Service Manager Document Engine

How to use and troubleshoot the document engine tailoring tool

HP® Management Software **Service Management**

invent

Introduction

What is the Document Engine?

The Document Engine comprises a set of tools and methodologies for developing and modifying Service Manager workflows. The Document Engine was originally tasked to develop an underlying set of base functionality that would support multiple modules inside Service Manager, improve consistency of the user interface between modules and reduce the amount of code needed for any new module.

The benefits of using the Document Engine as a tailoring tool include the possibility to customize the system without the need of RAD changes. It provides a centralized method for setting privileges and behavior for standard actions, such as list, view and search, thus increases consistency across modules. Since existing Processes can be reused for more modular programming and the integration with display is seamless, development time is greatly reduced. The Document Engine extends the display application capabilities with simplified and more extensive actions especially involving multiple application calls. In addition, the Document Engine provides frameworks for joined tables and master format control calls.

The Document Engine controls behavior with Objects. An Object is referenced whenever a form is opened and determines behavior for the state of the form (open, list, search, etc.). Implementers can change behavior easily for standard actions across Service Manager in one place with one Object.

The Document Engine comprises Objects, States, and Processes to centralize tasks. It is designed to meet the needs of most customers out of the box, yet retain flexibility.

Note: Before making changes to any Objects, States, or Processes, HP recommends making a backup of the files you modify.

The relation between Objects, States, and Processes is hierarchical:

- Objects handle States and Processes, which are called from the module. Objects are overall file behavior definitions.
- States define how a record is displayed and what options are available at specific times or circumstances. For instance, States can determine an action, such as Save, given a user's access privileges. Processes are called by States.
- Processes use RAD expressions or JavaScript to perform work. Processes modify information or perform an action on records.

Benefits of the Modular Approach

Developing in the Document Engine can reuse existing code and Processes. The advantages to the modular design are listed below.

Consistency

The fact that the engine allows all applications to run using the same base RAD applications brings consistency to the Service Manager application suite. Core functionality such as locking, alerts, approvals, use of record list functionality, etc. will work the same for any module as they are all using the same code base.

Reduced Development Times

The modularity of the engine will allow for reuse of existing code and Processes.

Flexibility

The engine's use of process records allows an easier mechanism to modify the behavior of modules inside of Service Manager applications. You can create a new process that has a different behavior than the base system, without the need to change or remove the original process from the system. Additionally, the system's base processes can be overridden with your own processes giving an unprecedented flexibility to the system developer.

Accessing the Document Engine

To access the Document Engine:

- Start a Service Manager Client and log in as an administrator.
- Click **Menu Navigation** in the Service Manager System Navigator.
- Click **Tailoring**.
- Click **Document Engine**. From here, you can access the main three areas that control the Documentation Engine: select **Objects**, **Processes** or **States**. Also available are **Alerts**, **Approvals** and **Search Configuration Records** which are used by **Objects** and explained further in the **Objects** tab section.

Objects

Objects are a base set of definitions that determine behavior of screens and panels and set the definitions and governing rules. Objects correspond one-to-one with dbdict records in Service Manager. If a file does not have a dedicated Object record, the **DEFAULT** Object is used and uses standard database functionality.

NOTE: Do not modify or delete the DEFAULT Object record as doing so could cause unwanted results.

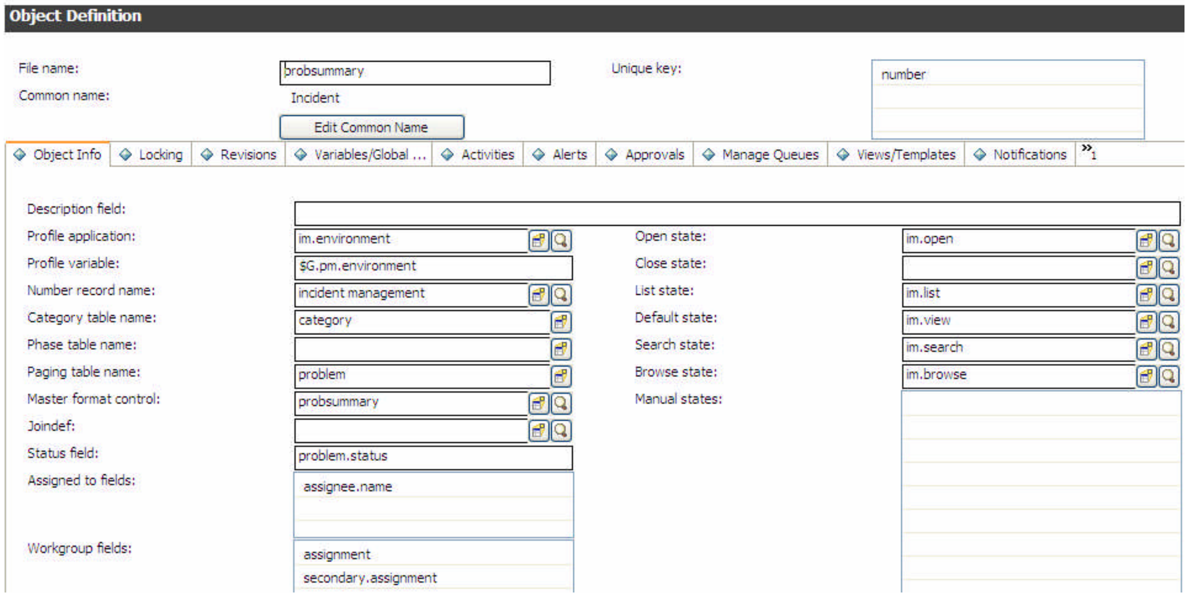
The Object record sets up the definitions and governing rules for the behavior of the file within the Document Engine. These include:

- The Service Manager table name and unique key structure.
- The application used to create the users profile within this Object that determines what actions this user may take against any record of that Object.
- The State records used in specific circumstances (see the *States* section for more details).
- The category, phase and paging file names for the Object.
- The name of the number record to be used for this Object.
- How locking is to be used by this Object.
- Setup of revisions for records in this table.
- Which variables should be available to processes that run against this Object.
- Which global lists should always be available when using this Object.
- Use of activity records.
- How alerts are processed against this Object.
- How approvals are processed for this Object.
- Settings for the work queues.
- Ability to set up personal or global views and default templates.
- Notifications on add / update / delete of a record in this Object.
- Ability to configure additional search choices against this Object.

Creating Objects

To create an Object:

- Navigate to the Document Engine. See *Accessing the Document Engine* for steps.
- Double click Objects. The Objects panel opens.
- Using the tabs on the Objects panel, fill in the fields required to create an Object that will perform the functions you desire. See the field descriptions that follow.



File name:	probsummary	Unique key:	number
Common name:	Incident		
Description field:			
Profile application:	im.environment	Open state:	im.open
Profile variable:	$G.pm.environment	Close state:	
Number record name:	incident management	List state:	im.list
Category table name:	category	Default state:	im.view
Phase table name:		Search state:	im.search
Paging table name:	problem	Browse state:	im.browse
Master format control:	probsummary	Manual states:	
Joindef:			
Status field:	problem.status		
Assigned to fields:	assignee.name		
Workgroup fields:	assignment secondary.assignment		

Fields on the Object Definition panel

File name - Enter the dbdict name for the Object

Common name - Enter a common name for the Object. The common name can be a simple name, such as Incident.

Unique key - The Unique key for the Object should match the corresponding dbdict unique key field.

The Object Info Tab

General properties and Object behavior are specified with the fields under the Object Info tab.

Description field - Enter a short description of the Object, if desired.

Profile application - Enter the RAD application that creates the profile that determines if a user can perform certain functions, such as add, delete and so on.

Profile variable - This field is optional. Enter a variable that can be accessed any time this Object is called without accessing the environment record.

Number record name - This field is optional. Define a number class inside the Object which can be referred to in a Process or RAD code.

Category table name - Enter the file name that links with the category file associated with this Object, if applicable. When displaying a record of this type, if a field called category exists, then the Object will go to the Category file and select a record with a corresponding name. If found, the system will store the Category File Name as a variable: $L.category

Phase table name - Enter the file name that links with the phase file associated with this Object, if applicable.

Paging table name - Enter the name of the file for storing pages. Pages are created every time a record is updated, creating a detailed audit trail.

Master format control - Enter the name of the Master Format Control record, if one exists for the record. Master Format Control allows you to define in one record the Format Control statements that apply to all phases in an area, for example Change Management Request Phases. Typically the name of the master format control is the name of the dbdict.

Joindef – Enter the name of the joindef record used.

Status field – Enter the field name for status for this Object.

Assigned to fields – Enter the field name that contains the assignee name field for this Object. This field will be referred to when Folder Entitlement verifies that the record is assigned to the logged in operator.

Workgroup fields – Enter the field name that contains the assignment group field for this Object. This field will be referred to when Folder Entitlement verifies that the record is assigned to any of the logged in operator's work groups.

Open state - Enter the State definition record to use upon opening.

Close state - Enter the State definition record to use upon closing.

List state - Enter the State definition record to use for listing results.

Default state - Enter the default State of the Object. The default state is always used for editing a record.

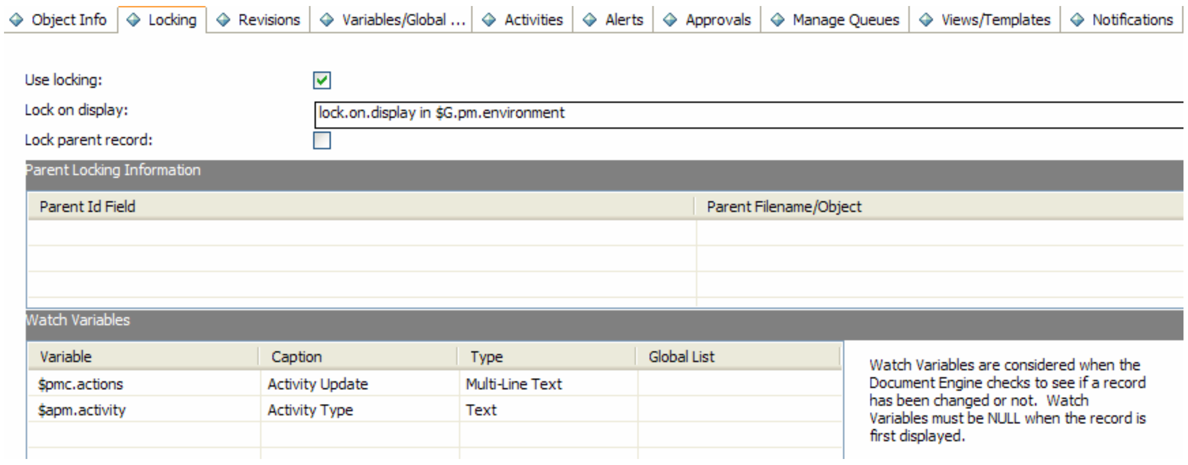
Search state - Enter the State definition record to use for searching.

Browse state - Defines the State definition record to use when records use locking. Essentially, this field defines a read-only State.

Manual states – Array of States that may be used with this Object other than the life-cycle states of open, close, list, view, search or browse.

The Locking Tab

The Locking tab is used to determine the locking behavior for the Object.



Use locking: ✓

Lock on display: lock.on.display in $G.pm.environment

Lock parent record: ☐

Parent Locking Information

Parent Id Field	Parent Filename/Object

Watch Variables

Variable	Caption	Type	Global List
$pmc.actions	Activity Update	Multi-Line Text	
$apm.activity	Activity Type	Text	

Watch Variables are considered when the Document Engine checks to see if a record has been changed or not. Watch Variables must be NULL when the record is first displayed.

Use locking - Select this check box to enable locking.

Lock on display - If true or a condition that evaluates to true, HP Service Manager attempts to lock the record as soon as it is displayed. **Use locking** must be enabled for this field to work.

Lock parent record - Locks the current record and the record's parent.

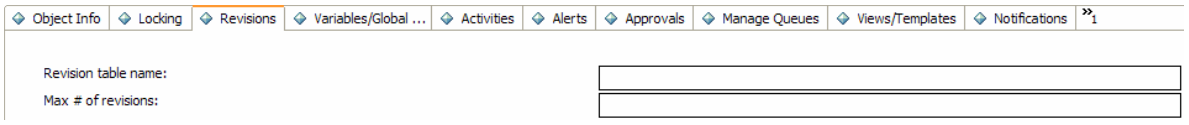
Parent Id Field - Enter a field name in the current record that contains the ID of the parent.

Parent Filename/Object - The name of the file that contains the parent record.

Watch Variables – As stated on the tab.

The Revisions Tab

The Revisions tab is used to determine the revisioning behavior for the Object.



Revision table name:

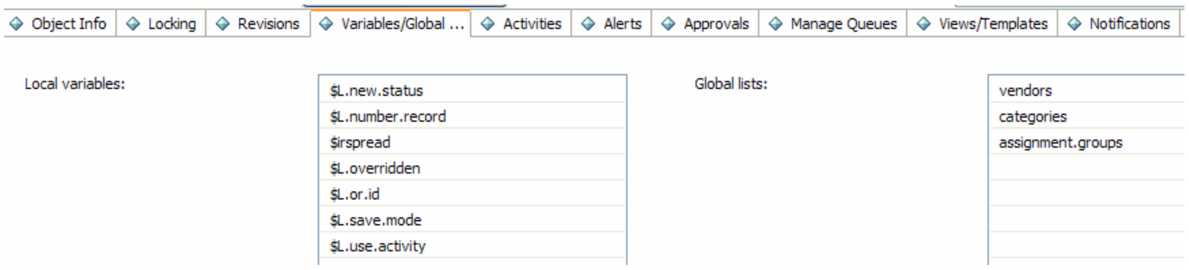
Max # of revisions:

Revision table name - Revisions will be saved to the table name you enter here.

Max # of revisions - The total number of revisions allowed for this Object. If left blank, then there is no upper limit.

Variable/Global Lists Tab

The Variable/Global Lists Tab is where you can describe local variables and global variables used by the Object. Global variables are built and stored in memory and available to the Object for as long as the Object persists.



Local variables:

$L.new.status
$L.number.record
$irspread
$L.overridden
$L.or.id
$L.save.mode
$L.use.activity

Global lists:

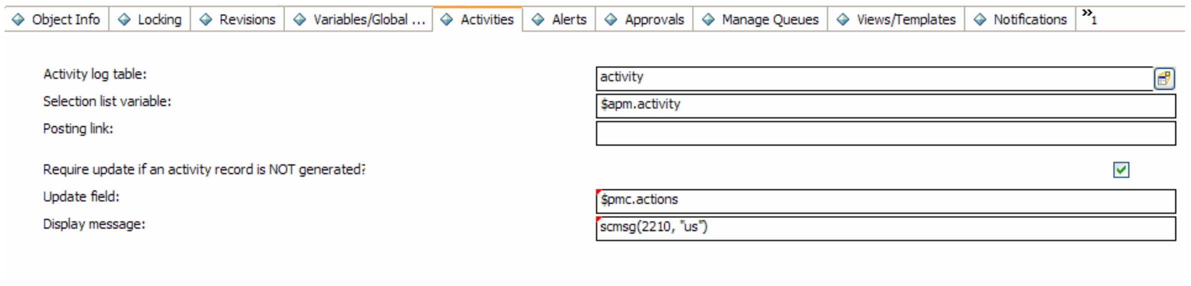
vendors
categories
assignment.groups

Local variables - Enter a list of local variables that can be used in Processes. Local variables are defined in the application. Local variables are assigned to the Object you are creating and are available to all Processes and States associated with the Object

Global lists - Global lists, once created, are available to all Processes. Global lists can be built on login, if they exist in the global list file. Global lists are available every time you access the Object.

The Activities Tab

The Activities tab is used for activity logging for the object.



Activity log table:	activity
Selection list variable:	$apm.activity
Posting link:	
Require update if an activity record is NOT generated?	✓
Update field:	$pmc.actions
Display message:	scmsg(2210, "us")

Activity log table – Name of the table to hold the activity log entries for the Object.

Selection list variable – The variable to use on an update form to display the types of activities that the operator can select from when performing an update on a ticket for a specific Object.

Posting link – Name of the link used to post information.

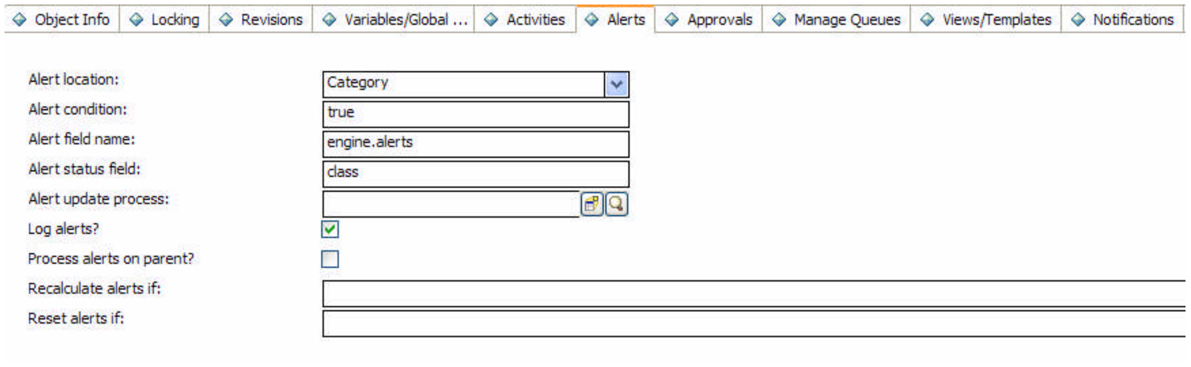
Require update if an activity record is NOT generated – Checkbox to indicate that an activity update is required.

Update field – Field or variable that contains the activity update on the form.

Display message – Message indicating that an activity update is required.

The Alerts Tab

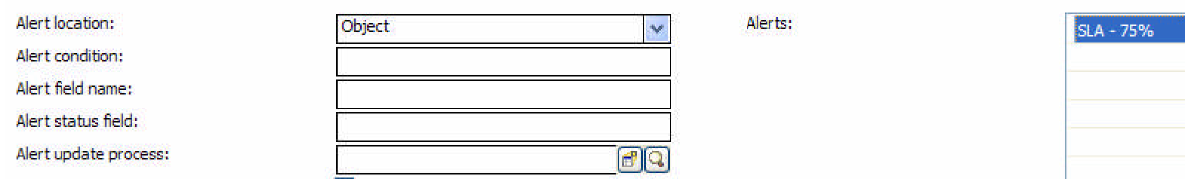
The Alerts tab is used to define where to set alerts and the overall condition for when to fire the alerts. Any Object that has a unique key can use these alerts.



Alert location:	Category
Alert condition:	true
Alert field name:	engine.alerts
Alert status field:	class
Alert update process:	
Log alerts?	✓
Process alerts on parent?	
Recalculate alerts if:	
Reset alerts if:	

Alert location - Specify the location to store alerts. You can enter one of the following:

- **Record**: Store alerts in the record itself.
- **Category**: Store alerts in the category file defined on the Object Info tab.
- **Phase**: Store alerts in the Phase record defined on the Object Info tab.
- **Object**: Store alerts in the Object record. When selected an Alerts array table will be displayed to enable you to fill in alert(s) to be used.



Alert location:	Object
Alert condition:	
Alert field name:	
Alert status field:	
Alert update process:	
Alerts:	SLA - 75%

Alert condition - Enter a logical condition to determine whether or not to process the alert. For example, **open in $L.file~=false**

Alert field name - Enter the field name that contains the actual alert name, as defined by **Alert Location**.

Alert Status Field - Enter the field in the current record in which to put the alert status, after the alert is processed.

Alert update process - Enter additional functions that the system will perform after the alert runs.

Log alerts? - If selected, then alerts are logged to the Alertlog file.

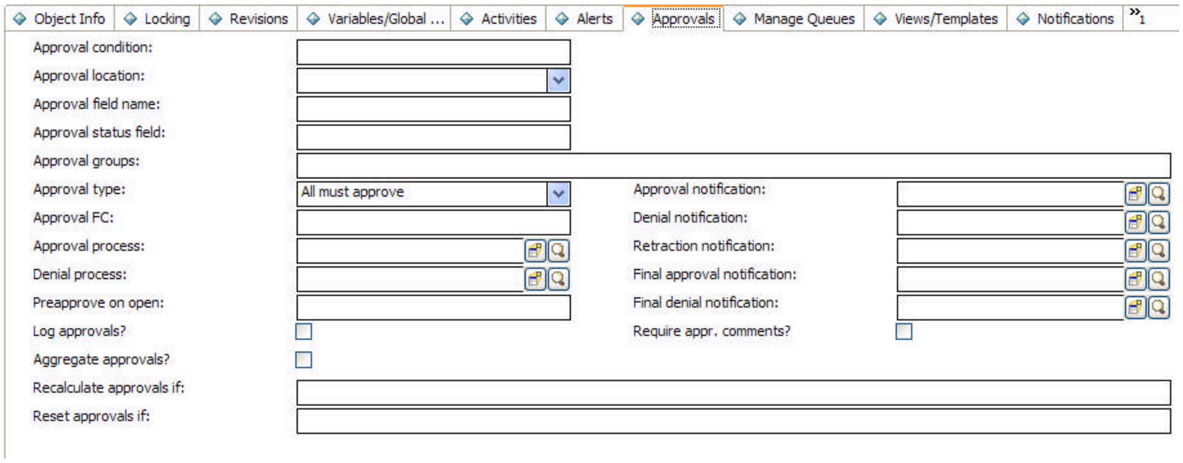
Process alerts on parent? - If you have selected the Locking Parent Record field on the Locking/Revisions tab and you select this checkbox, then when the alert is activated it will register against the parent record.

Recalculate alerts if - Conditions that determine whether to recalculate conditions on existing alerts.

Reset alerts if - Determines when to delete existing alerts and recalculate all conditions from scratch.

The Approvals Tab

The Approvals Tab is where approval and notification options are set for the Object. Approvals are defined in the ApprovalDef file.



Approval condition:
Approval location:
Approval field name:
Approval status field:
Approval groups:
Approval type: All must approve
Approval FC:
Approval process:
Denial process:
Preapprove on open:
Log approvals?
Aggregate approvals?
Recalculate approvals if:
Reset approvals if:
Approval notification:
Denial notification:
Retraction notification:
Final approval notification:
Final denial notification:
Require appr. comments?

Approval condition – Determines whether to use approvals.

Approval location – Indicates where the approval information is stored: record, phase, object, or category.

Approval field name - The field name that contains the actual approval name, as defined by Approval Location.

Approval status field – The field in which to store the approval status.

Approval groups - Stores a variable to contain the groups the current user must belong to in order to issue approvals for this Object.

Approval type - There are 4 pre-defined approval types:

- **All must approve**: The record is approved when all members of the approving group issue an approval.
- **One must approve**: The record is approved with one approval from any member of the approving group.
- **Quorum**: The record is approved as soon as a majority of the approving group indicates approval.

- **All must approve – immediate denial**: All approvers must approve the record. The first denial causes the status to change to Deny. Other approvers do not need to take any action.

Approval notification - Enter the notification that will run if the request was approved.

Approval FC - Enter the Format Control record to run upon approval.

Denial notification - Enter the notification that will run if one approver denied the request.

Approval process - Enter the Process that runs when the record is approved.

Retraction notification - Enter the notification that will run when retracting a previous action.

Denial process - Enter the Process that runs when the record is denied.

Final approval notification - Enter the notification that will be sent once the final approval was granted.

Preapprove on open - Determines whether the record should be automatically approved.

If the condition is true and the user belongs to one of the pending approval groups, the approval is processed automatically. If the user does not belong to one of the pending approval groups, the approval does not occur automatically and must then go through the regular approval process.

Final denial notification - Enter the notification that will be sent once the request was denied.

Log approvals? - Select this check box to log approvals.

Require appr. comments – If checked, approval comments are requested from the approver.

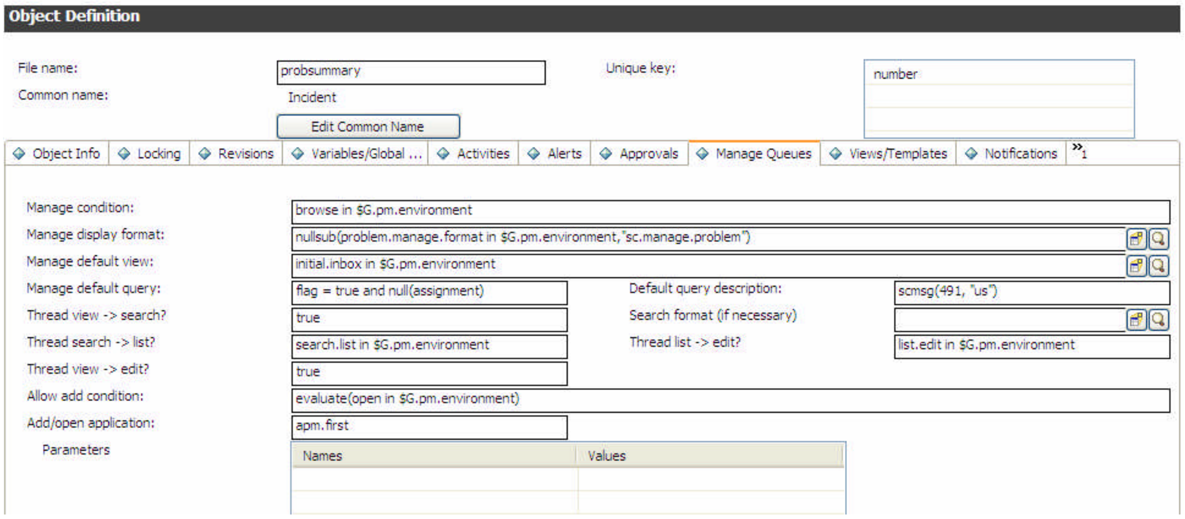
Aggregate approvals? – If checked, approvals will be cumulative.

Recalculate approvals if - Condition that determines whether to recalculate conditions on existing approvals.

Reset approvals if - Determines when to delete existing approvals and recalculate all conditions from scratch.

The Manage Queues Tab

The fields on the Manage Queues tab control Queues and how they are displayed as well as threading and who can create inboxes.



File name:	probsummary
Common name:	Incident
Unique key:	number
Manage condition:	browse in $G.pm.environment
Manage display format:	nullsub(problem.manage.format in $G.pm.environment,"sc.manage.problem")
Manage default view:	initial.inbox in $G.pm.environment
Manage default query:	flag = true and null(assignment)
Default query description:	scmsg(491, "us")
Thread view -> search?	true
Search format (if necessary)	
Thread search -> list?	search.list in $G.pm.environment
Thread list -> edit?	list.edit in $G.pm.environment
Thread view -> edit?	true
Allow add condition:	evaluate(open in $G.pm.environment)
Add/open application:	apm.first

Parameters

Names	Values

Manage condition - Enter a condition that allows only certain users to view queues. For example, **browse in $G.pm.environment**

Manage display format - This field determines which form to use to display the view. Out of the box Service Manager has a default display format: **sc.manage.generic** that is used if no other form is chosen. HP recommends you do not change the **sc.manage.generic** file.

Manage default view - State the default view for this queue. By specifying a user view for a particular user, a specific list of views can be set up for the HP Service Manager Manage queues. If a user does not have a record, the default user view is used.
Manage default query - Enter a default query to run if no default view was selected.

Default query description - Enter a description of the above field. You can associate a message with this field. For example, **scmsg(491, "us").**

Thread view -> search? - Enter true or an expression that evaluates to true to open a new thread when conducting a search.

Search format (if necessary) - Enter a default search format.

Thread search -> list? - Enter true or an expression that evaluates to true for a new thread when a user finds a list of records to view.

Thread list -> edit? - Enter true or an expression that evaluates to true for a new thread when a user selects a record to view out of a list of records.

Thread view -> edit? - Enter true or an expression that evaluates to true to open a new thread when the user views an existing record out of the queue.

Allow add condition - Enter an expression that evaluates an operator's ability to add a record.

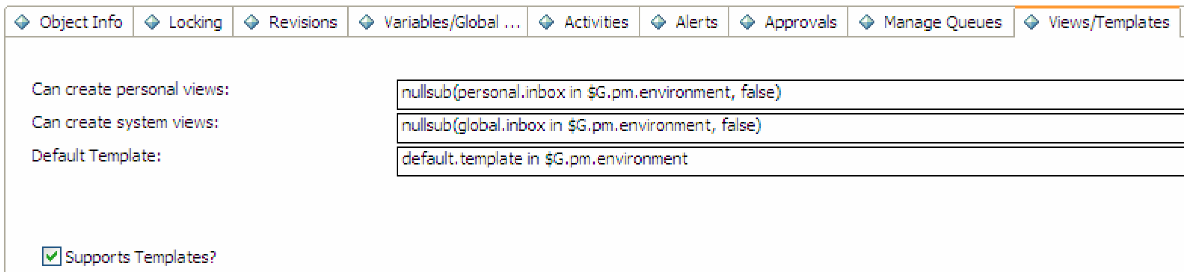
Add/open application - Fill in the name of the application to call when a record is added or opened.

Parameter Names - Enter the parameter names to be passed to the application specified in the **Add/open application** field.

Parameter Values - Enter parameter values to be passed to the application specified in the **Add/open application** field.

The Views/Templates Tab

The Views/Templates tab defines whether a user can create global and personal views.



Can create personal views: nullsub(personal.inbox in $G.pm.environment, false)
Can create system views: nullsub(global.inbox in $G.pm.environment, false)
Default Template: default.template in $G.pm.environment
✓ Supports Templates?

Can create personal views - A condition that evaluates to true or false to determine if the user can create personal views.

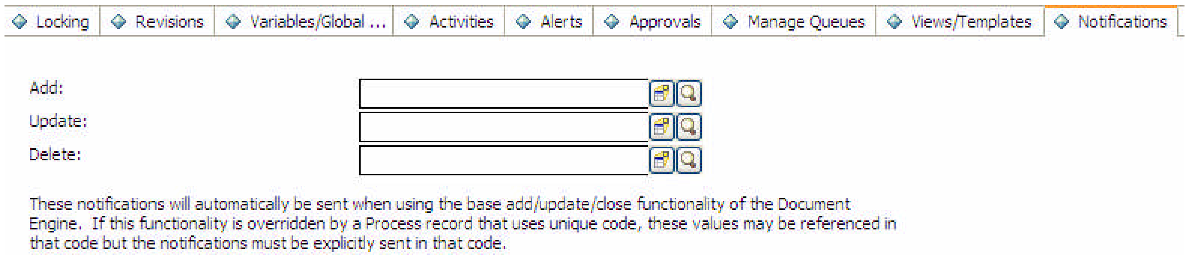
Can create system views - A condition that evaluates to true or false to determine if the user can create global views.

Default Template – Enter the default template to use.

Supports Templates? - Select this check box to enable the support of templates.

The Notifications Tab

Notifications entered here for add, update or delete will be automatically sent as defined on record add, update or deletes.



Add:	
Update:	
Delete:	

These notifications will automatically be sent when using the base add/update/close functionality of the Document Engine. If this functionality is overridden by a Process record that uses unique code, these values may be referenced in that code but the notifications must be explicitly sent in that code.

The Search Configuration Tab

The Search Configuration tab controls the available choices on the More Choices tab of the search screen.

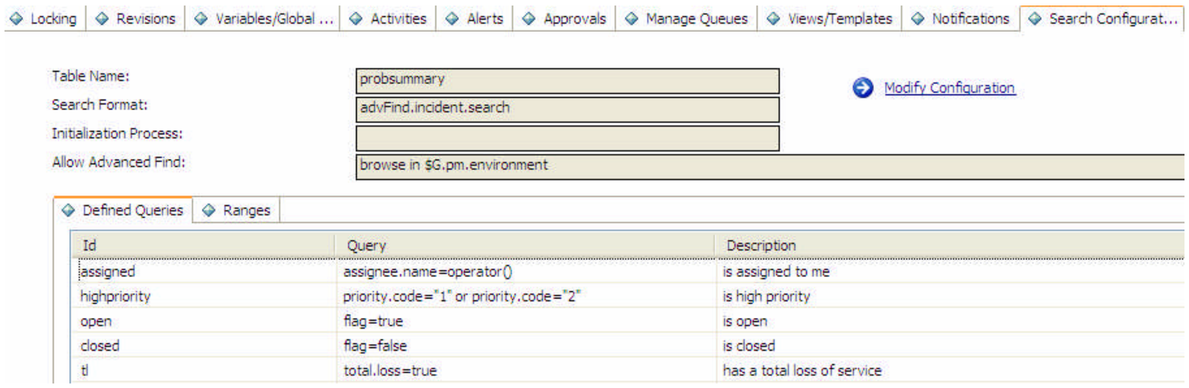


Table Name:	probsummary
Search Format:	advFind.incident.search
Initialization Process:	
Allow Advanced Find:	browse in $G.pm.environment

Id	Query	Description
assigned	assignee.name=operator()	is assigned to me
highpriority	priority.code="1" or priority.code="2"	is high priority
open	flag=true	is open
closed	flag=false	is closed
tl	total.loss=true	has a total loss of service

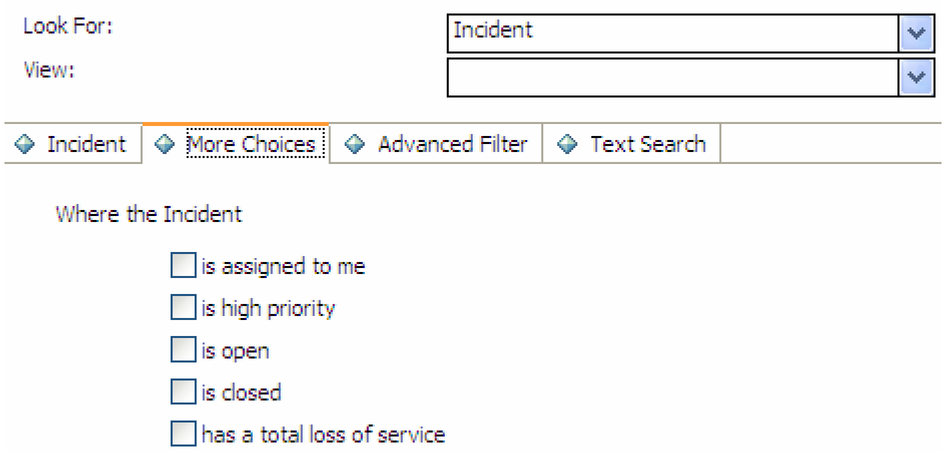
Table Name – The name of the table that will be queried.

Search Format – The name of the subformat that will be used for the More Choices tab.

Initialization Process – The name of a Process that will be run before displaying the search form.

Allow Advanced Find – Conditions that determine whether to allow Advanced Find.

Below is an example of the More Choices tab within the Incident search format:



Look For:	Incident
View:	

Where the Incident

- ☐ is assigned to me
- ☐ is high priority
- ☐ is open
- ☐ is closed
- ☐ has a total loss of service

The Ranges tab allows you to easily set up a search for begin and end date ranges. To do so define a variable each for begin and end to use as input in the form. Enter this variable in the Variable 1 column and use the Field and Operator 1 columns to define the query that will be executed.

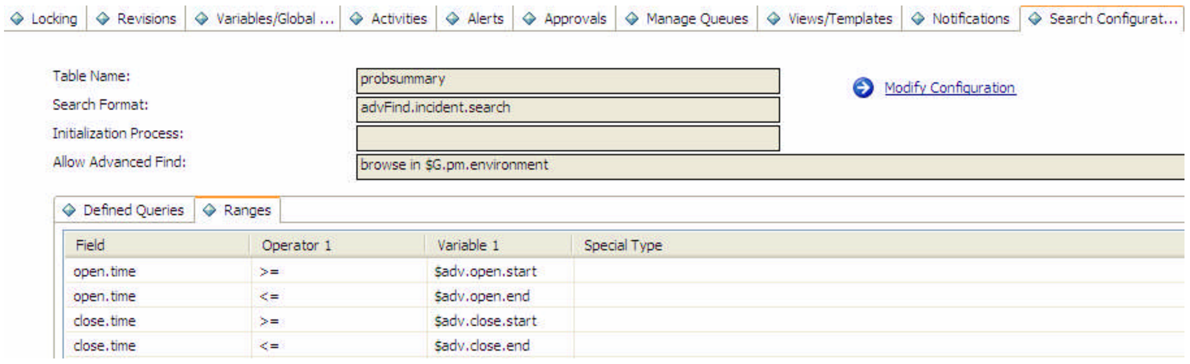
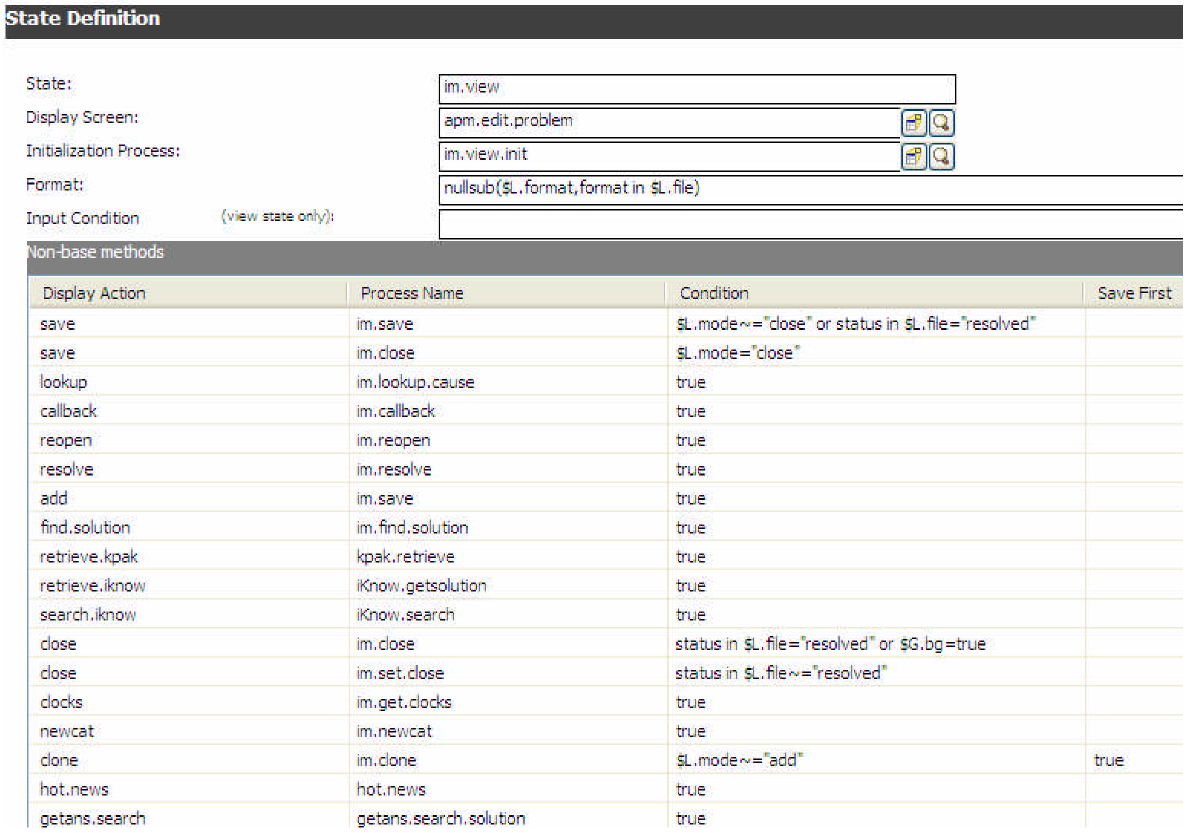


Table Name:	probsummary
Search Format:	advFind.incident.search
Initialization Process:	
Allow Advanced Find:	browse in $G.pm.environment

Field	Operator 1	Variable 1	Special Type
open.time	>=	$adv.open.start	
open.time	<=	$adv.open.end	
close.time	>=	$adv.close.start	
close.time	<=	$adv.close.end	

States

States are called by Objects and defined by Processes. The State record contains information on how a record looks and acts at a specific period in time.



State:	im.view
Display Screen:	apm.edit.problem
Initialization Process:	im.view.init
Format:	nullsub($L.format,format in $L.file)
Input Condition (view state only):	

Display Action	Process Name	Condition	Save First
save	im.save	$L.mode~="close" or status in $L.file="resolved"	
save	im.close	$L.mode="close"	
lookup	im.lookup.cause	true	
callback	im.callback	true	
reopen	im.reopen	true	
resolve	im.resolve	true	
add	im.save	true	
find.solution	im.find.solution	true	
retrieve.kpak	kpak.retrieve	true	
retrieve.iknow	iKnow.getsolution	true	
search.iknow	iKnow.search	true	
close	im.close	status in $L.file="resolved" or $G.bg=true	
close	im.set.close	status in $L.file~="resolved"	
clocks	im.get.clocks	true	
newcat	im.newcat	true	
clone	im.clone	$L.mode~="add"	true
hot.news	hot.news	true	
getans.search	getans.search.solution	true	

Fields on the States panel

State - Enter the name of the State.

Display Screen - Enter the Display screen to associate with the State.

Initialization Process - Enter the name of a Process to run prior to entering the State.

Format - Enter the Format that the record will be displayed in. This format can be stored in a record, be a variable or hard coded.

Input Condition - The Input Condition determines whether or not the record is read only. Enter False for read only, otherwise enter True.

Non-base methods:

Display Action- The action parameter that comes from a display action after input from a user.

Process Name - The Process that the State will call.

Condition - Enter an expression that has to evaluate to true for the Process to be called.

Save First - Choose to run the save Process before you run this Process. Enter **true** to save first otherwise enter **false**. The default is false.

Creating and Updating States

To create a new State:

- Access the Document Engine. See Accessing the Document Engine on page 2 for steps.
- On the State Definition form, fill in the fields required to create a State that will perform the functions you desire. See the field descriptions earlier in this chapter.

To modify an existing State:

- Access the Document Engine. See *Accessing the Document Engine* on page 2 for steps.
- Enter the name of the State you want to modify in the State field or press **Search** to search for the State.

Processes

Processes are the smallest discreet units of work available to the Document Engine and are the level where the data is manipulated. Users can create their own Process or use one of the over 700 Processes that ship with Service Manager. The Process panel consists of pre-RAD, RAD, and post-RAD expressions, entered on the Initial Expressions, RAD, and Final Expressions tabs, respectively.

Expressions are written using standard Service Manager expressions.

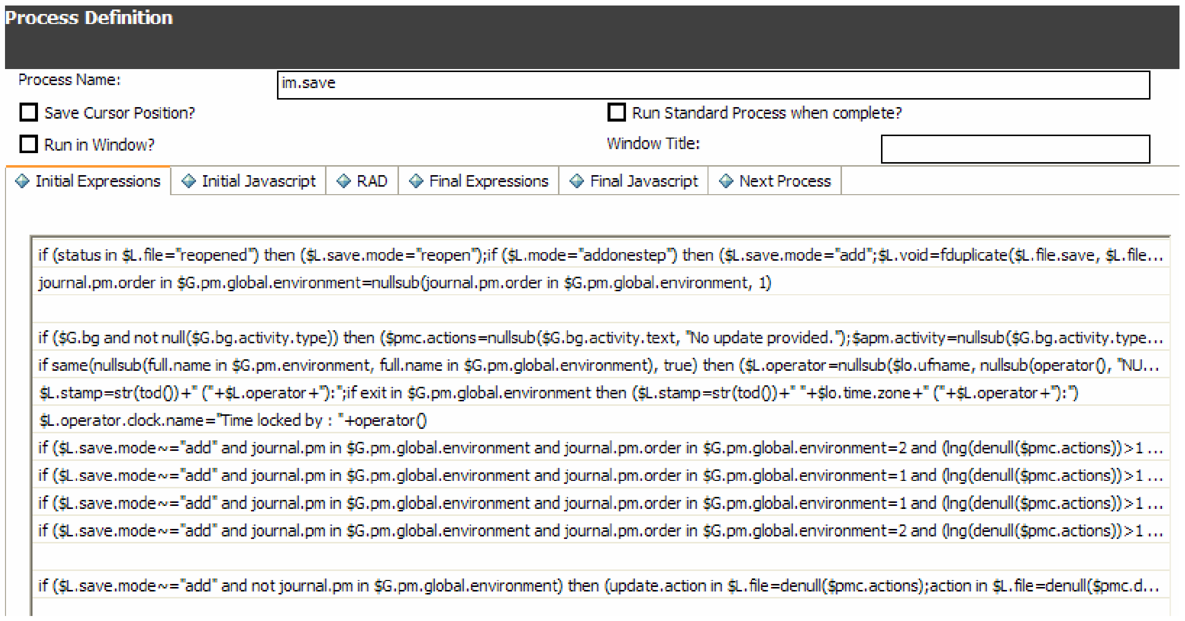
Creating Processes

To create a Process:

- Access the Document Engine. See Accessing the Document Engine for steps.
- Using the tabs on the Process panel, fill in the fields required to create a Process that will perform the functions you desire. See the field descriptions later in this chapter.

To modify an existing Process:

- Access the Document Engine.
- Enter the name of the Process you want to modify in the Process Name field or press Search to search for the Process.



Process Definition

Process Name: im.save

☐ Save Cursor Position? ☐ Run Standard Process when complete?

☐ Run in Window? Window Title:

Initial Expressions | Initial Javascript | RAD | Final Expressions | Final Javascript | Next Process

```
if (status in $L.file="reopened") then ($L.save.mode="reopen");if ($L.mode="addonestep") then ($L.save.mode="add";$L.void=fduplicate($L.file.save, $L.file...
journal.pm.order in $G.pm.global.environment=nullsub(journal.pm.order in $G.pm.global.environment, 1)

if ($G.bg and not null($G.bg.activity.type)) then ($pmc.actions=nullsub($G.bg.activity.text, "No update provided.");$apm.activity=nullsub($G.bg.activity.type...
if same(nullsub(full.name in $G.pm.environment, full.name in $G.pm.global.environment), true) then ($L.operator=nullsub($lo.ufname, nullsub(operator(), "NU...
$L.stamp=str(tod())+" ("+$L.operator+"):";if exit in $G.pm.global.environment then ($L.stamp=str(tod())+" "+$lo.time.zone+" ("+$L.operator+"):")
$L.operator.clock.name="Time locked by : "+operator()
if ($L.save.mode~="add" and journal.pm in $G.pm.global.environment and journal.pm.order in $G.pm.global.environment=2 and (lng(denull($pmc.actions))>1 ...
if ($L.save.mode~="add" and journal.pm in $G.pm.global.environment and journal.pm.order in $G.pm.global.environment=1 and (lng(denull($pmc.actions))>1 ...
if ($L.save.mode~="add" and journal.pm in $G.pm.global.environment and journal.pm.order in $G.pm.global.environment=1 and (lng(denull($pmc.actions))>1 ...
if ($L.save.mode~="add" and journal.pm in $G.pm.global.environment and journal.pm.order in $G.pm.global.environment=2 and (lng(denull($pmc.actions))>1 ...

if ($L.save.mode~="add" and not journal.pm in $G.pm.global.environment) then (update.action in $L.file=denull($pmc.actions);action in $L.file=denull($pmc.d...
```

The Process format

The Process format is where you define new Processes or edit exiting Processes. There are four fields not associated with a tab:

Process Name - The name of the process.

Save Cursor Position – Check this box if you want to return to the same cursor position after the action (for example on a fill)

Run Standard Process when complete? - If selected, then the system runs a standard process after completing the current action or Process. A standard process could be **save**, for example. If you have created a save process and want to run the save process that comes with the Document Engine after completing the save process you defined, check this box.

Run in Window – If checked, the process runs in a separate window.

Window Title - If the Run in Window? check-box is checked, enter a title for the window.

The Initial Expressions tab

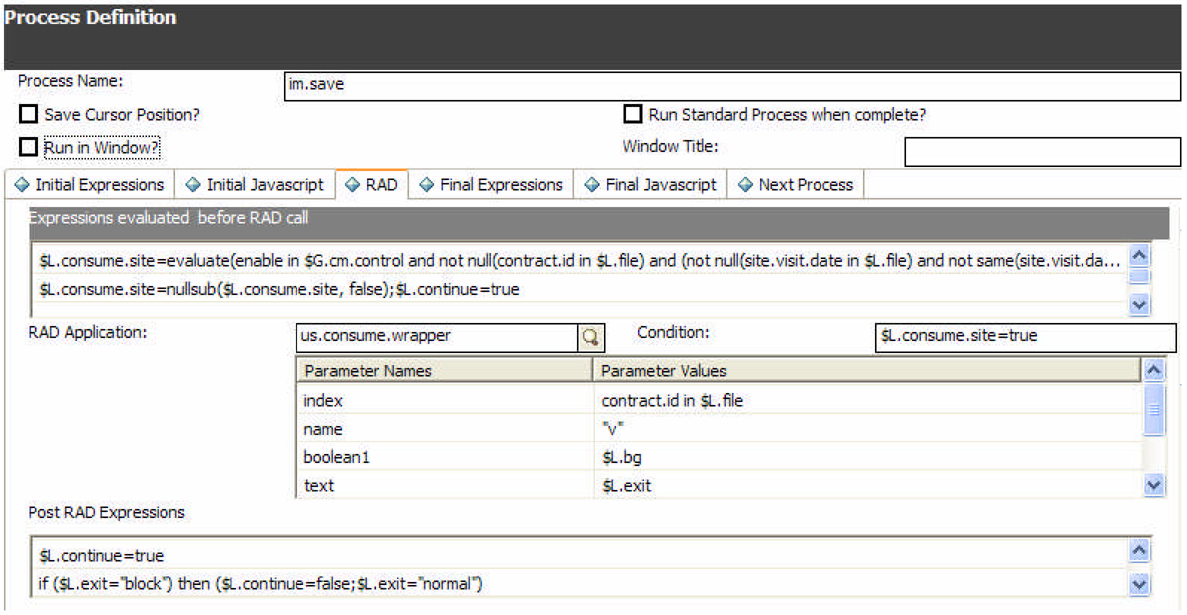
Initial expressions run prior to the RAD code defined on the RAD tab and are written using standard Service Manager expressions. Enter any initial RAD expressions on the first available line of the table.

The initial JavaScript tab

Initial JavaScript expressions that are run before the RAD tab.

The RAD tab

The RAD tab contains an array of pre-RAD expressions, RAD calls and post-RAD expressions:



Process Definition

Process Name: im.save

☐ Save Cursor Position? ☐ Run Standard Process when complete?

☐ Run in Window? Window Title:

Initial Expressions | Initial Javascript | RAD | Final Expressions | Final Javascript | Next Process

Expressions evaluated before RAD call

```
$L.consume.site=evaluate(enable in $G.cm.control and not null(contract.id in $L.file) and (not null(site.visit.date in $L.file) and not same(site.visit.da...
$L.consume.site=nullsub($L.consume.site, false);$L.continue=true
```

RAD Application: us.consume.wrapper Condition: $L.consume.site=true

Parameter Names	Parameter Values
index	contract.id in $L.file
name	"v"
boolean1	$L.bg
text	$L.exit

Post RAD Expressions

```
$L.continue=true
if ($L.exit="block") then ($L.continue=false;$L.exit="normal")
```

Expressions evaluated before RAD call - Enter an expression to run prior to the RAD application defined immediately after.

RAD Application - Enter the name of the RAD application to run.

Condition - Enter a condition associated with the RAD Application field.

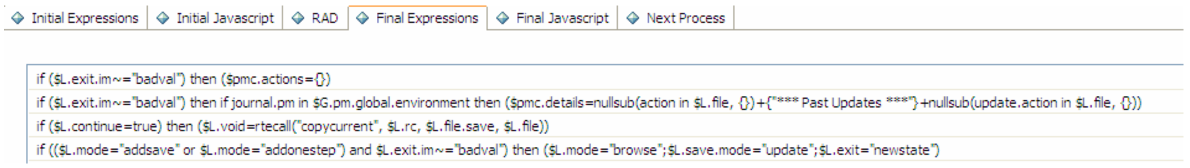
Parameter Names - Enter the parameter names to be passed to the RAD application.

Parameter Values - Enter parameter values to pass to the RAD application.

Post RAD Expressions - Enter a RAD expression that will run upon completion.

The Final Expressions tab

Final expressions run after this process is complete and are written using standard Service Manager expressions.



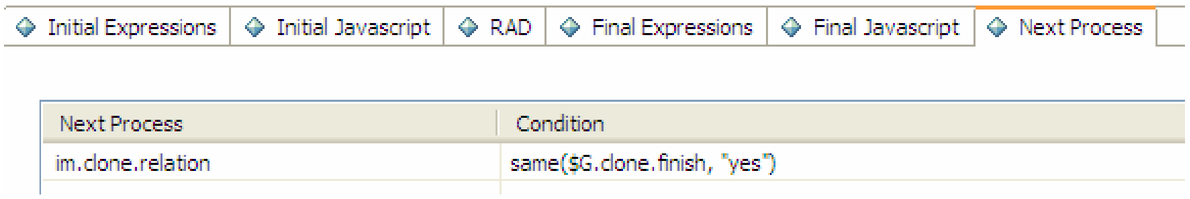
Initial Expressions | Initial Javascript | RAD | Final Expressions | Final Javascript | Next Process

```
if ($L.exit.im~="badval") then ($pmc.actions={})
if ($L.exit.im~="badval") then if journal.pm in $G.pm.global.environment then ($pmc.details=nullsub(action in $L.file, {})+{"*** Past Updates ***"}+nullsub(update.action in $L.file, {}))
if ($L.continue=true) then ($L.void=rtecall("copycurrent", $L.rc, $L.file.save, $L.file))
if (($L.mode="addsave" or $L.mode="addonestep") and $L.exit.im~="badval") then ($L.mode="browse";$L.save.mode="update";$L.exit="newstate")
```

The Final Javascript tab

JavaScript defined on this tab will be run after the final RAD expressions and after the RAD applications on the RAD tab.

The Next Process tab



Initial Expressions | Initial Javascript | RAD | Final Expressions | Final Javascript | Next Process

Next Process	Condition
im.clone.relation	same($G.clone.finish, "yes")

Next Process - Enter the Name of the next Process to run, if applicable.

Condition - Enter a condition associated with the Next Process field.

Technical Information

Document Engine usage in the current system

Database Manager

Any file that is accessed by the Database Manager will automatically use the rules and processes defined for the corresponding Object. If no Object has been defined for the file, it will use the DEFAULT Object. The DEFAULT Object was defined to duplicate the functionality of the old Database Manager.

A System Administrator may also access a file that has a corresponding Object via the DEFAULT Object. This is accomplished by accessing the file through Database Manager with the "Administration Mode" checkbox marked. This checkbox will only be available for System Administrators.

The Document Engine is always using environment profiles. In the old system, a System Administrator was granted all rights (add, update, delete, etc.), regardless of the format control settings. Using the Document Engine, the rights granted will match those defined in the format control record. Administrators that prefer the old method of granting rights can simply modify the "Profile Application" setting in the DEFAULT Object record from "db.environment" to "db.environment.sysadmin".

Main RAD Applications and Flow

When working with a database driven application, there are three basic record sets that the user will be viewing at any one time:

- Zero records – when searching for information.
- Many records – when looking at a list of information.
- One record – when making changes to a single record.

The three main RAD applications used by the Document Engine mirror this idea.

- se.search.engine

The search engine is used when there are no records being viewed. The main purpose of this routine is to formulate a query and select records from the correct table. This routine may also be used for the initial entry of information into a blank record for the purpose of adding a new record to the database.

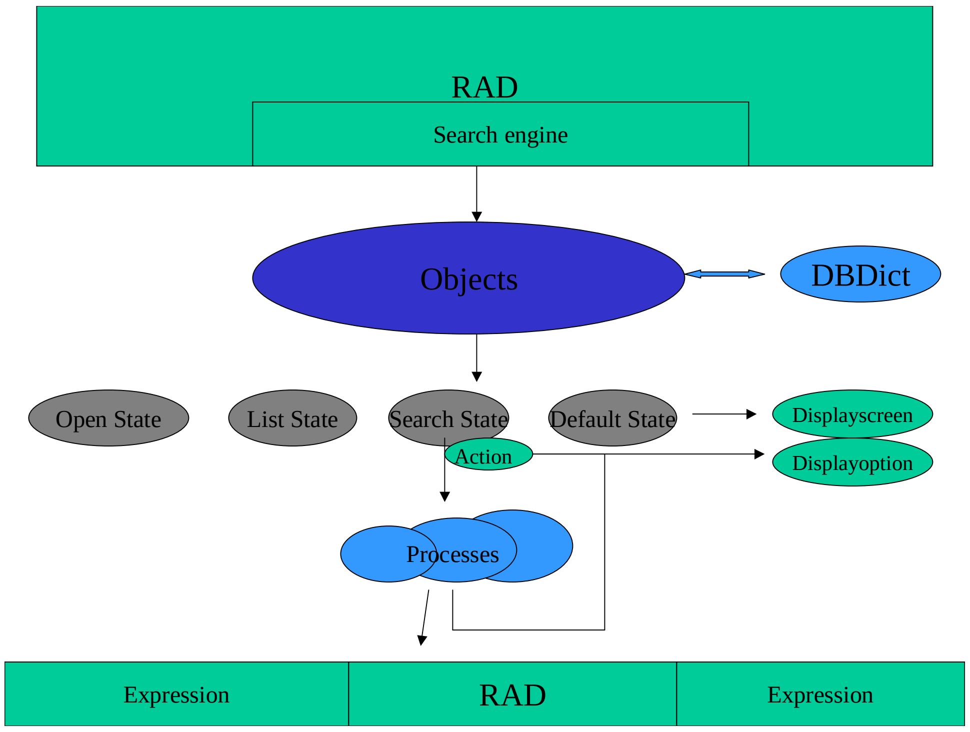
- se.list.engine

The list engine is used to display multiple records. Using the list engine a user may select a specific record from the list, or perform actions on the entire list of records.

- se.view.engine

The view engine is used to display a single record. This application is used to perform actions against a specific record, such as updates or deletes. The RAD routine used is simply determined by the number of records being acted on at any time.

Note: When using Service Manager's record list functionality, the view engine is used for both the list and single record information



When a file is displayed using the Document Engine it will always be using one of the three applications specified earlier in this document. The display screen, display options, and format are determined by the current State of the record. When a display option is triggered, after the standard display functionality is performed, the engine checks the "action" of that option against the available processes defined in the current State record. If there is a process defined that has a condition that evaluates to true, the engine then performs that process against the current record (or record set). If the action is not defined in the State record, the engine will then check to see if that action is defined as a "Base Process" in the current application. If it is, then the system will perform that base process against the record. If not, no action is taken.

The State record that is used depends on what how many records the user is viewing, as described below:

Searching

When there are no records in the current file variable, it is assumed that the user is in "search" mode. The State that is used is the "Search State" defined in the file's Object record. The default search state is "db.search".

QBE Lists

When viewing a list of records without the record list functionality being used, the "List State" defined in the file's Object record is used. The default list state is "db.list".

Viewing a single record

When viewing a single record, the record is first checked to see if there is a "State" field in the dbdict. If the field exists and is populated, the contents of that field will be used as the current State of the record. If this field does not exist or is NULL, the "Default State" defined in the file's Object record is used. The default value for the default state is "db.view".

Browsing a record

When browsing a record in read-only mode, the "Browse State" defined in the file's Object record is used. Note that only files that use locking need a browse state. Although viewing a record without having update rights looks similar to the browse screen, it is using the default state, not the browse state. There is no default browse state.

Integration tips for display application

If a display screen does not have any display events associated to it, the system automatically uses the se.default display event that performs se.lock.object in case of “OnFormModified”.

If a display option record has the “Modifies Record” check box activated, the record is locked when the button is pressed.

Available Local Variables

The following is a list of Standard Variables that are usually available when using Document Engine:

$L.mode – the mode the viewed record is in, typically add to create a new record, update to modify an existing record or close to finish processing of an existing record.

$L.format – name of the format used to display the record

$irspread – determines the IR discovery options: 0=shallow search, 2=deep search, 4=complete match

$L.action - the display action value from the display option.

$L.sql or $L.query- the current query

$L.sort - the current sort order

$L.exit - internal exit parameter

$L.file - The current file variable

$L.mult - Flag that is true if there are multiple records in the $L.file variable

$L.env - The current environment record

$L.object - The object record

$L.file.save - A copy of the record in its original state

$L.category - The category record (if available)

$L.phase - The phase record (if available)

$L.bg - Background flag

$L.state – The state record the system is using (=the state the record is in)

Variables that are available in View mode (when viewing a single record)

$L.fc – copy of the detail FormatControl record

$L.fc.master – copy of the master FormatControl record

Base Functions

The following functions are built into the Document Engine and may be used by any Object (or overwritten in the State record to call a different Process).

Implemented in the se.search.engine

Default state: db.view

Display Action	Description
back	Exits the engine

find	Standard Service Manager find functionality
fill	Standard Service Manager fill functionality
advanced	Starts the Advanced Search process
clear	Clears the current screen
openinbox / inbox	Prompts the user to open a view associated with the current file
search	Performs a standard "genquery" (query by example) search.
add	Performs "add" format control and attempts to add a record to the database.
restore	Restores the contents of the screen (after a "clear" is performed)
irquery	Starts the IR Query process
validitylookup	Performs standard Service Manager validity lookup functionality
expandarray	Performs standard Service Manager expand array functionality
reset	Resets the current file
regen	Regens the indexes of the current file
export/unload	Starts the "no records" export/unload process.
views	Displays format control views
findrevision	Displays revisions for records in this object
initrevision	Creates a revision of a record in this object

Implemented in the se.list.engine

Default State: db.list

Display Action	**Description**
exit (or back)	Returns to the search (or calling) screen
inbox.save / inbox	Prompts the user to save the current query as a new view
count	Performs standard count functionality
refresh	Refreshes the list using the current query
big.green	"Big Green Arrow" – completely exits the current module
print	Prints the list or record set displayed
views	Standard Service Manager alternate forms functionality
export/unload	Standard Service Manager export/unload functionality
massadd	Standard Mass Add functionality
massupdate	Standard Mass Update functionality
massdelete	Standard Mass Delete functionality

Implemented in the se.view.engine

Default State: db.view

Display Action	**Description**
save	Performs update format control and attempts to update the record

add	Performs add format control and attempts to add a new record
ok	If the record has changed, perform a save, otherwise exit
reselect	Reselects the current record from the database (in case it was changed)
fill	Standard Service Manager fill functionality
find	Standard Service Manager find functionality
next	Move to next record (after checking for changes)
previous	Move to previous record (after checking for changes)
back	Back to list or search screen
menu	Exit the module completely
delete	Performs delete format control and attempts to delete record
views	Standard alternate forms functionality
print	Prints the current record
printlist*	Prints the current list of records or record list
validitylookup	Standard validity lookup functionality
export/unload	Standard Service Manager export/unload functionality (single record)
massunload*	Standard Service Manager export/unload functionality (record list)
massadd*	Standard Mass Add functionality
massupdate*	Standard Mass Update functionality
massdelete*	Standard Mass Delete functionality
irquery	Standard IR Query functionality
expandarray	Standard expand array functionality
count*	Counts records in the record list
audithistory	Calls the standard audit history routine
inbox.save / inbox*	Saves the current query as an inbox
approval.log	Views the approval log for the current record
alert.log	Views the alert log for the current record
alerts	Views the current and scheduled alerts for the current record
pagelist / listpages	Views the pagelist for the current record
update	If in text mode, changes from "browse" to "edit" mode.
clocks	Display clocks records for this record
xmlfill	Handles XML fields, such as the user options in Service Catalog

* These commands only apply if the record list functionality is being used.

Troubleshooting

To successfully troubleshoot the Document Engine you will need to gather the following information:

1. What dbdict / Object is being used?
2. What State is that record in?

3. What was the Process being called?
4. Steps to reproduce (STR)

Researching the application path through Document Engine

In troubleshooting the Document Engine, as with troubleshooting any Service Manager application, enter **RTM:3** and **debugdbquery:999** in the Service Manager `sm.ini` and then start a new client connection. Unless this user process is the very first to invoke the Document Engine processes to debug, it will not show the selection of the State or Process records in the `sm.log` file when doing this trace, but it will give helpful hints as to which Process was being invoked.

To answer the question concerning which dbdicts or Object is being used, search the log file for the following:

```
1320 07/18/2006 11:00:36 RADTRACE  20 [ 1] se.get.object  get.object
select  CPU(    0   1411 )
1320 07/18/2006 11:00:36 (0x0129AC08) DBACCESS - Cache Find  against file
Object found 1 record, query: file.name="pcsoftware"
1320 07/18/2006 11:00:36 RADTRACE     20 [ 1] se.get.object  set.access
process  CPU(    0   1411 )
```

The next question to look at is which State the record is in. To get that information, search for the following in the trace sm.log:

```
1320 07/18/2006 11:00:48 RADTRACE  10 [ 1] se.get.state  select.state
select  CPU(    0   1491 )
  1320 07/18/2006 11:00:48 (0x01292FB0) DBACCESS - Cache Find   against
file States found 1 record, query: state="pcs.list"
1320 07/18/2006 11:00:48 RADTRACE  10 [ 1] se.get.state  exit.normal
process  CPU(    0   1491 )
```

The name of the Process can be found as well, by searching for

```
1320 07/18/2006 11:00:50 RADTRACE  20 [ 1] se.call.process
select.process  select  CPU(    0   1542 )
1320 07/18/2006 11:00:50 (0x00B56810) DBACCESS - Cache Find   against
file Process found 1 record, query: process="upgrade.pcs"
1320 07/18/2006 11:00:50 RADTRACE  20 [ 1] se.call.process  run.pre.exp
process  CPU(    0   1542 )
```

in the sm.log containing the trace.

Researching application errors

Processes then call a number of RAD applications and execute a number of expressions, with a possibility of invoking more Processes afterwards. If any of the applications or expressions caused an error exit due to wrong syntax or wrong logic, this information can be found in the sm.log file as well:

```
Process panel run.pre.exp in RAD se.call.process encountered error in
line 1 (se.call.process,run.pre.exp)
Cannot evaluate expression (se.call.process,run.pre.exp)
Cannot evaluate expression (se.call.process,run.pre.exp)
Bad arg(2) oper = (se.call.process,run.pre.exp)
Cannot evaluate expression (se.call.process,run.pre.exp)
Cannot evaluate expression (se.call.process,run.pre.exp)
Bad arg(2) oper = (se.call.process,run.pre.exp)
Cannot evaluate expression (se.call.process,run.pre.exp)
Bad arg(2) oper nullsub (se.call.process,run.pre.exp)
Cannot evaluate expression (se.call.process,run.pre.exp)
Bad arg(2) oper  in  (se.call.process,run.pre.exp)
Cannot evaluate expression (se.call.process,run.pre.exp)
Unrecoverable error in application:  se.search.objects on panel
call.list.engine
```

```
Unrecoverable error in application:  se.list.engine on panel
call.process.1
Unrecoverable error in application:  se.call.process on panel run.pre.exp
```

In this example (not out of box) the error occurred in se.call.process, run.pre.exp, or in other words, while evaluating the initial expressions of the Process. To find out which Process was causing the issue, go through the steps outlined above and note the process from the line
DBACCESS - Cache Find   against file Process found 1 record, query: process="upgrade.pcs"
Go to the Process record by the name of upgrade.pcs and check for any statements on the initial expressions tab. In this specific case, the expression will include the word **nullsub**. For example, the expression in question for this test may be

$L.icount=nullsub($L.icount, anynumberIwant)

anynumberIwant is not a valid field, literal or variable, so it will have to be changed to prevent this issue.

Another useful troubleshooting technique is to print out values of variables or results of expressions. You can use the JavaScript print() function or use $L.void=rtecall("log", $L.rc, "message") in the RAD expressions.

For more information

Please visit the HP Management Software support Web site at:

http://www.hp.com/managementsoftware/support

This Web site provides contact information and details about the products, services, and support that HP Management Software offers.

HP Management Software online software support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued customer, you can benefit by being able to:

- Search for knowledge documents of interest
- Submit and track progress on support cases
- Submit enhancement requests online
- Download software patches
- Manage a support contract
- Look up HP support contacts
- Review information about available services
- Enter discussions with other software customers
- Research and register for software training

Note: Most of the support areas require that you register as an HP Passport user and sign in. Many also require an active support contract.

To find more information about support access levels, go to the following URL:

http://www.hp.com/managementsoftware/access_level

To register for an HP Passport ID, go to the following URL:

http://www.managementsoftware.hp.com/passport-registration.html