-brace) syntax can be used to access list elements. For example you can specify the attribute field="products[0].productName"and it will reference the productName of the first (position zero) element in the products List. To make this more useful you can pull a list index from the environment using something like field="products[${currentIndex}].productName".

There are two extensions to the [] syntax that can be used when referring to an environment location that is the target of an operation. If you do not include a number between the square braces the value will be put at the end of the list. If you put a "+" (plus sign) in front of the number between the square braces (ie: [+2]) it will insert the value before that position in the list instead of replacing the value at that location. For example, specifying [+0] would insert the value at the beginning of the list.

In fact, you can use the ${} syntax to substitute any string or other value at any location in a field or other string constant. So, you could even reference a Map member named in some other environment variable. For example you could use field="products[${currentIndex}].productName".

Okay, enough of the general stuff, you may find in the XSD file descriptions of the available operations. Here is simply a categorized list of them.

Call Operations

call-map-processor
set-service-fields
call-service
call-service-asynch
script
call-bsh
call-simple-method

Java Call Operations

create-object
call-object-method
call-class-method

Control and Error Handling Operations

check-errors
add-error
return

Event Specific Operations

field-to-request
field-to-session
request-to-field
request-parameters-to-list
session-to-field
webapp-property-to-field

Service Specific Operations

field-to-result

Method Environment Operations

map-to-map
field-to-list
list-to-list
order-map-list
set
string-append
string-to-list
to-string
clear-field
first-from-list

All operations in red below have been be replaced by the set operation
field-to-field (deprecated, do not use anymore)
env-to-env (deprecated, do not use anymore)
env-to-field (deprecated, do not use anymore)
field-to-env (deprecated, do not use anymore)
string-to-field (deprecated, do not use anymore)

Control Operations

iterate
iterate-map
loop
first-from-list

Entity Engine Misc. Operations

now-timestamp
now-date-to-env
sequenced-id
make-next-seq-id
entity-data

Entity Engine Find Operations

find-by-primary-key
find-by-and
entity-one
entity-and
entity-condition
entity-count
get-related-one
get-related
order-value-list
filter-list-by-and
filter-list-by-date

Entity Engine Value Operations

make-value
clone-value
create-value
store-value
refresh-value
remove-value
remove-related
remove-by-and
clear-cache-line
clear-entity-caches
set-pk-fields
set-nonpk-fields

Entity Engine List Operations

store-list
remove-list

Entity Engine Transaction Operations

transaction-begin
transaction-commit
transaction-rollback

Conditional (If) Operations

if
if-validate-method
if-instance-of
if-compare
if-compare-field
if-regexp
if-empty
if-not-empty
if-has-permission
check-permission
check-id
assert
while

Other Operations

property-to-field
log
calculate
set-current-user-login
set-calendar

Here is an example of an XML snippet that performs the calculation **a=b+(((c+x+2)-d)/e)**, or in Reverse Polish Notation (a little bit closer to the resulting XML, and the notation used in the Rule Engine) **a=(b,/(((c,x,2),-d),e))**.

Here is the XML:

```
<calculate field="a">
<calcop operator="get" field="b"/>
<calcop operator="divide">
<calcop operator="multiply">
<calcop operator="add" field="c">
<calcop operator="get" field="x"/>
<number value="2"/>
</calcop>
<calcop operator="negative" field="d"/>
</calcop>
<calcop operator="get" field="e"/>
</calcop>
</calculate>
```

Simple Methods Example

Note that fail-message is used here for the purpose of demonstration but in general it's better to use fail-property because fail-message is not localisable

```
<simple-methods xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="[http://ofbiz.apache.org/dtds/simple-methods.xsd]\]">

    <simple-method method-name="createProduct" short-description="Create a Product">
        <check-permission permission="CATALOG" action="_CREATE">
            <alt-permission permission="CATALOG_ROLE" action="_CREATE"/>
            <!-- Note that fail-message is used here for the purpose of demonstration but in general it's
better to use fail-property because fail-message is not localisable -->
            <fail-message message="Security Error: to run createProduct you must have the CATALOG_CREATE or
CATALOG_ADMIN permission, or the limited CATALOG_ROLE_CREATE permission"/>
        </check-permission>
        <check-errors/>

        <make-value value-field="newEntity" entity-name="Product"/>
        <set-nonpk-fields map="parameters" value-field="newEntity"/>

        <set from-field="parameters.productId" field="newEntity.productId"/>
        <if-empty field="newEntity.productId">
            <sequenced-id sequence-name="Product" field="newEntity.productId"/>
        <else>
            <check-id field="newEntity.productId"/>
            <check-errors/>
        </else>
        </if-empty>
        <field-to-result field="newEntity.productId" result-name="productId"/>

        <now-timestamp field="nowTimestamp"/>
        <set from-field="nowTimestamp" field="newEntity.createdDate"/>
        <set from-field="nowTimestamp" field="newEntity.lastModifiedDate"/>
        <set from-field="userLogin.userLoginId" field="newEntity.lastModifiedByUserLogin"/>
        <set from-field="userLogin.userLoginId" field="newEntity.createdByUserLogin"/>
        <if-empty field="newEntity.isVariant">
            <set field="newEntity.isVariant" value="N"/>
        </if-empty>
        <if-empty field="newEntity.isVirtual">
            <set field="newEntity.isVirtual" value="N"/>
        </if-empty>
        <if-empty field="newEntity.billOfMaterialLevel">
            <set field="newEntity.billOfMaterialLevel" value="0" type="Integer"/>
        </if-empty>

        <create-value value-field="newEntity"/>

        <!-- if setting the primaryProductCategoryId create a member entity too -->
```

```
        <!-- THIS IS REMOVED BECAUSE IT CAUSES PROBLEMS FOR WORKING ON PRODUCTION SITES
        <if-not-empty field="newEntity.primaryProductCategoryId">
            <make-value entity-name="ProductCategoryMember" value-field="newMember"/>
            <set from-field="productId" map-name="newEntity" to-field-name="productId" to-map-name="newMember"/>
            <set from-field="primaryProductCategoryId" map-name="newEntity" to-field-name="productCategoryId"
to-map-name="newMember"/>
            <now-timestamp field="nowStamp"/>
            <set from-field="nowStamp" field="newMember.fromDate"/>
            <create-value value-field="newMember"/>
        </if-not-empty>
        -->

        <!-- if the user has the role limited position, add this product to the limit category/ies -->
        <if-has-permission permission="CATALOG_ROLE" action="_CREATE">
            <entity-and entity-name="ProductCategoryRole" list="productCategoryRoles" filter-by-date="true">
                <field-map field-name="partyId" from-field="userLogin.partyId"/>
                <field-map field-name="roleTypeId" value="LTD_ADMIN"/>
            </entity-and>

            <iterate entry="productCategoryRole" list="productCategoryRoles">
                <!-- add this new product to the category -->
                <make-value value-field="newLimitMember" entity-name="ProductCategoryMember"/>
                <set from-field="newEntity.productId" field="newLimitMember.productId"/>
                <set from-field="productCategoryRole.productCategoryId" field="newLimitMember.productCategoryId"
/>
                <set from-field="nowTimestamp" field="newLimitMember.fromDate"/>
                <create-value value-field="newLimitMember"/>
            </iterate>
        </if-has-permission>
    </simple-method>

    <simple-method method-name="createWorkEffort" short-description="Create Work Effort">
        <make-value value-field="newEntity" entity-name="WorkEffort"/>
        <if-empty field="parameters.workEffortId">
            <sequenced-id sequence-name="WorkEffort" field="newEntity.workEffortId"/>
            <else>
                <set field="newEntity.workEffortId" from-field="parameters.workEffortId"/>
            </else>
        </if-empty>
        <field-to-result field="newEntity.workEffortId" result-name="workEffortId"/>
        <set-nonpk-fields map="parameters" value-field="newEntity"/>

        <now-timestamp field="nowTimestamp"/>
        <set from-field="nowTimestamp" field="newEntity.lastStatusUpdate"/>
        <set from-field="nowTimestamp" field="newEntity.lastModifiedDate"/>
        <set from-field="nowTimestamp" field="newEntity.createdDate"/>
        <set field="newEntity.revisionNumber" value="1" type="Long"/>
        <set from-field="userLogin.userLoginId" field="newEntity.lastModifiedByUserLogin"/>
        <set from-field="userLogin.userLoginId" field="newEntity.createdByUserLogin"/>
        <create-value value-field="newEntity"/>

        <!-- create new status entry, and set lastStatusUpdate date -->
        <make-value value-field="newWorkEffortStatus" entity-name="WorkEffortStatus"/>
        <set from-field="newEntity.workEffortId" field="newWorkEffortStatus.workEffortId"/>
        <set from-field="newEntity.currentStatusId" field="newWorkEffortStatus.statusId"/>
        <set from-field="nowTimestamp" field="newWorkEffortStatus.statusDatetime"/>
        <set from-field="userLogin.userLoginId" field="newWorkEffortStatus.setByUserLogin"/>
        <create-value value-field="newWorkEffortStatus"/>

        <!-- Attach the workeffort to a requirement if passed -->
        <if-not-empty field="parameters.requirementId">
            <make-value value-field="workFullfillment" entity-name="WorkRequirementFulfillment"/>
            <set from-field="newEntity.workEffortId" field="workFullfillment.workEffortId"/>
            <set from-field="parameters.requirementId" field="workFullfillment.requirementId"/>
            <create-value value-field="workFullfillment"/>
        </if-not-empty>

        <!-- attach to a customer request if passed and copy attached docs -->
        <if-not-empty field="parameters.custRequestId">
            <!-- check status of customer request if valid -->
            <entity-one entity-name="CustRequest" value-field="lookedUpValue"/>
```

```
            <set field="goodStatusId" value="CRQ_ACCEPTED"/>
            <if-compare-field operator="not-equals" field="lookedUpValue.statusId" to-field="goodStatusId" >
                <set field="entity" value="Customer request"/>
                <add-error><fail-property resource="CommonUiLabels" property="CommonErrorStatusNotValid"/></add-
error>
                <check-errors/>
            </if-compare-field>
            <!-- create customer request / work effort relation -->
            <make-value value-field="custRequestWorkEffort" entity-name="CustRequestWorkEffort"/>
            <set field="custRequestWorkEffort.workEffortId" from-field="newEntity.workEffortId" />
            <set field="custRequestWorkEffort.custRequestId" from-field="parameters.custRequestId"/>
            <create-value value-field="custRequestWorkEffort"/>
            <!-- update status of customer request -->
            <set field="updCustReq.custRequestId" from-field="parameters.custRequestId"/>
            <set field="updCustReq.statusId" value="CRQ_REVIEWED"/>
            <call-service service-name="setCustRequestStatus" in-map-name="updCustReq"/>
            <entity-and list="custRequestContents" entity-name="CustRequestContent">
                <field-map field-name="custRequestId" from-field="parameters.custRequestId"/>
            </entity-and>
            <iterate entry="custRequestContent" list="custRequestContents">
                <set field="newWorkEffortContent.workEffortId" from-field="newEntity.workEffortId"/>
                <set field="newWorkEffortContent.contentId" from-field="custRequestContent.contentId"/>
                <set field="newWorkEffortContent.workEffortContentTypeId" value="SUPPORTING_MEDIA"/>
                <call-service service-name="createWorkEffortContent" in-map-name="newWorkEffortContent"/>
            </iterate>
        </if-not-empty>
    </simple-method>
    <simple-method method-name="updateWorkEffort" short-description="Update Work Effort">
        <!-- check permissions before moving on: if update or delete logged in user must be associated OR have
the corresponding UPDATE or DELETE permissions -->

        <!-- temporarily commented out, because users assigned to a project or phase should
              have the capability to modify status on sub-tasks, right? Hmmmm.... -->

<!--        <set from-field="workEffortId" map-name="parameters" to-map-name="findWepaMap"/>
        <set from-field="partyId" map-name="userLogin" to-map-name="findWepaMap"/>
        <find-by-and entity-name="WorkEffortPartyAssignment" map="findWepaMap" list="wepaList"/>
        <if-empty field="wepaList">
            <check-permission permission="WORKEFFORTMGR" action="_UPDATE">
                <fail-message message="Security Error: to run updateWorkEffort you must have the
WORKEFFORTMGR_UPDATE or WORKEFFORTMGR_ADMIN permission"/>
            </check-permission>
            <check-errors/>
        </if-empty>-->

        <entity-one entity-name="WorkEffort" value-field="lookedUpValue"/>
        <clone-value value-field="lookedUpValue" new-value-field="savedValue"/>

        <now-timestamp field="nowTimestamp"/>

        <!-- if necessary create new status entry, and set lastStatusUpdate date -->
        <if>
            <condition>
                <and>
                    <not><if-empty field="parameters.currentStatusId"/></not>
                    <if-compare-field field="parameters.currentStatusId" to-field="lookedUpValue.
currentStatusId" operator="not-equals"/>
                </and>
            </condition>
            <then>
                <if-not-empty field="lookedUpValue.currentStatusId">
                    <!-- check if the status change is a valid change -->
                    <entity-and entity-name="StatusValidChange" list="validChange">
                        <field-map field-name="statusId" from-field="lookedUpValue.currentStatusId"/>
                        <field-map field-name="statusIdTo" from-field="parameters.currentStatusId"/>
                    </entity-and>

                    <if-empty field="validChange">
                        <add-error>
                            <fail-message message="The status change from ${lookedUpValue.currentStatusId} to
${parameters.currentStatusId} is not a valid change"/>
```

```
                        </add-error>
                        <log level="error" message="The status change from ${lookedUpValue.currentStatusId} to
${parameters.currentStatusId} is not a valid change"/>
                        <check-errors/>
                    </if-empty>
                </if-not-empty>

                <set from-field="nowTimestamp" field="lookedUpValue.lastStatusUpdate"/>
                <make-value value-field="newWorkEffortStatus" entity-name="WorkEffortStatus"/>
                <set from-field="lookedUpValue.workEffortId" field="newWorkEffortStatus.workEffortId"/>
                <set from-field="parameters.currentStatusId" field="newWorkEffortStatus.statusId"/>
                <set from-field="parameters.reason" field="newWorkEffortStatus.reason"/>
                <set from-field="nowTimestamp" field="newWorkEffortStatus.statusDatetime"/>
                <set from-field="userLogin.userLoginId" field="newWorkEffortStatus.setByUserLogin"/>
                <create-value value-field="newWorkEffortStatus"/>
            </then>
        </if>
        <!-- after checking status change, set all parameters -->
        <set-nonpk-fields map="parameters" value-field="lookedUpValue"/>

        <!-- only save if something has changed -->
        <if-compare-field field="lookedUpValue" to-field="savedValue" operator="not-equals" type="Object">
            <!-- only set lastModifiedDate after comparing new & old to see if anything has changed -->
            <set from-field="nowTimestamp" field="lookedUpValue.lastModifiedDate"/>
            <set from-field="userLogin.userLoginId" field="lookedUpValue.lastModifiedByUserLogin"/>
            <if-not-empty field="lookedUpValue.revisionNumber">
                <set field="lookedUpValue.revisionNumber" value="${lookedUpValue.revisionNumber + 1}" type="
Long"/>
            <else>
                <set field="lookedUpValue.revisionNumber" value="1" type="Long"/>
            </else>
            </if-not-empty>
            <store-value value-field="lookedUpValue"/>
        </if-compare-field>
    </simple-method>

</simple-methods>
```