WORKFLOW ASCENDANT

6.1

DEVELOPER'S GUIDE

Concerning Horizon Ascendant, Inc.

Horizon Ascendant is a software publishing company software whose flagship product is Workflow Ascendant, an all-in-one design template which allows individuals with minimal programming experience to develop robust workflow applications in an IBM Notes or Web 2.0 / Responsive environment. Horizon Ascendant also offers a range of services around that software package including customer development, training and consulting.

www.horizonascendant.com

Copyrights and Trademarks

Chapter 1 - Introduction

1.1) General

Workflow Ascendant is an all-in-one design template which allows individuals with minimal programming experience to develop robust workflow applications in a responsive web environment. Anchored in a coherent methodology, this development model has been refined over the years through extensive experience in working with large companies in an international environment. IBM Notes developers can leverage their existing expertise with minimal knowledge in web technologies. Non IBM Notes developers can easily adapt to Workflow Ascendant's "plug and play" component approach. End-users can intuitively navigate workflow documents.

All aspects of a process are defined via easily configurable documents. This includes document states, routing rules, notifications, time triggers, etc. There are no "black boxes" to deal with, providing complete visibility and direct control over every aspect of the workflow. The source code of Workflow Ascendant is exposed.

Workflow Ascendant is the proprietary software owned by Horizon Ascendant Inc. Use of this Software and derived applications from the Software are subject to the terms and conditions of the Horizon Ascendant Master Subscription License Agreement. Please review the terms of this document carefully. By using the Software and/or derived applications, you agree to all of the terms contained therein. Briefly here, applications developed using Workflow Ascendant are licensed on a subscription basis per IBM Notes Storage Facility database instance: ***95 € / nsf file / month***.

We highly recommend that you view the various presentation and training videos made available on our web site: *www.horizonascendant.com*. These videos demonstrate and explain in great detail the various mechanisms of Workflow Ascendant and how they affect application behavior. After viewing these, you should have a firm grasp of the methodology which underlies this workflow development tool.

All the materials needed to get you going are available on the Horizon Ascendant web site. This includes the basic Workflow Ascendant template; videos along with the actual applications used therein, additional sample applications as well as a complete Domino environment to run those applications. To note that provision of the latter in no way confers licenses to use that software. It is made available only to provide you a jump start – you will need to obtain corresponding licenses from the IBM Corporation for applications put into production.

Use this guide as a reference. In addition to systematically presenting Workflow Ascendant, it has been designed to provide easy access reminders to assist you in the context of you developing your workflow applications.

Our well wishes to you in your development efforts from all of us here at Horizon Ascendant!

.

1.2) Benefits

Workflow Ascendant allows you to rapidly create Business Process Management applications with the latest web functionality *USING A COHERENT AND COMPREHENSIVE METHODOLOGY*.

- **Minimal end-user training costs**. Avoid the "not *another application to learn*" syndrome. Once users learn how to use one application developed with Workflow Ascendant, they know how to use any new ones later put in place as they all operate by the same principles: only the data/content changes.
- **Minimal maintenance costs**. Workflow Ascendant developed applications are designed such that volatile elements (the process, email notifications, allocating specific users, etc.) are configured in easily understood documents (by version!) while more stable and complex elements (code) are referenced as stand-along modules in a centralized code library. And as is true for end-users, a system administrator need only understand how to manage one of these applications in order to manage any new ones.
- **Minimal development costs**. *Workflow Ascendant has been designed to leverage existing, traditional IBM Notes developer skills.* Develop fast and efficiently. Seeing is believing. We invite you to take a look at the various presentation videos contained on our web site.

A few specific benefits to cite:

- Routing Conditions. As rules are rooted in a methodology - mix and match them to suite your needs: manual and automatic routing; selective, parallel, inclusive, extended parallel and extended sequential operations; automatically create child documents; route documents based on time events, etc.
- **Workflow Actions**. Send personalized emails, launch agents and update document fields based on document state changes.
- Reference Allocation. Automatically assign unique and sequential references to documents.
- **Counters / Delays**. Automatically send emails and/or update document fields when a trigger is activated or deactivated.
- Real-Time Delegation. Users can designate replacements (while away from the office, for example). Control is automatically returned to the original users after the designated period.
- Intervention History. Each intervention is automatically recorded along with the number of days spent at that step in the process.
- **Display Statistics Graphically**. Display Bar Charts and Gantt Charts for your end-users via plug-and-play components.
- **Version Control**. Migrate easily from one workflow version to another. All existing documents adhere to the previous process while new documents follow the new workflow.
- **Multi-lingual end-user environment**. Change languages with the click of a button.
- **Multi-date format**. Force all dates to appear in a specific format (American or European) - regardless of the date parameters set on user's PCs.
- **Rich Test Environment**. Simulate the passing of time via built-in agents to test your workflow definitions. Variable logging levels allow you to monitor every aspect of the workflow process.
- **Integrated Responsive Web Design**. Automatically adjusts to the characteristics of the device being used (mobile phone, tablet, desktop), making your Workflow Ascendant applications out-of-the-box ready for mobile access.

1.3) Principles

Processes

- An IBM Notes Form contains all the data associated with a workflow document and corresponds to a collection of State (configuration) documents which define the process.
- Each State document has a unique name (identifier) in the context of a given process.
- State documents are connected to each other via a Current State – Next State relationship.
- The behavior of a given workflow document is dictated by the definitions contained in the corresponding State document while in that state.
- The State document entitled *$created$* is always the first state in the workflow.
- A State document which has no Next State is known as a Terminal State (archived).
- Each State document is comprised of a set of rules, each of which is associated with a given Constraint.
- Constraints refer to a profile of those who can intervene in a workflow document at that state.
- Workflow documents advance in the process when a user selects the *Send* button.
- A workflow document will be automatically archived when the *Send* button is selected if the following state is a Terminal State or if there are no valid users defined for the next state (which constitutes an undefined error condition).

Constraints

- Constraints refer to the profiles of those who can intervene in a workflow document at a particular state.
- A Constraint consists of either a role name (in the initial state only) or a field name (in all other states).
- In a workflow document, the corresponding field of a State document constraint must resolve to a list of user names.
- The aforementioned list of user names are the only individuals who can intervene in the workflow document at that particular state.
- Workflow Ascendant provides several mechanisms by default to facilitate the assignation of individual user names to the various roles and fields of a workflow application.

Content

- Who can modify a workflow document at a given point in time is dictated by the constraints defined in the corresponding State document.
- In addition, Workflow Ascendant provides a number of mechanisms by which to define *which parts* of a document the aforementioned individuals can modify.
- Information in workflow documents can be initialized and controlled by routines referenced in the various State documents.
- Routines which initialize and control data in the workflow documents, some of which are supplied by default and others supplied by the workflow developer, are contained in a centralized code library.

1.4) Roles and Reserved Names

ROLES

- **[WAArchive]**. Workflow Ascendant automatically assigns the current user (field *WACurrentAuthors*) to this role when a document arrives at a terminal state.
- **[WAListRI]**. Those individuals who are responsible for managing lower level configuration documents such as lists accessed in the workflow documents.
- **[WAManager]**. Those individuals who are responsible for managing the database. By default, the administrative portion of Workflow Ascendant is only visible to users assigned to this role.
- **[WARefAlloc]**. Workflow Ascendant automatically assigns the current user (the server specified in the active **Language** document) to this role when a document is awaiting reference allocation. This is the case if you are working in a distributed environment and the replica you are working with does not reside on that server.
- **[WASupervisor]**. Those individuals who can see all documents.

FIELD NAMES

- **WAAuthor**. The user name of the individual who created the workflow document.
- **WACurrentAuthors**. The list of user names who can intervene in the document at that moment in time.
- **WACurrentAuthorsDisplay**. The list of current user names displayed to the end users in the various views.
- **WACurrentState**. The name of the workflow document's current state.
- **WADocRef**. Contains the unique reference allocated to the workflow document.
- **WAErrorMessage**. Contains any error messages which block advancing the document in the workflow.
- **WAFormName**. The process name displayed to end users.
- **WAHistoryAuthors**. The list of users who have intervened in the workflow document to date.
- **WAReaders**. The list of users who can visualize the workflow document (in addition to those who have or who can modify the workflow document).
- **WAVersionRef**. The version of the workflow document.

1.5) Recommended Practices

- **Information Completeness**. A workflow document should contain all the information necessary for a responsible individual (RI) to make a decision regarding the next step in the process.
- **Process Ignorance**. Individuals who intervene in a given process should generally not need to know what that process is – only what they need to do when it is their turn to intervene in the workflow.
- **Next State Coherence**. Choices to be made by a RI relative to a given dossier should be presented in a consistent way across a given workflow and across applications. An example of this would be to always present the following choices in the same order: *To be modified*, *Rejected*, *Approved*.
- **Data Modification**. One and only one individual should be responsible for modifying any given piece of data in the workflow document (to ensure data integrity). For those cases where modifications need to be made, the workflow document should be routed back to the responsible individual (such as the document author).
- **Comments**. Each individual who intervenes in the workflow should have a field reserved for his or her usage to add any and all comments related to that dossier. To note this is particularly essential in the context of a request for modification (as mentioned prior).
- **Email Notifications**. When a document is advanced in the workflow, those who are next in line to intervene should be notified by email with a document link back to the original workflow document. It is also often the case that all those who have intervened in the document are notified when the document is eventually archived (or rejected).

1.5) About This Guide

This guide is designed to be used in conjunction with videos and applications (*WAApplication01.nsf* and *WAApplication02.nsf*) provided on the Horizon Ascendant website. The videos demonstrate what the various mechanisms look like in operation to the end-user while the applications provided hands-on experience in using them. The guide illustrates (in most cases graphically) the components which make up those mechanisms and (hopefully) serves as an ongoing reference in developing workflow applications.

The objective of this guide is not to demonstrate *how* to put those mechanisms in place. As most of the "development" involves configuring documents, a large majority of these manipulations should be straightforward. In cases where they are not, however, videos have been provided to assist in this and others will continue to be added as time goes on and the need for such becomes apparent.

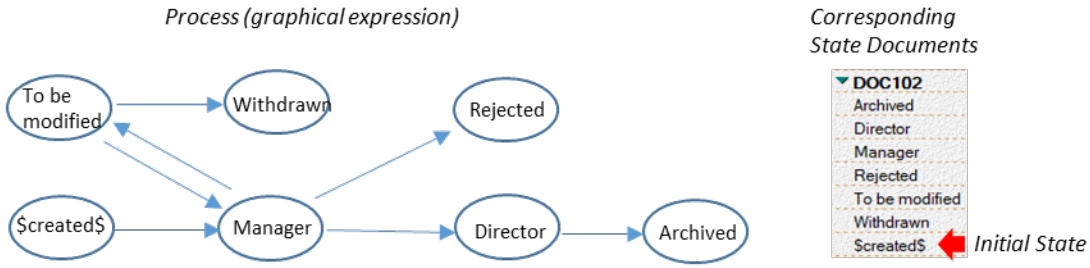
This guide was created with you the developer in mind.

- **One concept, one page**. Everything regarding a given topic is put right in front of you.
- **Consistent presentation:**
  - *Purpose* – a concise description of what the section is about
  - *What it looks like* – typically what the end-user sees in operation
  - *How to make it happen* – an illustration of the components behind the displayed functionality
  - *Additional notes and references* – where to go for additional information
- **Graphical presentation**. Visually see how components are connected to each other.
- **Hyperlinks**. Select **Ctrl+Click** on entries in the index or any blue highlighted text to navigate through various subjects in the document. Select **Alt+<left arrow>** to return to where you were previously.

Chapter 2 – Process: Routing Documents

2.1) General

The process or workflow for a given type of document is defined in State documents. Contained in these documents are all rules / process related definitions.



State documents are categorized first by version, then by form (process) and finally by state. The current workflow version for a given process is stored in the active Language document. When that version value is changed, newly created workflow documents will follow the new State documents while those already in the process will follow the previous ones. Also stored in the Language document are the actual Notes form names which corresponds to the Form Names (an alias) that the user sees.

STATE			
Version	**Form Name**	**State Name**	**State Type**
V01	DOC102	Manager	(•) Standard ( ) Virtual

Workflow | Notifications | Actions | Documentation

GENERAL CONSTRAINTS

	The following RIs can intervene...	And forward to the following states...	Which will be displayed as...	Parallel Operation	Extended Operation	Inclusive Operation	View Display	Historical
1	Manager	To be modified	<state>	-	-	-	<user>	<state>
2	Manager	Rejected	<state>	-	-	-	<user>	<state>
3	Manager	Director	Approved	-	-	-	<user>	<state>

Workflow States are connected by Current State - Next State relationships. Detailed in each **State** document, these relationships define the paths of the workflow. In the example above, if the user selects the option *Approved* (followed by the *Send* button) in state *Manager*, the document will be routed to state *Director*.

The following sections demonstrate some of the many ways to route documents. What is not addressed is how specific users are designated as being those responsible to intervene at a given state. This is discussed in detail in Chapter 3 - Constraints: Specifying Users. Note that the applications referenced here are configured with sample users (available in the default configuration for download):

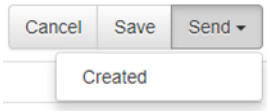
- *a1/Horizon*
- *blue/Horizon*
- Etc.

2.2) Simple Document Routing

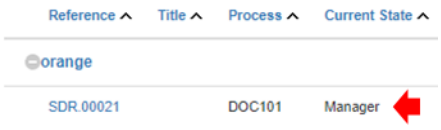
PURPOSE

Simple Document Routing is when there is only one next state defined from the current state. This is typically the case when a user submits an initial request to be approved by a manager. In that scenario, the user selects the *Send* button and the document advances to that state. In the Notes client, no choices are presented to the end user and the document advances directly.

WHAT IT LOOKS LIKE

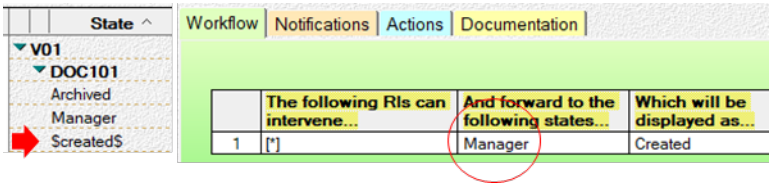


Only one selection is available here for the end



The result after selecting *Send* from the initial state *$created$* in the process

HOW TO MAKE IT HAPPEN



State document *V01* (version), *DOC101* (process), *$created$* (state)

[*] in the initial state indicates that any user with access to the database can create a document of this type.
$created$ is a reserved word and always refers to the initial state in a process.

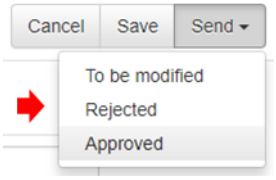
ADDITIONAL NOTES / REFERENCES

2.3) Manual Document Routing

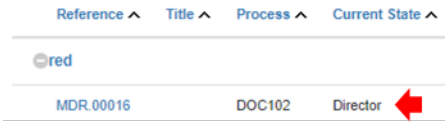
PURPOSE

Manual Document Routing indicates that the user is a given a choice as to where to advance the document. The user selects the desired option from the drop down list followed by the *Send* button. The workflow document is then forwarded accordingly. In the Notes client, the user first selects the button *Send*, upon which a dialog box is presented from which to make the selection.

WHAT IT LOOKS LIKE

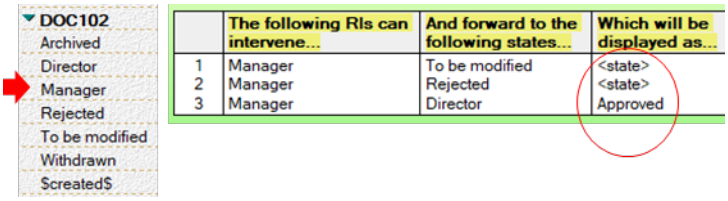


Multiple selections are made available to the user in state *Manager*



The resulting state of the workflow document is *Director* after the user selects *Approved* followed by

HOW TO MAKE IT HAPPEN



State document *V01, DOC101, Manager*. Note the alias *Approved* which is presented to the user in place of

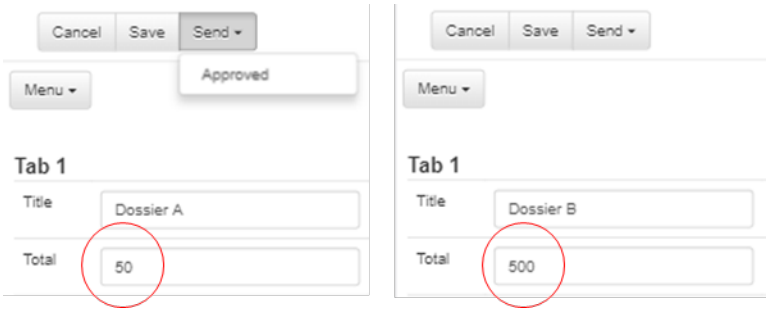
ADDITIONAL NOTES / REFERENCES

2.4) Automatic Document Routing

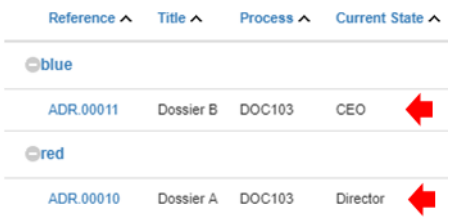
PURPOSE

Automatic Routing **advances a document in the workflow according to a set of predefined rules**. What we have seen up to this point in time are Manual Rules which allow the user to select the next state. Upon selecting the *Send* button, Workflow Ascendant automatically determines the next document state and advances it based on the conditions defined in the State document evaluated against values stored in the workflow document.

WHAT IT LOOKS LIKE

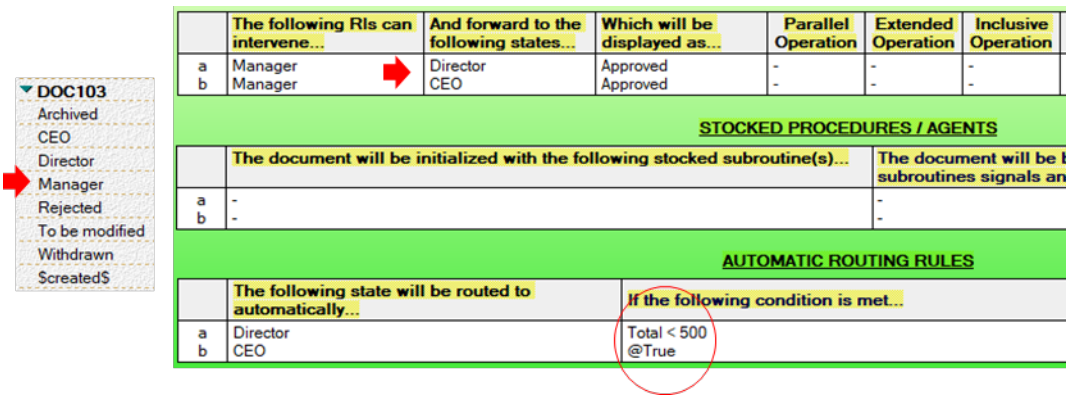


Two documents are created with different total amounts (as seen from an iPhone X)



From state *Manager*, the user selects *Send* for both documents which are in turn routed to two different states

HOW TO MAKE IT HAPPEN



The last frame to the right directly above forms an *If-Then, If-Then, …Else* statement (exactly like the *@If* command) and are evaluated from top to bottom. The last entry in this list must always be *@True* (the default condition). Automatic rules must always have the same display name per responsible individual (in this case *Approved*).

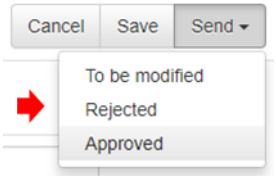
ADDITIONAL NOTES / REFERENCES

2.5) Manual and Automatic Document Routing

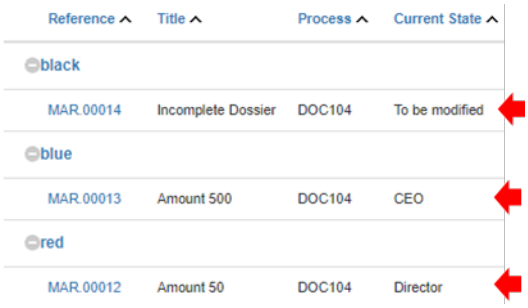
PURPOSE

Manual and automatic rules can be combined in the same state. A classic example of this is a manager who may send a request back to the author, reject the request or approve it. In the latter case however, the document may take a different path depending on the amount of the request. If the amount is less than or equal to, say, *500*, the document will be sent to a *Director*. If, however, the amount exceeds *500*, it will be sent to the *CEO*. In the corresponding State document, the first two rules will be manual, the last two rules automatic.

WHAT IT LOOKS LIKE



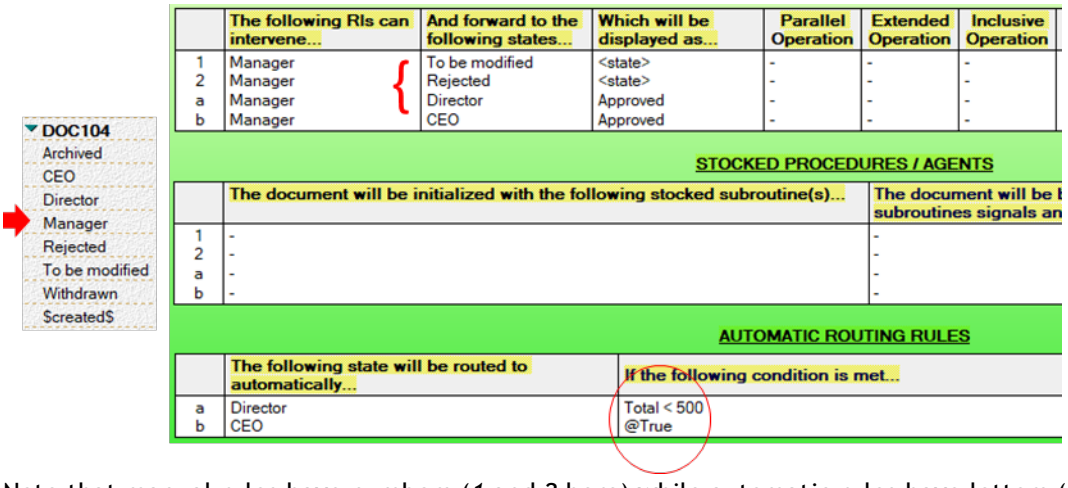
Four documents are created with different total amounts



Reference	Title	Process	Current State
black			
MAR.00014	Incomplete Dossier	DOC104	To be modified
blue			
MAR.00013	Amount 500	DOC104	CEO
red			
MAR.00012	Amount 50	DOC104	Director

From state *Manager*, one document is sent back to the author, another is rejected (displayed in the view *Archived*) and the remaining two (*50* and *500*) are

HOW TO MAKE IT HAPPEN



DOC104
- Archived
- CEO
- Director
- Manager
- Rejected
- To be modified
- Withdrawn
- ScreatedS

	The following RIs can intervene...	And forward to the following states...	Which will be displayed as...	Parallel Operation	Extended Operation	Inclusive Operation
1	Manager	To be modified	<state>	-	-	-
2	Manager	Rejected	<state>	-	-	-
a	Manager	Director	Approved	-	-	-
b	Manager	CEO	Approved	-	-	-

STOCKED PROCEDURES / AGENTS

	The document will be initialized with the following stocked subroutine(s)...	The document will be subroutines signals an
1	-	-
2	-	-
a	-	-
b	-	-

AUTOMATIC ROUTING RULES

	The following state will be routed to automatically...	If the following condition is met...
a	Director	Total < 500
b	CEO	@True

Note that manual rules have numbers (*1* and *2* here) while automatic rules have letters (*a* and *b* here).

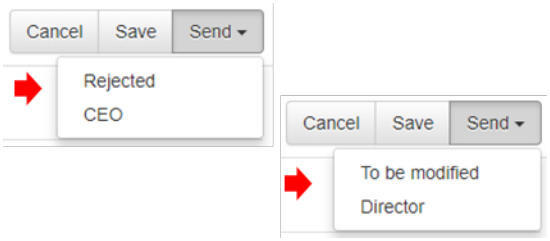
ADDITIONAL NOTES / REFERENCES

2.6) Selective Operations

PURPOSE

Different constraints (discussed in Chapter 3) can be combined within the same state. When this occurs, users only see the next states associated with their constraints. This is often referred to as ***Next State Visibility***. The easiest way to understand this is in the context of an example.

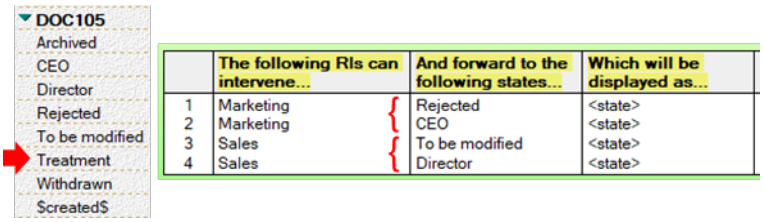
WHAT IT LOOKS LIKE



These are the choices that are presented to *Marketing* in state *Treatment*

These are the choices that are presented to *Sales* in state *Treatment*

HOW TO MAKE IT HAPPEN



	The following RIs can intervene...	And forward to the following states...	Which will be displayed as...
1	Marketing	Rejected	<state>
2	Marketing	CEO	<state>
3	Sales	To be modified	<state>
4	Sales	Director	<state>

Note that if you intervene as the database manager, you will see all four choices as by default you assume all roles!

Note that in this example, any user from *Marketing* or *Sales* is sufficient to advance the document to a different state.

ADDITIONAL NOTES / REFERENCES

2.7) Parallel Operations

PURPOSE

There are times in a process where multiple individuals must intervene in a given state before the document can be advanced. In the previous examples only one authorized user is sufficient to advance those documents in the workflow. A process that requires that an individual from multiple different groups (roles or fields) must intervene at a given point in time is referred to as a Parallel Operation. A process that requires that *all* individuals from a particular group (role or field) must intervene at a point in time is referred to as an Inclusive Operation (described in the following section). These operations can, of course, be combined in the same state.

WHAT IT LOOKS LIKE

Reference	Title	Process	Current State
e1			
POP.00006	Test Parallel	DOC106	Treatment
e2			
POP.00006	Test Parallel	DOC106	Treatment
green			
POP.00006	Test Parallel	DOC106	Treatment
yellow			
POP.00006	Test Parallel	DOC106	

Users *e1* or *e2* (*Production*), *green* (*Sales*) and *yellow* (*Marketing*) are all solicited to intervene in doc POP.00006

Reference	Title	Process	Current State
green			
POP.00006	Test Parallel	DOC106	Treatment
yellow			
POP.00006	Test Parallel	DOC106	Treatment

The result after user *e1* intervenes in the document. Users *green* and *yellow* must now both intervene in the document for it to advance in the workflow

HOW TO MAKE IT HAPPEN

DOC106
Archived
Director
Treatment
Screated$

	The following RIs can intervene...	And forward to the following states...	Which will be displayed as...	Parallel Operation	Extended Operation	Inclusive Operation
1	Marketing	Director	Treated	Yes	-	-
2	Sales	Director	Treated	Yes	-	-
3	Production	Director	Treated	Yes	-	-

Note that in a parallel operation, users can intervene in any order.

ADDITIONAL NOTES / REFERENCES

2.8) Inclusive Operations

Purpose

There are times in a process where multiple individuals must intervene in a given state before the document can be advanced. The previous section Parallel Operations treated the case where an individual from multiple different groups must intervene at a point in time. This section treats the case where all individuals from a particular group must intervene at a given point in time – referred to as an *Inclusive Operation*. These operations can, of course, be combined within the same state.

What it looks like

Reference	Title	Process	Current State
e1			
IOP.00004	Test Inclusive	DOC107	Treatment
e2			
IOP.00004	Test Inclusive	DOC107	Treatment

All the users in *Production* (users *e1* and *e2*) are solicited to intervene in doc

Reference	Title	Process	Current State
red			
IOP.00004	Test Inclusive	DOC107	Director

The resulting state after both *e1* and *e2* intervene

How to make it happen

	The following RIs can intervene...	And forward to the following states...	Which will be displayed as...	Parallel Operation	Extended Operation	Inclusive Operation
1	Production	Director	Treated	-	-	Yes

As with parallel operations, users can intervene in any order in an inclusive operation. Note that if you intervene as the database manager, the document will advance regardless of who else has intervened prior!

Additional Notes / References

2.9) Parallel and Inclusive Operations

PURPOSE

Parallel Operations and Inclusive Operations can be defined in the same state and even in the same *rule*. For a description of each, reference the previous two sections.

WHAT IT LOOKS LIKE

Reference	Title	Process	Current State
e1			
PIO.00004	Parallel & Inclusive Op	DOC108	Treatment
e2			
PIO.00004	Parallel & Inclusive Op	DOC108	Treatment
green			
PIO.00004	Parallel & Inclusive Op	DOC108	Treatment
yellow			
PIO.00004	Parallel & Inclusive Op	DOC108	Treatment

Users *e1* and *e2* (*Production*), *green* (*Sales*) and *yellow* (*Marketing*) are all solicited to intervene in doc POP.00002

e2			
PIO.00004	Parallel & Inclusive Op	DOC108	Treatment
green			
PIO.00004	Parallel & Inclusive Op	DOC108	Treatment
yellow			
PIO.00004	Parallel & Inclusive Op	DOC108	Treatment

The result after user *e1* intervenes in the document (compare this to the example in Parallel Operations). Users e2, *green* and *yellow* must now all intervene in the document for it to advance in the process

HOW TO MAKE IT HAPPEN

DOC108
Archived
Director
Treatment
Screated$

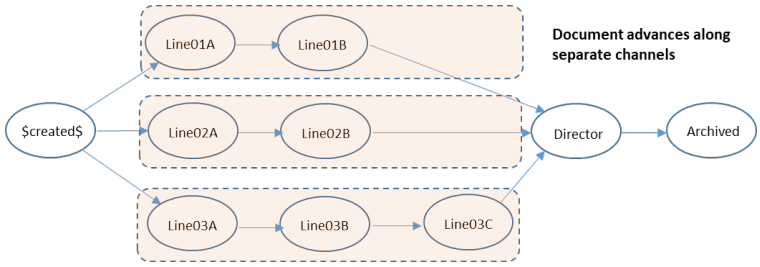
	The following RIs can intervene...	And forward to the following states...	Which will be displayed as...	Parallel Operation	Extended Operation	Inclusive Operation
1	Marketing	Director	Treated	Yes	-	-
2	Sales	Director	Treated	Yes	-	-
3	Production	Director	Treated	Yes	-	Yes

In the above example, all users in *Production* and any one user from *Marketing* and any one user from *Sales* must intervene (in any order) at state *Treatment* for the document to advance..

ADDITIONAL NOTES / REFERENCES

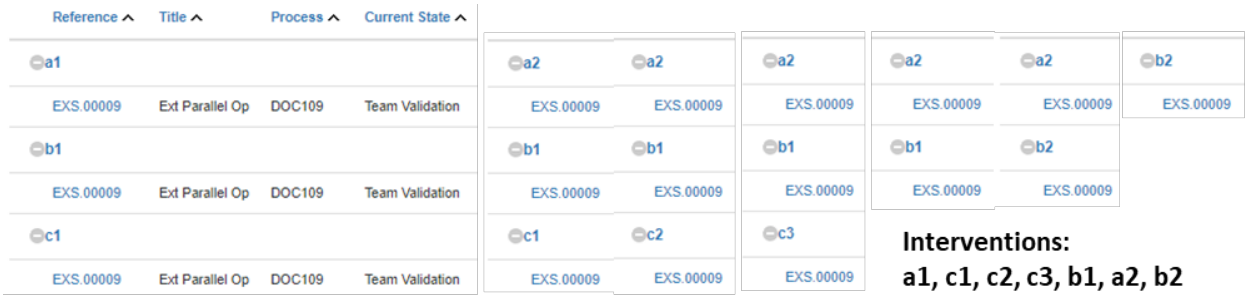
2.10) Extended Parallel Operations

PURPOSE

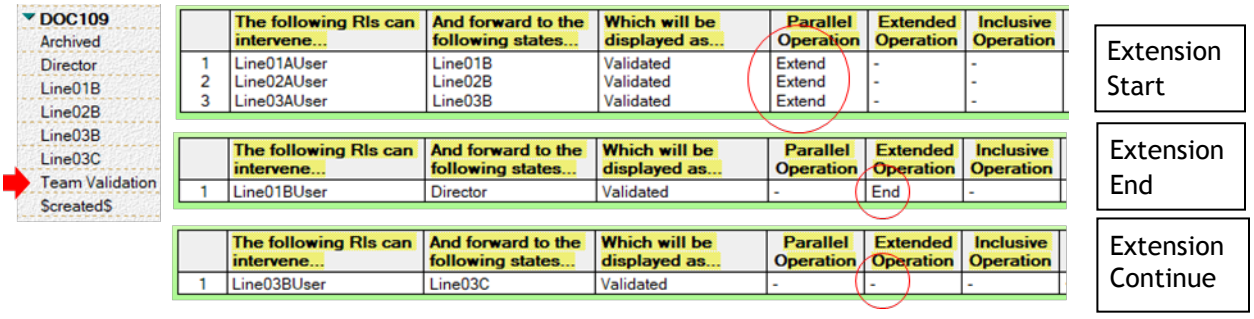


Extended States "extend" the scope of a parallel operation. In a standard parallel operation, an actor from each of the different constraints (i.e., roles) must intervene for the document to advance. An extended parallel state, however, allows a document to advance along *separate channels* in parallel. The easiest way to understand this concept is in terms of the above graphic. The workflow document can advance from *Line02A* to *Line02B* without, for example, waiting for *Line01A* to advance to *Line01B* and *Line03A* to advance to *Line03B* (or any combination therein). The document only advances to *Director*, however, after the designated user(s) have all intervened in *Line01B*, *Line02B* and *Line03C*.

WHAT IT LOOKS LIKE



HOW TO MAKE IT HAPPEN



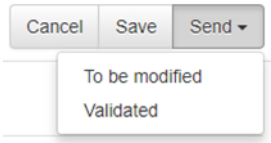
ADDITIONAL NOTES / REFERENCES

2.11) Extended Sequential Operations

PURPOSE

Extended Sequential Operations refer to the ability to "fuse" two states together. Prior examples have all looked at how one person advances a document in the process from one state to the next. Even in the case where multiple users intervene in parallel, it is always the last person who selects the *Send* button who determines the next state. What if, however, the next state depends on the *collective* opinion of a group of users? In this case we need to make use of what we refer to as a *Virtual State*. When a document advances to a *Virtual State*, the rules of that state are executed immediately – effectively chaining two levels of rules together, one after the other.

WHAT IT LOOKS LIKE



In process DOC110, *Marketing*, *Sales* and *Production* are all solicited to intervene in documents at state *Treatment*, each being presented with the above choices. The business logic is that at least two of the three entities must validate the document for it to advance to the *Director* (otherwise it is sent back to the author for modification).

Reference	Title	Process	Current State
black			
ESO.00004	Ext Sequential Op 1	DOC110	To be modified
red			
ESO.00005	Ext Sequential Op 2	DOC110	Director

User *e2* (*Production*) said *no* to both documents, *green* (*Sales*) said *no* to the first, *yes* to the second and lastly yellow (*Marketing*) said *yes* to both documents

HOW TO MAKE IT HAPPEN

DOC110
Archived
Director
To be modified
Treatment
Virtual01
Withdrawn
Screated$

	The following RIs can intervene...	And forward to the following states...	Which will be displayed as...	Parallel Operation
1	Marketing	Virtual01	To be modified	Yes
2	Marketing	Virtual01	Validated	Yes
3	Sales	Virtual01	To be modified	Yes
4	Sales	Virtual01	Validated	Yes
5	Production	Virtual01	To be modified	Yes
6	Production	Virtual01	Validated	Yes

Actions

#	Field Names	
1		
2	Count	Count + 1
3		
4	Count	Count + 1
5		
6	Count	Count + 1

State: Virtual01

State Type: ○ Standard ◉ Virtual

	The following RIs can intervene...	And forward to the following states...	Which will be displayed as...
a	[*]	To be modified	Next
b	[*]	Director	Next

	The following state will be routed to automatically...	If the following
a	To be modified	Count < 2
b	Director	@True

Each time *Validated* is selected in state *Treatment*, field *Count* is incremented by *1* in the document. When the document advances to *Virtual01*, the automatic rules are executed immediately and route the doc based on *Count*.

ADDITIONAL NOTES / REFERENCES

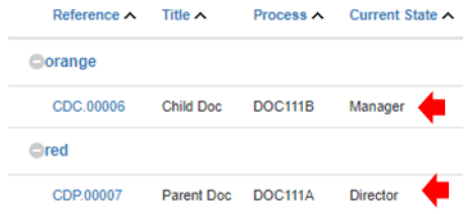
Action definitions are contained on the third tab of a State document and explained in Section 2.14.

2.12) Child Documents

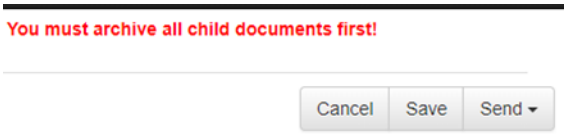
PURPOSE

Child documents are documents which are created from a parent document. Workflow Ascendant can manage a hierarchy of these documents. Examples might include: a Purchase Request, which can lead to (or create) one or more Purchase Orders, which can lead to (or create) one or more Delivery Tickets. At document creation, each of these child documents subsequently follows its own process / set of State documents.

WHAT IT LOOKS LIKE

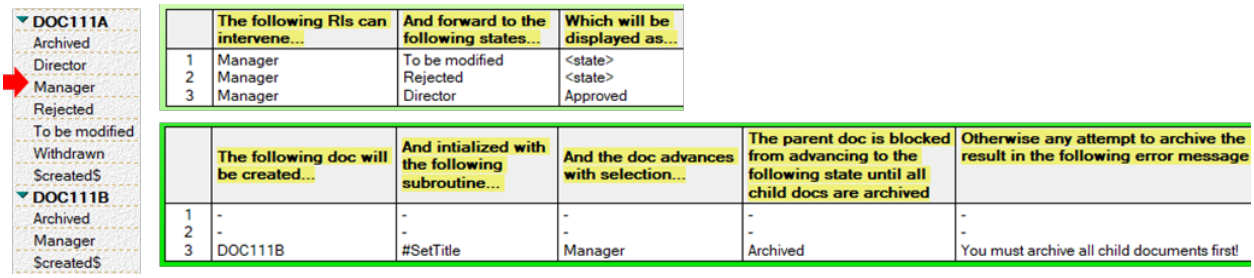


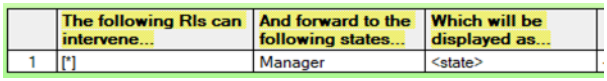
When *Approved* is selected at state *Manager*, child document *DOC111B* is automatically created



An error message is displayed at any attempt to archive the parent document before the child document(s)

HOW TO MAKE IT HAPPEN





DOC111B. This document is created (and forwarded) with the rights/profile of *Manager*.

ADDITIONAL NOTES / REFERENCES

The use of invoked routines (*#SetTitle* in the example above) is explained in Section 4.2.

Note that you will normally want to hide the direct creation of a child document, an option which you can select in the Language document.

2.13) Email Notifications

PURPOSE

Email notifications are typically sent when a document is advanced in the workflow. These messages can be personalized for each rule or "path" in State documents which include the: *To*, *Cc*, *Bcc*, *Subject* and *Body* fields. Default values are used from the active Language document for any fields left blank. A link back to the pertinent workflow document is automatically included at the end of each email notification.

WHAT IT LOOKS LIKE

DOC201. Dossier archived: "FDC.00039"
red
To: black
Cc: red, orange, black

For information only. The dossier FDC.00039 (DOC201) has been archived.

For additional details please select the following link:
[--- click here ---]

Notification sent from process *DOC201* state *Director* when *Archive* and the *Send* button were selected. Emails can be sent in Notes or MIME format (see Language document).

HOW TO MAKE IT HAPPEN

	The following RIs can intervene...	And forward to the following states...	Which will be displayed as...
1	Director	To be modified	<state>
2	Director	Rejected	<state>
3	Director	Archived	Archive

WAHistoryAuthors is a field containing all users who have intervened in the workflow document to date (Cc: *orange* intervened at state *Manager*)

#	To	cc	bcc	Subject*	Body*
1	*N	WAHistoryAuthors			
2	*A	WAHistoryAuthors		WAFormName + ". Dossier declined: " + @Char(34) + WADocRef + @Char(34)	"" "For information only. The dossier " + WADocRef + " (" + WAFormName + ") has been archived without further action." "" "For additional details, please select the following link:"
3	*A	WAHistoryAuthors		WAFormName + ". Dossier archived: " + @Char(34) + WADocRef + @Char(34)	"" "For information only. The dossier " + WADocRef + " (" + WAFormName + ") has been archived." "" "For additional details, please select the following link:"

Workflow Ascendant interprets the *Subject* and *Body* fields which contain executable IBM Notes @Formula language code. The latter of these is of multi-line format where each line represents an executable statement. In the address fields (*To*, *cc*, *bcc*), you can specify multiple values (one per line) mixing any of the following formats:

- ***N** – indicates the next individuals slated to intervene in the document
- ***A** – indicates the document author
- Specific user names or group names (from the Notes Directory)
- Field names (which in turn must contain valid user names or group names)

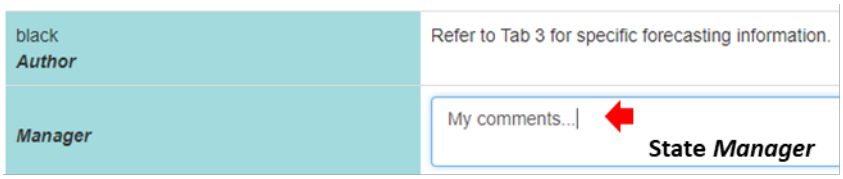
ADDITIONAL NOTES / REFERENCES

2.14) Actions (Comments)

PURPOSE

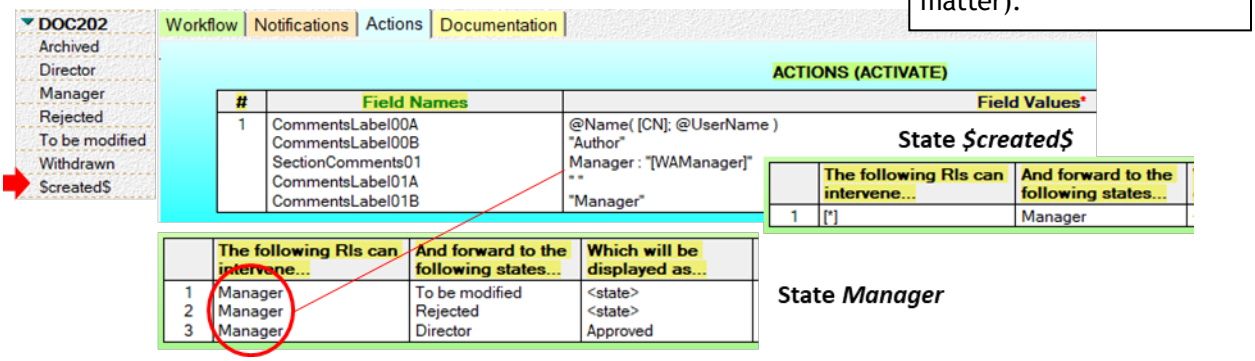
The *Actions* tab in the State document contains definitions to set fields in the workflow document. When the *Send* button in the workflow document is selected, as part of the executed designated rule, fields are updated according to those definitions. The specific example contained herein addresses how these definitions can be used to dedicate specific (comment) fields in the workflow document for each contributor in the process.

WHAT IT LOOKS LIKE



Manager orange can modify the comments field reserved for that role but not the comments field reserved for the document author (or anyone else's for that matter).

HOW TO MAKE IT HAPPEN



Field names on the left are updated with executable @Formula statements which are interpreted on the right (one per line). In this specific example where the handling of comments is the object, the reserved field names are as follows (where # corresponds to a row in the *Comments* table):

- *CommentsLabel#A* – The user name to be displayed
- *CommentsLabel#B* – The user role to be displayed
- *SectionComments#* – The users (Constraint) that are authorized to modify the comments field for that row

ADDITIONAL NOTES / REFERENCES

Reference Appendix D for the most commonly used @Formulas used in State documents.

2.15) Historical Text

PURPOSE

The *History* tab of each workflow document details each intervention along with the number of days the document has remained with that user. The text description by default is set to the name of the next state but can be customized. Each time an individual selects the *Send* button, Workflow Ascendant adds a line to a table in that document in the tab entitled *Historical*:

- The user name
- The date the user advanced the document in the workflow
- A text description of what his or her action represents

WHAT IT LOOKS LIKE

Contributor	Date	Description	Days
black	04-03-2018	Created (Author)	0
orange	05-03-2018	Approved (Manager)	1
<Current State>			2
Total			3

DOC202 at state *Director* subsequent to the document being created by *black* and approved by *Manager orange*.

HOW TO MAKE IT HAPPEN

GENERAL CONSTRAINTS

	The following RIs can intervene...	And forward to the following states...	Which will be displayed as...	Parallel Operation	Extended Operation	Inclusive Operation	View Display	Historical Description
1	Manager	To be modified	<state>	-	-	-	<user>	To be modified (Manager)
2	Manager	Rejected	<state>	-	-	-	<user>	Rejected (Manager)
3	Manager	Director	Approved	-	-	-	<user>	Approved (Manager)

Process *DOC202* state *Director* with personalized historical descriptions.

ADDITIONAL NOTES / REFERENCES

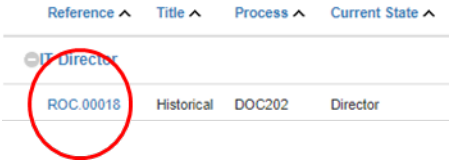
Note that you will typically not want to add an historical entry in the case of a Virtual State, which you specify as part of the State document definitions.

2.16) Document References

PURPOSE

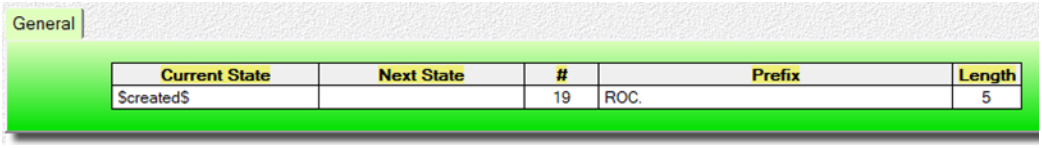
Workflow Ascendant provides for the automatic allocation of unique and sequential references to workflow docs. Virtually all processes require this in order to ensure the unique identity of a document and that no document is lost. With respect to the latter, you should always archive your workflow documents as opposed to deleting them. Reference documents are defined by version and process version in view *Workflow - References*.

WHAT IT LOOKS LIKE



Reference *ROC.00018* is assigned to a workflow doc belonging to process *DOC202*

HOW TO MAKE IT HAPPEN



Reference document *DOC202*. In this example, the next workflow document forwarded from state *$created$* to any other state will be assigned the prefix *ROC* followed by a text number of length *5* (filled with leading 0s) with the next number in the sequence (*19*) to be allocated: *ROC.00019*.

ADDITIONAL NOTES / REFERENCES

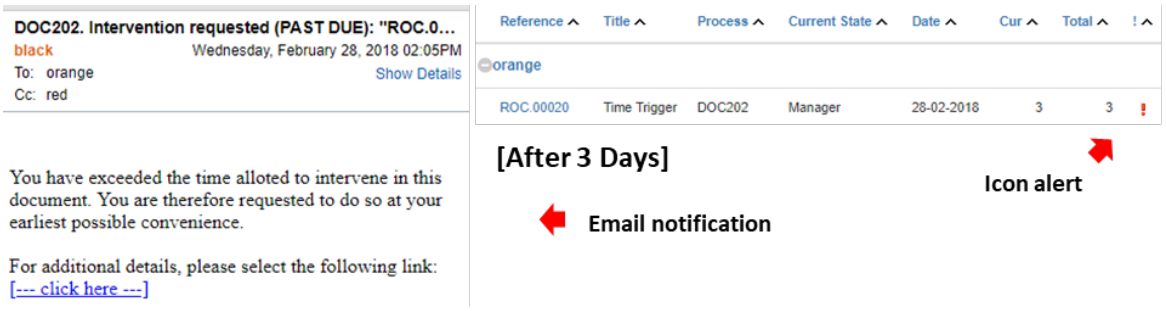
If you deploy your application on multiple servers, you will need to specify which server will allocate this unique reference in the Language document.

2.17) Alert Timeouts

PURPOSE

Emails can be sent and alerts displayed if users do not intervene in a designated amount of time. These are defined in Time Trigger documents in the *Workflow - Time Triggers* view. In this example (*DOC202*), an email is sent to *Manager* (cc to *Director*) and an alert displayed when the document remains in state *Manager* for *3* days.

WHAT IT LOOKS LIKE



HOW TO MAKE IT HAPPEN

General | Notifications | Actions | Documentation

CONTEXT

Start State	End State	State Change	Condition
Manager	-	-	@True

DETAILS

Trigger Type	Trigger Value	Agent to launch	History (User)	History (Description)
Days	3	-	n/a	n/a

EMAIL NOTIFICATIONS

To	cc	bcc	Subject	Body
WACurrentAuthors	Director		WAFormName + ". Intervention requested (PAST DUE): " + @Char(34) + WADocRef + @Char(34)	"" "You have exceeded the time alloted to intervene in this document. You are therefore requested to do so at your earliest possible convenience." "" "For additional details, please select the following link:"

ACTIONS (ACTIVATE)

Field Names	Field Values
WADelayIcon	150
WAChartCount01	WAChartCount01 : WACurrentAuthors

ACTIONS (DEACTIVATE)

Field Names	Field Values
WADelayIcon	0

Field *WADelayIcon* is set to *150* (*!* In the view) when the trigger is activated, to *0* upon a

ADDITIONAL NOTES / REFERENCES

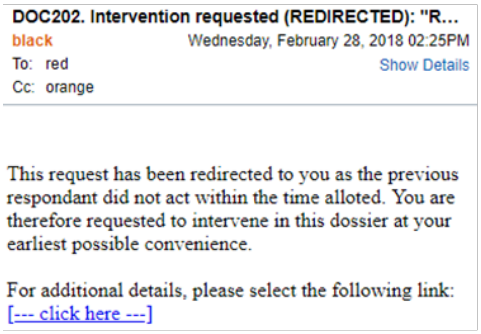
For testing purposes, you can simulate the passage of time using the controls displayed in the Admin view of the General Web Interface.

2.18) Time Based Routing

PURPOSE

Workflow documents can be automatically routed based on elapsed time. These are defined in Time Trigger documents in the *Workflow - Time Triggers* view. In this example (*DOC202*), the document is routed to *Director* when *Manager* does not intervene with the allotted *5* day window. Multiple Time Trigger documents can be assigned to the same state.

WHAT IT LOOKS LIKE



[After 3 Days]

[After 5 Days]

Contributor	Date	Description	Days
black	28-02-2018	Created (Author)	0
Forced Document Routing	05-03-2018	Final delay exceeded	5
<Current State>			0
Total			5

HOW TO MAKE IT HAPPEN

General | Notifications | Actions | Documentation

CONTEXT

Start State	End State	State Change	Condition
Manager	-	Director	@True

DETAILS

Trigger Type	Trigger Value	Agent to launch	History (User)	History (Description)
Days	5	-	Forced Document Routing	Final delay exceeded

EMAIL NOTIFICATIONS

To	cc	bcc	Subject	Body
Director	WACurrentAuthors		WAFormName + ". Intervention requested (REDIRECTED): " + @Char(34) + WADocRef + @Char(34)	"" "This request has been redirected to you as the previous respondant did not act within the time alloted. You are therefore requested to intervene in this dossier at your earliest possible convenience." "" "For additional details, please select the following link:"

ACTIONS (ACTIVATE)

Field Names	Field Values
WADelaylcon	0
SectionComments02	Director : "[WAManager]"
CommentsLabel02A	"IT Director"

ACTIONS (DEACTIVATE)

Field Names	Field Values

Reference the Email Notifications, Actions and Alert Timeouts sections for additional explanations

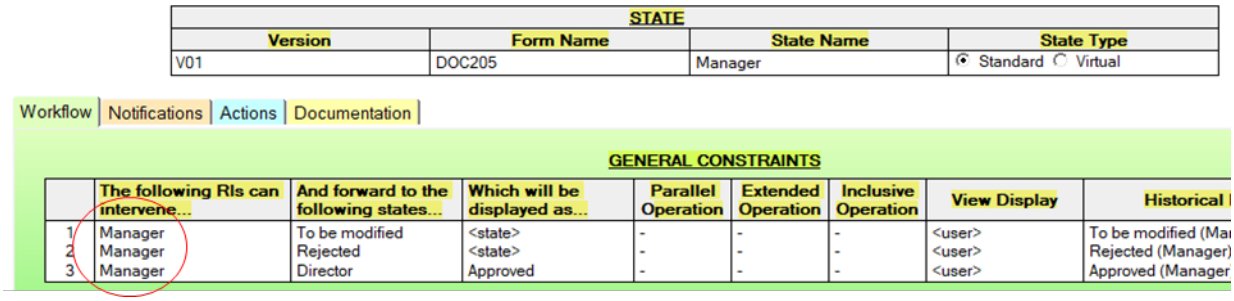
ADDITIONAL NOTES / REFERENCES

For testing purposes, you can simulate the passage of time using the controls displayed in the Admin view of the General Web Interface.

Chapter 3 – Constraints: Specifying Users

3.1) General

This chapter addresses the notion of constraints which refer to the profiles of those who can intervene in a workflow document at a particular state. This is in contrast to *Chapter 2* which dealt exclusively with the routing of workflow documents between states. Constraints are defined in terms of roles or fields, each of which must in turn resolve to a set of user names. Only those users defined in *The following RIs can intervene* (see the graphic below) are allowed to modify the document for that particular document state. So in the following example, when a workflow document belonging to process *DOC205* is in state *Manager*, only the users whose names are contained in the corresponding workflow document field *Manager* can intervene (i.e., affect any modifications to that document).



STATE

Version	Form Name	State Name	State Type
V01	DOC205	Manager	(•) Standard ( ) Virtual

Workflow | Notifications | Actions | Documentation

GENERAL CONSTRAINTS

	The following RIs can intervene...	And forward to the following states...	Which will be displayed as...	Parallel Operation	Extended Operation	Inclusive Operation	View Display	Historical
1	Manager	To be modified	<state>	-	-	-	<user>	To be modified (Ma
2	Manager	Rejected	<state>	-	-	-	<user>	Rejected (Manager)
3	Manager	Director	Approved	-	-	-	<user>	Approved (Manager

Note that it is strictly a coincidence that the state name and the constraint names are identical. Different constraints can also be defined in the same document (see Selective Operations).

In the workflow document, field *WACurrentAuthors* contains the list of users authorized to modify the document at that moment in time (field *WACurrentAuthorsDisplay* is what is displayed in the view). If a given user cannot edit a workflow document at a given state, it is because he or she is not listed in *WACurrentAuthors*.

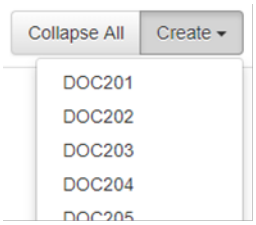
If a workflow document is unexpectedly archived, this normally indicates that there were no valid user names in the state being routed to (an error condition).

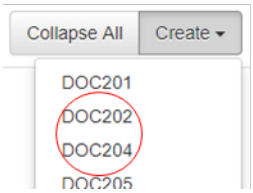
3.2) Roles

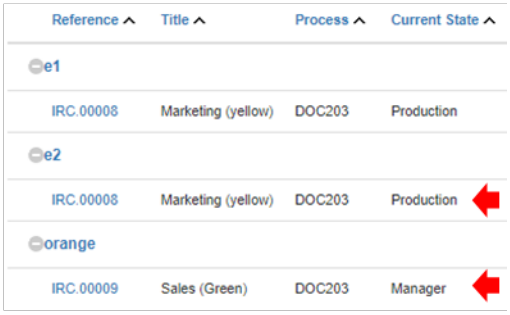
PURPOSE

Role constraints refer specifically to IBM Domino roles and Role documents. The latter are contained in view *Workflow - Roles* from which the ACL can be managed directly via the button *Update ACL*. Role constraints must only be used in the initial State document *$created$* and only role constraints should be used therein. If role *[*]* is used, then any user can create a document for that process. If specific role names are specified, then only users belonging to those roles (with the exception of *[WAManager]*) are allowed to create workflow documents of that type. In example *DOC203* below, *yellow* from *Marketing* and *green* from *Sales* have both created a document followed by selecting the *Send* button.

WHAT IT LOOKS LIKE







The above graphic to the left displays the available document creation choices for the *Marketing* and *Sales* people. In the graphic just underneath this, however, note the absence of *DOC203* from the list of choices presented to user *b1* (who doesn't belong to either of these two groups).

HOW TO MAKE IT HAPPEN

	The following RIs can intervene...	And forward to the following states...	Which will be displayed as...
1	[Marketing]	Production	Created
2	[Sales]	Manager	Created

Only role constraint types should ever be used in initial state *$created$*

ADDITIONAL NOTES / REFERENCES

3.3) Fields

PURPOSE

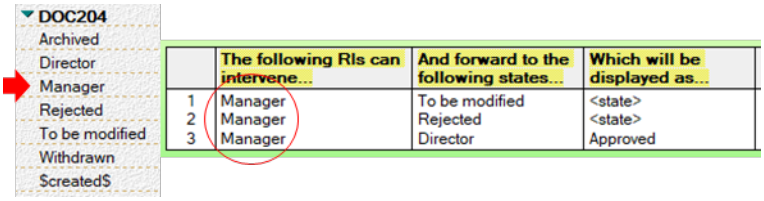
Field constraints refer to field names contained in the corresponding workflow documents and Field documents. The latter are contained in view *Workflow - Fields*. Field constraints should be used in every State document with the exception of initial state *$created$*. The corresponding fields in the workflow documents must contain one or more valid user names. In example *DOC204* below, users *d1* and *d2* are designated as the responsible individuals (field constraint *Manager*) in state *Manager*.

WHAT IT LOOKS LIKE

Reference	Title	Process	Current State
d1			
FCN.00003	Fields	DOC204	Manager
d2			
FCN.00003	Fields	DOC204	Manager

Users *d1* and *d2* are designated to intervene at

HOW TO MAKE IT HAPPEN



DOC204
Archived
Director
Manager
Rejected
To be modified
Withdrawn
$created$

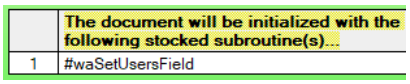
	The following RIs can intervene...	And forward to the following states...	Which will be displayed as...
1	Manager	To be modified	<state>
2	Manager	Rejected	<state>
3	Manager	Director	Approved

DOC204 State document *Manager* dictates that only those users contained in the workflow document field *Manager* can modify the document at that state

DOC204

Director	Director	e1
Manager	Manager	d1,d2
WAAuthor	Author	

Users *d1* and *d2* are assigned to field *Manager* in Field document *Manager*



	The document will be initialized with the following stocked subroutine(s)...
1	#waSetUsersField

Routine *#waSetUsersField* in State document *$created$* sets the user names in the workflow document from the Field documents on document creation (see section Document

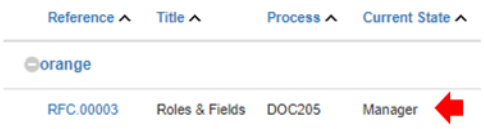
ADDITIONAL NOTES / REFERENCES

3.3) Roles and Fields

PURPOSE

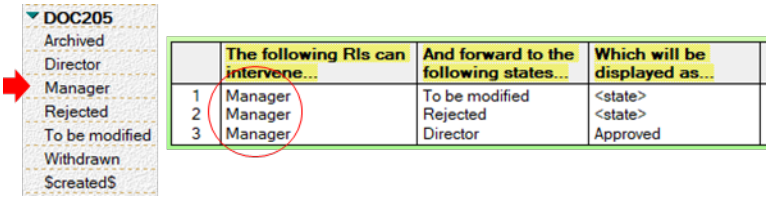
Role and Field constraints can be used in harmony. With the exception of the initial state, all constraints in the workflow must resolve to fields containing user names. Since Field constraints are tied to the process version (and thus would require the recopying of user names between these documents), Workflow Ascendant provides a system to use roles which are independent of that. In example *DOC205* below, user *orange* is designated as the responsible individual (field constraint *Manager* which derives from role *Manager*) in state *Manager* (note that while the state name happens to be the same as that of the constraint in this example, they are in reality completely independent).

WHAT IT LOOKS LIKE

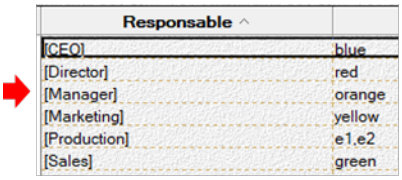


User *orange* is designated to intervene at state *Manager*

HOW TO MAKE IT HAPPEN

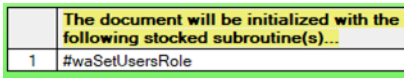


The Field documents must not contain any user names as these names would take precedence over any names contained in the Role docs



Field Documents

Role Documents



Routine *#waSetUsersRole* in State document *$created$* sets the user names in the workflow document from the Role documents on document creation (see section Document Initialization) provided there is a corresponding Field

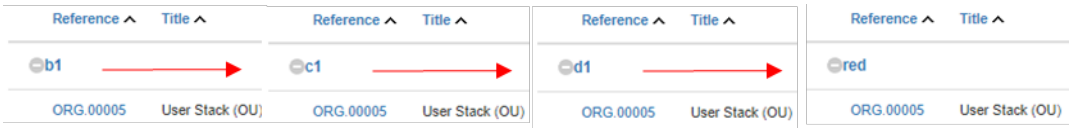
ADDITIONAL NOTES / REFERENCES

3.4) User Stack (Organizational Hierarchy)

PURPOSE

Workflow Ascendant provides the ability to implement a user stack from an organizational hierarchy. This hierarchy can be defined in the view *OU By Hierarchy* in *OU* documents or this information can be taken from an external source (in that case you would replace *#waSetUsersMGR* below with your own routine). To note that the system accounts for hierarchies of dynamic lengths. In this particular example, user *a1* creates and sends a *DOC206* document into the workflow where it progresses through the user's approval hierarchy.

WHAT IT LOOKS LIKE



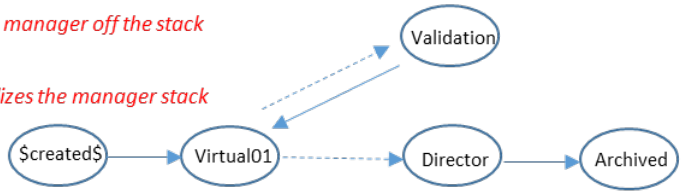
HOW TO MAKE IT HAPPEN

ManagerIndex Manager	ManagerIndex + 1 @GetField("Manager" + @If(ManagerIndex >=10; ""; "0") + @Text(ManagerIndex))

3) State Validation Actions "Approved" pops the next manager off the stack

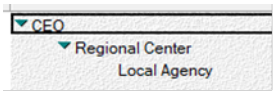
1) State $created$ Actions (#waSetUsersMngr) initializes the manager stack

	The document will be initialized with the following stocked subroutine(s)...
1	#waSetUsersMngr | #waSetUsersRole



2) State Virtual01 routes the document to Director if no other managers are left on the stack

	The following state will be routed to automatically...	If the following condition is met...
a	Validation	(0 < ManagerMax) & (ManagerIndex <= ManagerMax)
b	Director	@True



View OU By Hier.

TYPE	Organisational Unit (OU)
Organisational Unit	CEO
Manager	d1/Horizon
Members	c1/Horizon

TYPE	Organisational Unit (OU)
Organisational Unit	Regional Center
Manager	c1/Horizon
Members	b1/Horizon

TYPE	Organisational Unit (OU)
Organisational Unit	Local Agency
Manager	b1/Horizon
Members	a1/Horizon a2/Horizon a3/Horizon a4/Horizon a5/Horizon black/Horizon

Top to bottom

ADDITIONAL NOTES / REFERENCES

Reference Section 4.2 for explications regarding workflow document initialization.

3.5) State Stack

PURPOSE

Workflow Ascendant provides the ability to implement a state stack using Actions and an Extended Sequential Operation. The latter, also referred to as a *Virtual State*, essentially acts as a subroutine which is called by the other states. In the example below, the workflow for *Marketing* is *State 01* to *State 02* to *State 03* while that for *Sales* is *State 03* to *State 02* to *State 01*. These stacks are initialized when the document is first advanced in the process and reset in State document *To be modified* (not shown below).

WHAT IT LOOKS LIKE

Marketing

Reference	Title	Process	Other	Current State
green				
SST.00006		DOC207		State 01
SST.00006		DOC207		State 02
SST.00006		DOC207		State 03
SST.00006		DOC207		Director

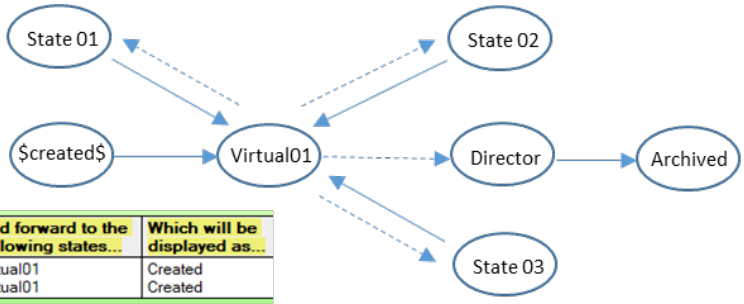
Sales

Reference	Title	Process	Other	Current State
e1				
SST.00007		DOC207		State 03
SST.00007		DOC207		State 02
SST.00007		DOC207		State 01
SST.00007		DOC207		Director

HOW TO MAKE IT HAPPEN

DOC207
- Archived
- Director
- Rejected
- State 01
- State 02
- State 03
- To be modified
- Virtual01
- Withdrawn
- ScreatedS

1) State *$created$* Actions initializes the stack and sets the first state (alias)



	The following RIs can intervene...	And forward to the following states...	Which will be displayed as...
1	[Marketing]	Virtual01	Created
2	[Sales]	Virtual01	Created

WAStepsOrig	"S3" : "S2" : "S1"
WASteps	WAStepsOrig
WAStep	@Subset(WASteps; 1)
WASteps	@If(@Elements(WASteps) = 1; ""; @Subset(WASteps; - (@Elements(WASteps) - 1)))

Sales

Marketing
"S1" : "S2" : "S3"

	The following state will be routed to automatically...	If the following condition is met...
a	State 01	WAStep = "S1"
b	State 02	WAStep = "S2"
c	State 03	WAStep = "S3"
d	Director	@True

2) State *Virtual01* Actions routes the document by *WAStep* and pops the next state off the stack

WAStep	@Subset(WASteps; 1)
WASteps	@If(@Elements(WASteps) = 1; ""; @Subset(WASteps; - (@Elements(WASteps) - 1)))

ADDITIONAL NOTES / REFERENCES

3.6) External (ERP, RDBMS …)

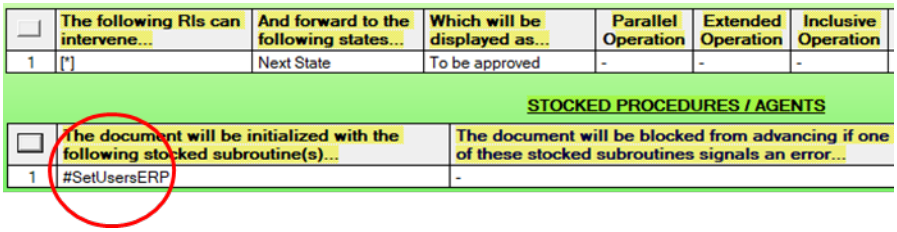
PURPOSE

Workflow Ascendant provides the ability to pull users in from an external data source. This external source could be an ERP, a relational database or any number of other directories where user information is stored. To implement extracting users from an external source, you simply complement the built-in mechanisms of Workflow Ascendant with your own.

WHAT IT LOOKS LIKE

[This applies to any of the previous examples in this Chapter which all happen on the back-end.]

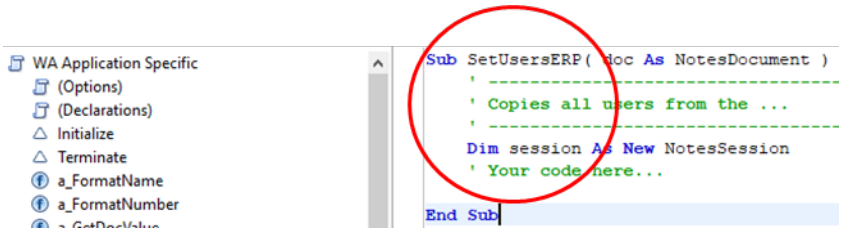
HOW TO MAKE IT HAPPEN



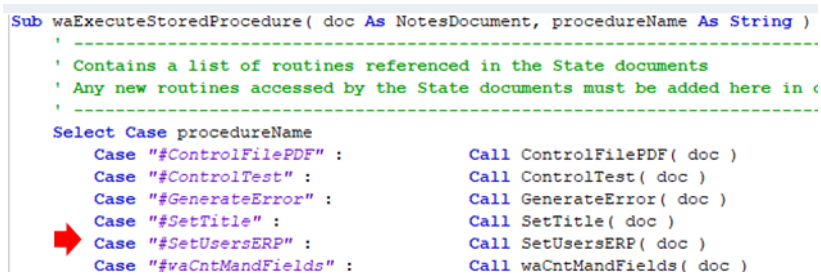
	The following RIs can intervene...	And forward to the following states...	Which will be displayed as...	Parallel Operation	Extended Operation	Inclusive Operation
1	[*]	Next State	To be approved	-	-	-

STOCKED PROCEDURES / AGENTS

	The document will be initialized with the following stocked subroutine(s)...	The document will be blocked from advancing if one of these stocked subroutines signals an error...
1	#SetUsersERP	-



Create your subroutine in script library *WA Application Specific*



Add your subroutine to *waExecuteStoredProcedure* in this same script library

ADDITIONAL NOTES / REFERENCES

Reference Section 4.2 for explications regarding workflow document initialization.

Chapter 4 – Events: Directing Execution

4.1) General

Workflow Ascendant provides the mechanisms to control every aspect of the document cycle. Controls at a more global level such as where a document is routed and who can modify a document at a particular state are treated in Chapter 2 – Process: Routing Documents and Chapter 3 – Constraints: Specifying Users. This chapter deals with what occurs when a workflow document is opened, modified and saved (sent in the workflow) including:

- Initializing the document
- Controlling who can modify which field or section of the document
- Validating field values and blocking a state advance for user data entry errors
- Validating field values prior to selecting the *Send* button (i.e., while the document is still being modified)

In addition to this, you can specify what events take place when a document has gone past a certain amount of days for a given state (or for a given section of the workflow for that matter).

Modification and validation rules are dependent by user and by state. What is modifiable and/or obligatory at one point in the process may or may not be at a different point in the process. To that end, Workflow Ascendant provides generic subroutines which are included in script library *WA Application Specific*; you complement these with your own application specific code and reference both in State documents (by rule - which implies by user and by what action is to be taken). These routines use the following two fields for all error handling:

- **WAErrorMessage** – any text put in this field will be displayed as an error and block the workflow document from changing state.
- **FieldError** – if field *WAErrorMessage* is not empty, then any field names put in this field will result in a red circle being displayed next to those fields (provided you follow the conventions in 5.3 Fields).

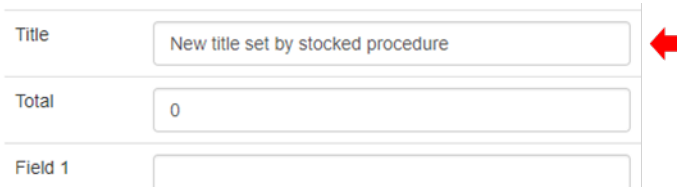
You can of course choose to use your own mechanisms.

4.2) Initialization

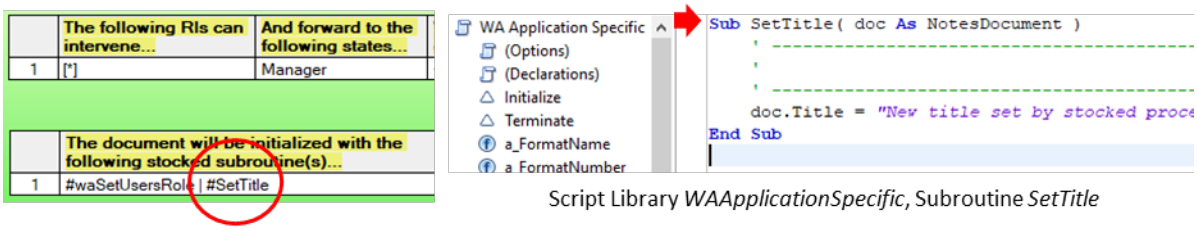
PURPOSE

Workflow Ascendant provides multiple ways to initialize field values in workflow documents. To do so, you simply complement the built-in mechanisms that Workflow Ascendant itself uses with your own. Call application specific routines in the pertinent State document using field *The document will be initialized with the following stocked subroutine(s)*. Any subroutines included in this list (each name must be preceded by #) will be executed when a user modifies a workflow document at that state – provided that the corresponding code is duly created in script library *WAApplicationSpecific*. In the example below, the *Title* field is automatically updated with stocked text when document *DOC208* is first created (state *$created$*). Note that if your code updates field *WAErrorMessage* with text, it will be displayed as an error message when the document is opened (in a Notes client, the opening of the document is also blocked).

WHAT IT LOOKS LIKE



HOW TO MAKE IT HAPPEN



Script Library *WAApplicationSpecific*, Subroutine *SetTitle*

```
Sub waExecuteStoredProcedure( doc As NotesDocument, procedureName As String )
    ' ----------------------------------------------------------------------
    ' Contains a list of routines referenced in the State documents
    ' Any new routines accessed by the State documents must be added here in
    ' ----------------------------------------------------------------------
    Select Case procedureName
        Case "#ControlFilePDF" :          Call ControlFilePDF( doc )
        Case "#ControlTest" :             Call ControlTest( doc )
        Case "#GenerateError" :           Call GenerateError( doc )
        Case "#SetTitle" :                Call SetTitle( doc )
        Case "#SetUsersERP" :             Call SetUsersERP( doc )
        Case "#waCntMandFields" :         Call waCntMandFields( doc )
```

In the same script library, include the call to your code in the following subroutine: *waExecuteStoredProcedure*

ADDITIONAL NOTES / REFERENCES

For updating document fields at state change, reference section Actions (Comments).

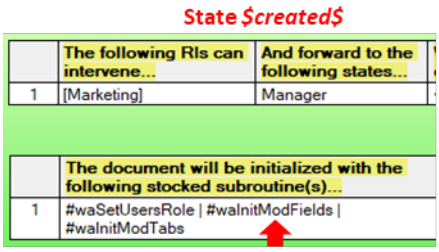
4.3) Modification Control (Field)

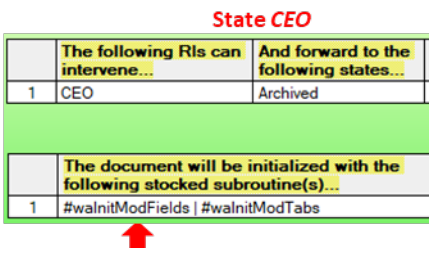
PURPOSE

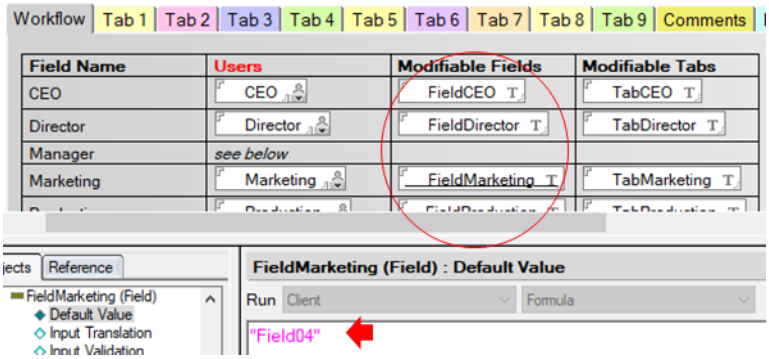
Workflow Ascendant provides ways to control who can modify which fields in a document. In the *Workflow* tab of a State document further below, fields under *Users* are set dynamically by *#waSetUsersRole* and fields under *Modifiable Fields* are used dynamically by *#walnitModFields*. To render these definitions effective, you will need to add XPages and/or Notes controls for each field which operate independently (meaning use one or the other or both depending on the clients you use to access the application). In example *DOC209* below, only *Marketing* can modify *Field04*.

WHAT IT LOOKS LIKE

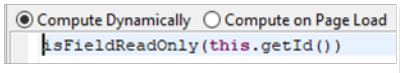
HOW TO MAKE IT HAPPEN



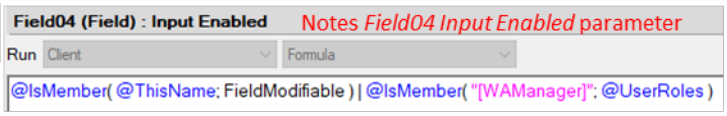




Put the field names to be controlled in fields *Field<name>*. These will be copied into field *FieldModifiable* by routine *#walnitModFields* when the corresponding RIs (under *Users*) modify the document, rendering the applicable fields modifiable in the document.



XPages *Field04 Read-only* parameter



ADDITIONAL NOTES / REFERENCES

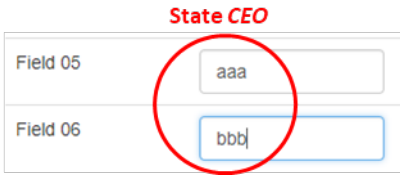
Reference Section 4.2 for explications regarding workflow document initialization.

4.4) Modification Control (Panel/Section)

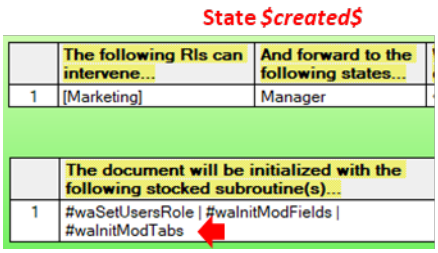
PURPOSE

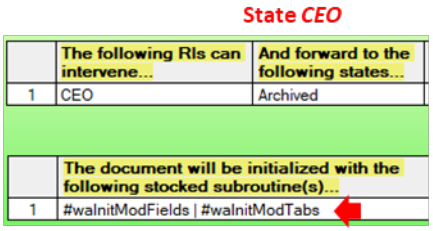
Workflow Ascendant provides ways to control who can modify which portions of a document. In the *Workflow* tab further below, fields under *Users* are set dynamically by *#waSetUsersRole* and panels under *Modifiable Tabs* are used dynamically by *#waInitModTabs*. To render these definitions effective, you will need to add XPages and/or Notes controls for each panel/section which operate independently (meaning implement those applicable to the clients you use to access the application). In example *DOC209* below, only *CEO* can modify *Tab03*.

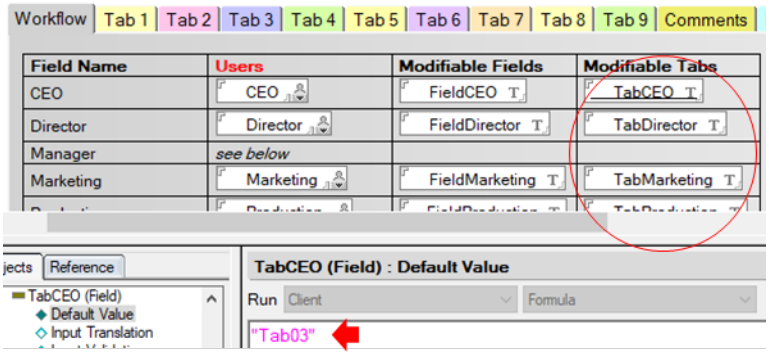
WHAT IT LOOKS LIKE



HOW TO MAKE IT HAPPEN

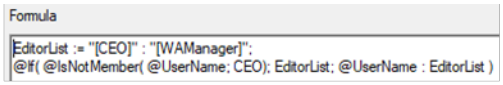






Put the tab names to be controlled in fields *Tab<name>*. These will be copied into field *TabModifiable* by routine *#waInitModTabs* when the corresponding RIs (under *Users*) modify the document, rendering the applicable panels or sections modifiable in the document.

XPages *Content_DocWF01 Tab03 Read-only* parameter



Notes Section Control *Tab03 Computed for display* parameter

ADDITIONAL NOTES / REFERENCES

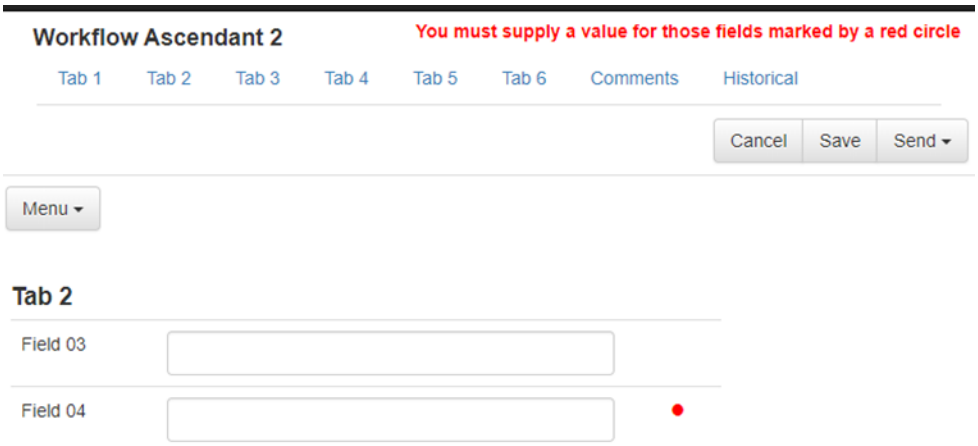
Reference Section 4.2 for explications regarding workflow document initialization.
isTabReadOnly(this.getId())

4.5) Validation Control (General)

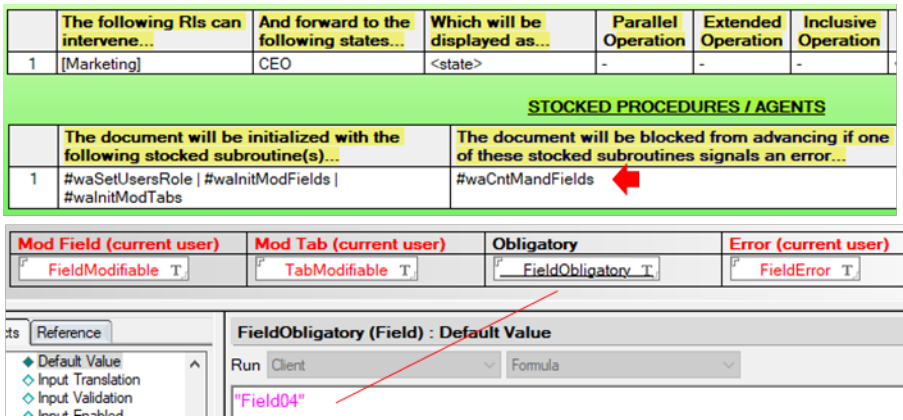
PURPOSE

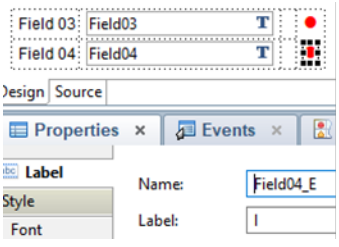
Workflow Ascendant provides a general, built-in field validation mechanism. It verifies *by state* that designated fields contain a value - but only if the current user has the ability to modify those fields. To that end, this mechanism must be used in conjunction with Modification Control (Field). Should this not meet your requirements, reference the following section Validation Control (Custom). In the example *DOC210* below, the *Field04* field must contain a value for the document to advance. *Note that [WAManager] is **not** controlled by default!*

WHAT IT LOOKS LIKE

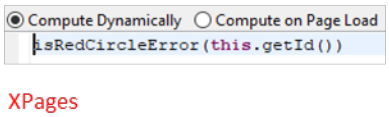


HOW TO MAKE IT HAPPEN



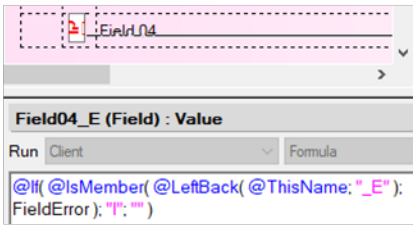


XPages



(Wingding font)

Notes



ADDITIONAL NOTES / REFERENCES

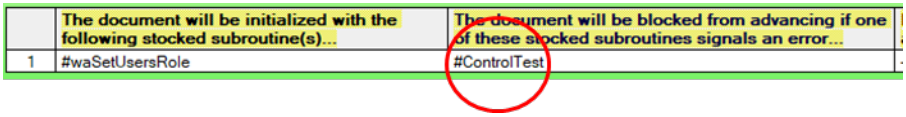
4.6) Validation Control (Custom)

PURPOSE

Workflow Ascendant provides for custom validation mechanisms. Call your application specific subroutines in the pertinent State document using field ***The document will be blocked from advancing if…***. Any subroutines included in this list will be executed when the *Send* button is selected at that state – provided that the corresponding code is duly created in script library *WAApplicationSpecific*. In the example below, *Field 1* and *Field 2* must contain identical values when document *DOC211* is forwarded (state *$created$*).

WHAT IT LOOKS LIKE

HOW TO MAKE IT HAPPEN



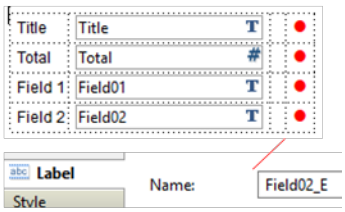
	The document will be initialized with the following stocked subroutine(s)...	The document will be blocked from advancing if one of these stocked subroutines signals an error...
1	#waSetUsersRole	#ControlTest

Script Library *WAApplicationSpecific*

```
Sub ControlTest( doc As NotesDocument )
    ' ------------------------------------
    '
    ' ------------------------------------
    Dim fieldErrorL As New ItemList

    ' Initialize variables
    ' --------------------
    errorMessage = ""

    ' Get the error field info
    ' ------------------------
    If ( doc.Field01( 0 ) <> doc.Field02( 0 )) Then
        Call fieldErrorL.PushOnItem( "Field01" )
        Call fieldErrorL.PushOnItem( "Field02" )
        If Not fieldErrorL.IsListEmpty() Then errorMessage = "Fields marked w
    End If

    ' Update the document
    ' -------------------
    doc.WAErrorMessage = errorMessage
    Call List_SetDocFieldFromList( doc, "FieldError", fieldErrorL )
    Call waSetRedCirclesNotes( doc )
End Sub
```

```
Sub waExecuteStoredProcedure( doc As NotesDocument, procedureName As S
    ' ------------------------------------------------------------------
    ' Contains a list of routines referenced in the State documents
    ' Any new routines accessed by the State documents must be added h
    ' ------------------------------------------------------------------
    Select Case procedureName
        Case "#ControlFilePDF" :        Call ControlFilePDF( doc )
        Case "#ControlTest" :           Call ControlTest( doc )
        Case "#ControlTotal" :          Call ControlTotal( doc )
```

Custom Control



ADDITIONAL NOTES / REFERENCES

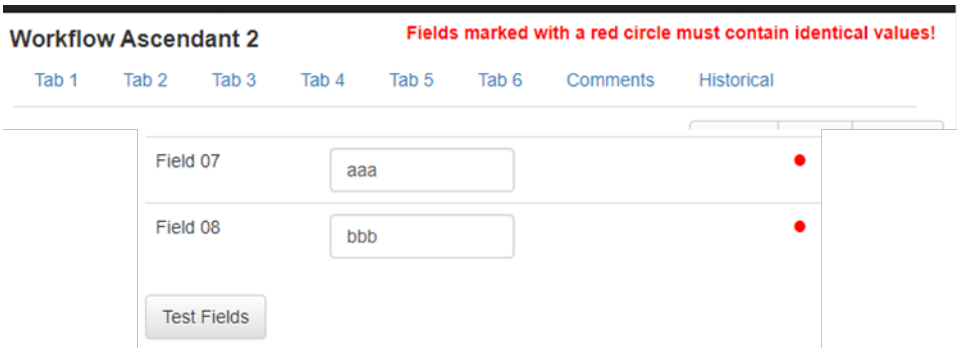
For information regarding the *ItemList* class, reference Appendix E – List Classes.

4.7) In-State Controls

PURPOSE

Workflow Ascendant provides a template from which you can control events which occur outside of a state change. The agent *myInStateFieldControl* is provided as a model you can use to customize for your own specific application requirements. In example *DOC211* below, an error condition is flagged when button *Test Fields* is selected if fields *Field07* and *Field08* do not contain identical values.

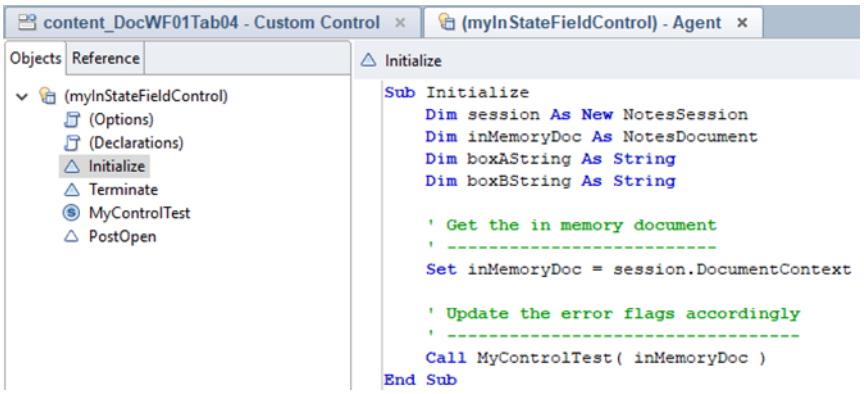
What it looks like



How to make it happen

```
var doc:NotesDocument = document1.getDocument(true)
var docVirtual = database.createDocument()
var docVirtualHolderAgent = database.getAgent("(myInStateFieldControl)")

docVirtual.replaceItemValue("Field07", doc.getItemValue("Field07"))
docVirtual.replaceItemValue("Field08", doc.getItemValue("Field08"))

docVirtualHolderAgent.runWithDocumentContext(docVirtual)

viewScope.waErrorMessage = docVirtual.getItemValueString("WAErrorMessage")
viewScope.waErrorFieldNames = docVirtual.getItemValue("FieldError")
var noErrors:Boolean = ((viewScope.waErrorMessage == "" ) || (viewScope.waErrorMessage == null )
```

Button *Test Fields*, *onClick* event



This agent executes subroutine *MyControlTest* (essentially identical to that in the previous section) on the in-memory (virtual) document

Additional Notes / References

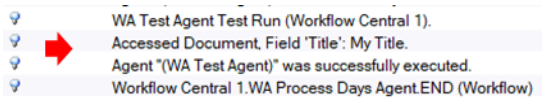
For additional information on the field configuration of the "red circle" mechanism, refer to Section 5.3.

4.8) Timed Events

PURPOSE

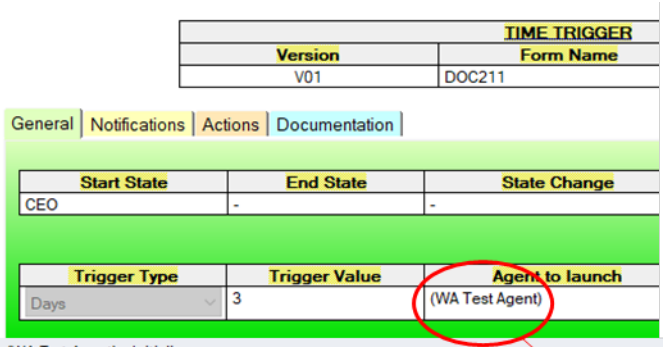
Workflow Ascendant provides the capability to launch events based on elapsed time. These are defined in Time Trigger documents in the *Workflow - Time Triggers* view. In example *DOC211* below, the application log is updated by agent *(WA Test Agent)* when the document remains in state *CEO* for *3* days. In addition to launching agents, you can specify stocked subroutines (the names of which to be preceded by #) in script library *WA Application Specific*.

WHAT IT LOOKS LIKE



Text written to the agent log provided that this has been created and configured in the Language document.

HOW TO MAKE IT HAPPEN



In this example, all fields in *Notifications* and *Actions* are empty. To display alerts and/or route documents as timed events, reference Sections 2.17) Alert Timeouts and 2.18) Time Based Routing.

(WA Test Agent) - Initialize

```
Option Public
Use "WA Utility Routines"
Sub Initialize
    ' ------------------------------------------------------------------------
    '                                                    Agent (WA Test Agent)
    ' ------------------------------------------------------------------------
    Dim docCurrent As NotesDocument
    Dim agent As NotesAgent

    ' Initialize classes
    ' ------------------
    Set agent = wagv_Session.CurrentAgent

    ' Get the document from which the agent was launched
    ' --------------------------------------------------
    If Not ( agent.ParameterDocID = "" ) Then
        Set docCurrent = wagv_DBCurrent.GetDocumentByID( agent.ParameterDocID )
    End If

    ' Write entry to the log
    ' ----------------------
    Call LogWrite( wagv_LogJournal, agent.Name & " Test Run (" & wagv_Applicati
    If ( docCurrent Is Nothing ) Then
        Call LogWrite( wagv_LogJournal, "Intentional Error: the current documen
    Else
        Call LogWrite( wagv_LogJournal, "Accessed Document, Field 'Title': " &
    End If
End Sub
```

ADDITIONAL NOTES / REFERENCES

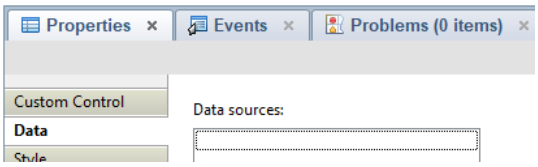
Chapter 5 – Content: Managing Data

5.1) General

This chapter addresses document content - the information to be routed in the process. Described in detail in the previous chapter, Workflow Ascendant uses a system to facilitate controlling who can modify which portions of a document in addition to what information the document must contain. This chapter exposes the mechanics of that system as well as walk through various data structures commonly found in workflow documents:

- Various field types
- Simple dynamic lists
- Tiered dynamic lists
- Tables
- Bar Charts
- Gantt Chart
- Selective visibility

Custom Control *zcontent_DocFields* contains the basic XPages field types that are available. If you are not familiar with XPages, you may want to consider coping/pasting fields from there into your Custom Control along with the "red circles" in order to take advantage of the Workflow Ascendant Error Handling mechanism. That, of course, is your choice. If you do choose to take components directly from the *Controls* menu, take care to ensure that you do not have any *Data sources* specified when you save the Custom Control.



No *Data sources* specified. In the Workflow Ascendant design schema, no "content" Custom Control (i.e., those contained in a workflow document) should contain a data source.

It is recommended that you create new XPages and Custom Controls from existing ones. From there it is a relatively straightforward process to change the names referenced therein in the XML Source panes/tabs.

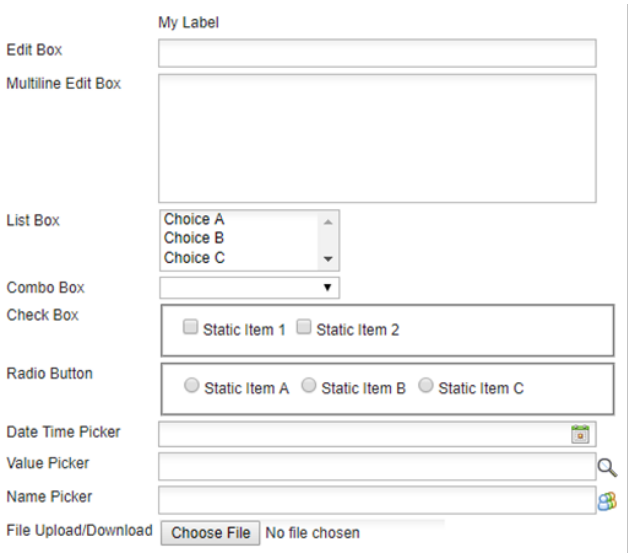
The Design and Source panes. This example is taken from the context of creating new XPage *Document02* from *Document01*.

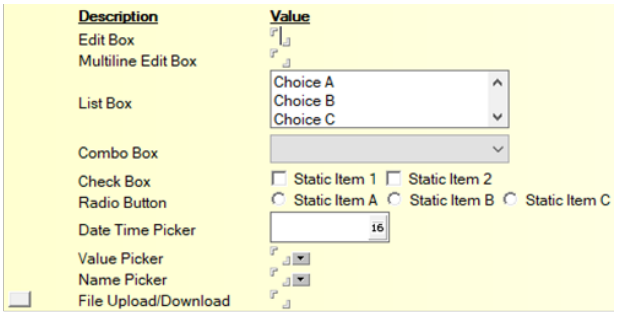
5.2) Field Types

PURPOSE

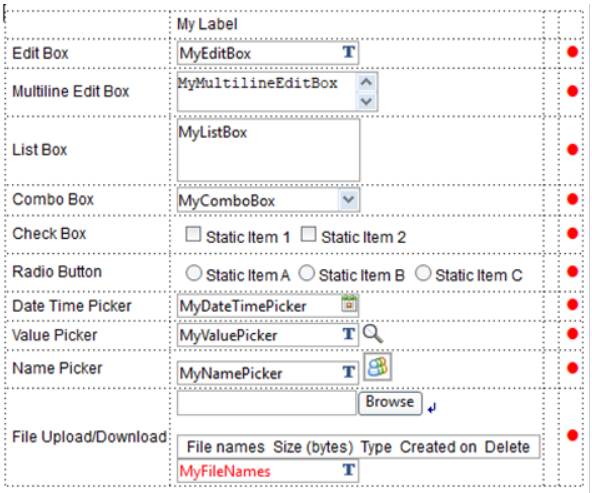
There is a variety of field types at your disposal. Notes Form *Document02 Tab 2* and XPage *Tab 2* (Custom Control *zcontent_DocFields*) contain the primary / most frequently used ones that you can copy and paste from.

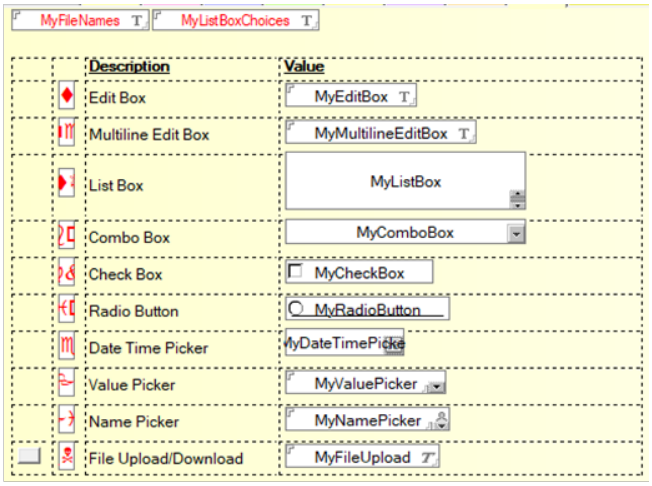
WHAT IT LOOKS LIKE





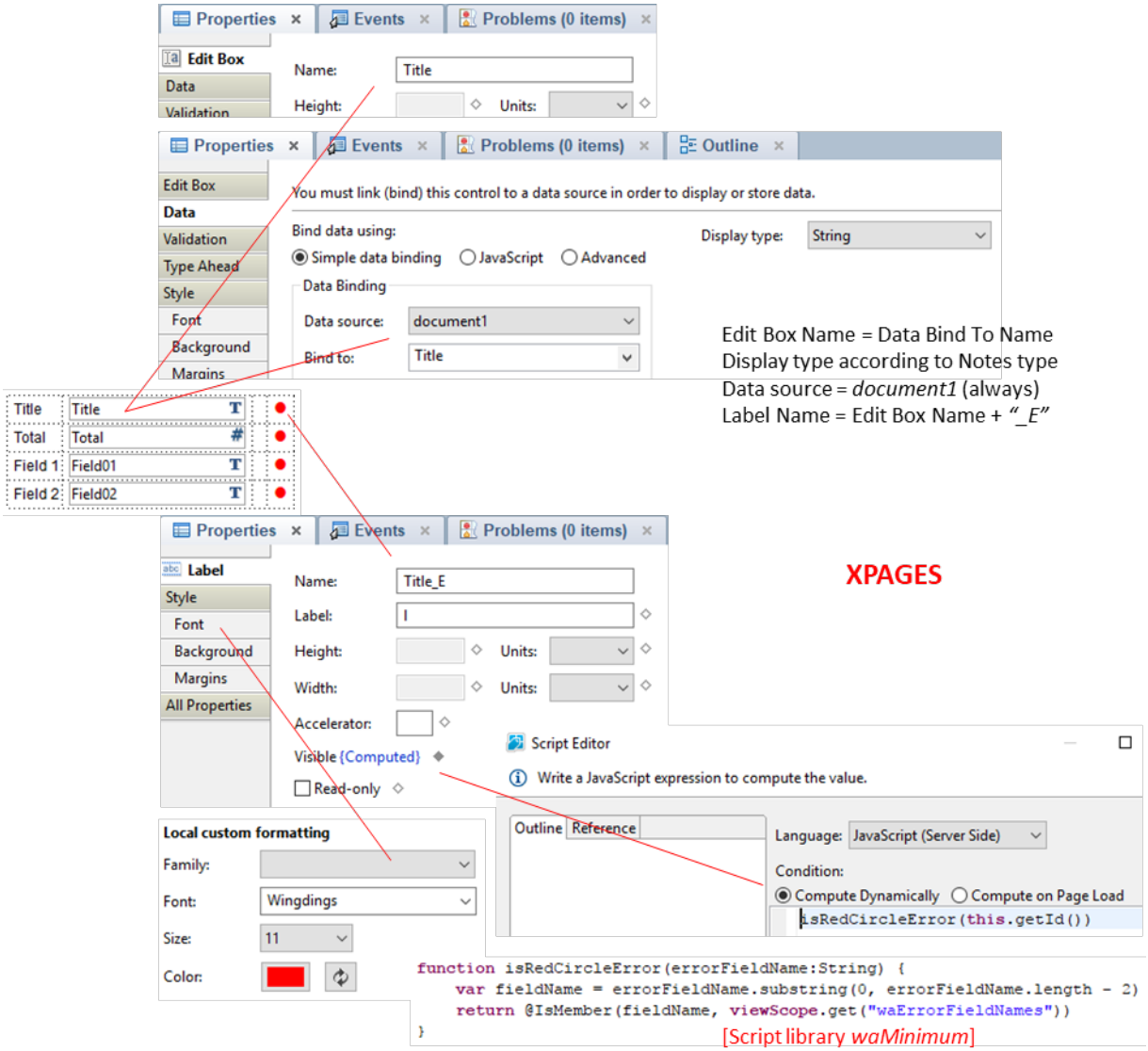
HOW TO MAKE IT HAPPEN

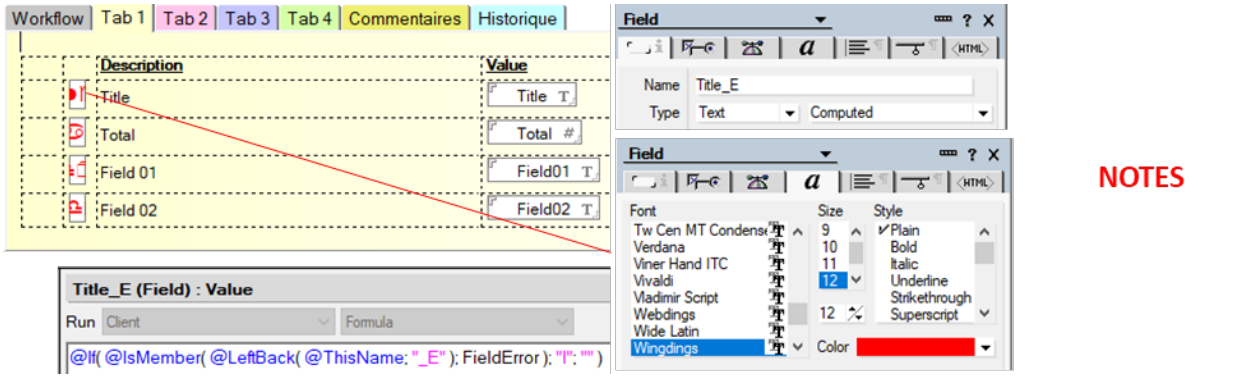




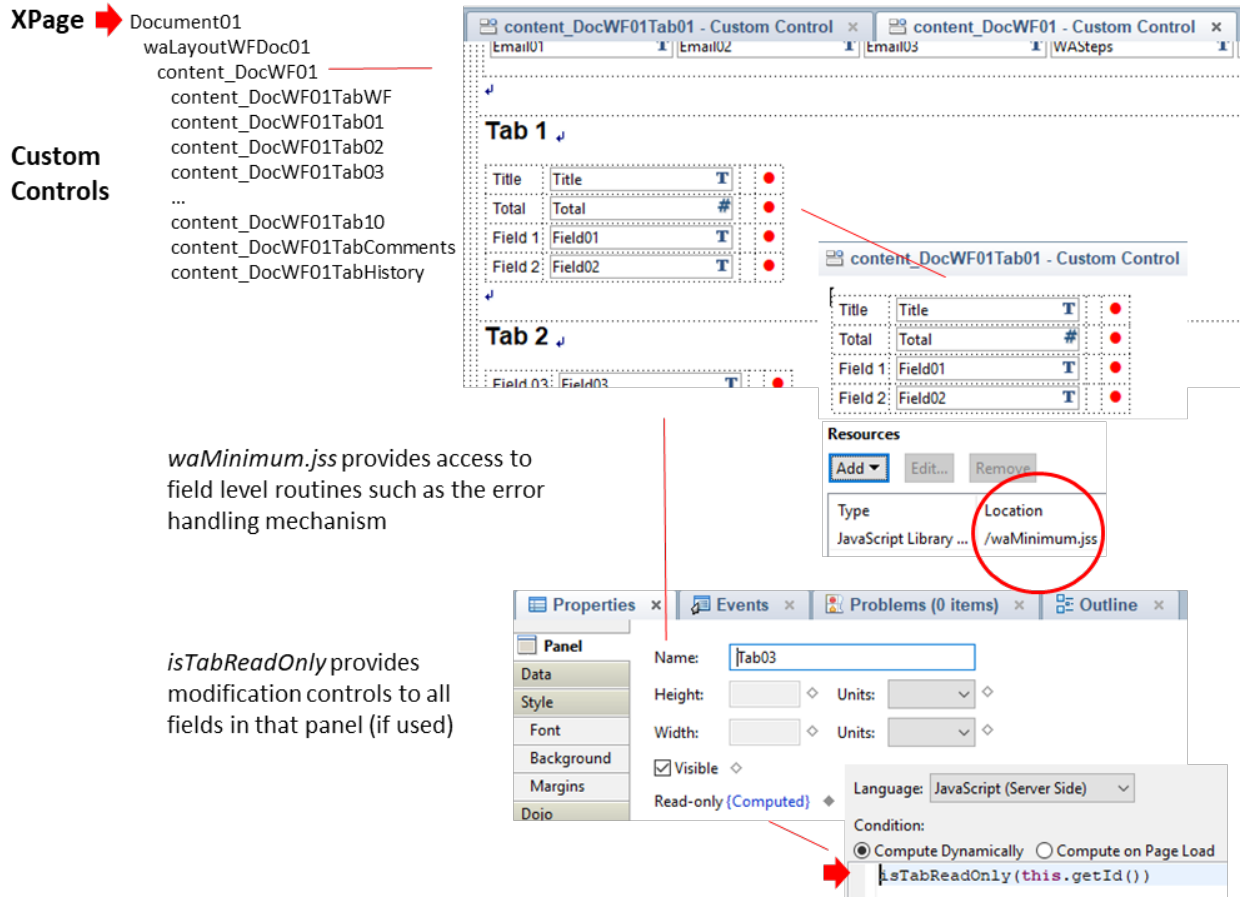
ADDITIONAL NOTES / REFERENCES

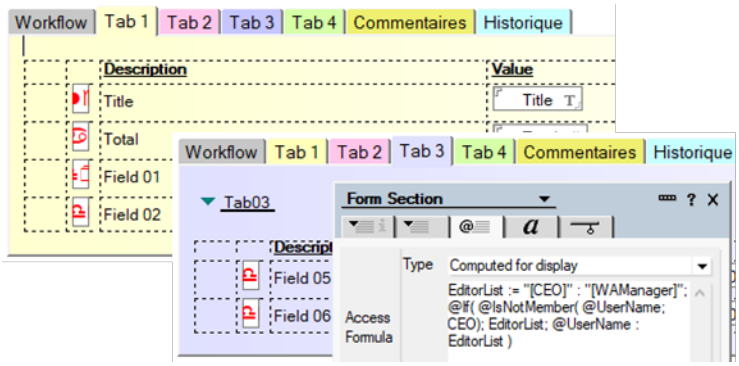
5.3) Fields


Properties
Events
Problems (0 items)
Outline
Edit Box
Name: Title
Height:
Units:
Data
Validation
Type Ahead
Style
Font
Background
Margins
You must link (bind) this control to a data source in order to display or store data.
Bind data using:
Simple data binding
JavaScript
Advanced
Display type: String
Data Binding
Data source: document1
Bind to: Title
Edit Box Name = Data Bind To Name
Display type according to Notes type
Data source = document1 (always)
Label Name = Edit Box Name + "_E"
Title
Total
Field 1
Field 2
Field01
Field02
Label
Name: Title_E
Label: I
All Properties
Accelerator:
Visible {Computed}
Read-only
XPAGES
Script Editor
Write a JavaScript expression to compute the value.
Local custom formatting
Family:
Font: Wingdings
Size: 11
Color:
Outline
Reference
Language: JavaScript (Server Side)
Condition:
Compute Dynamically
Compute on Page Load
isRedCircleError(this.getId())
function isRedCircleError(errorFieldName:String) {
var fieldName = errorFieldName.substring(0, errorFieldName.length - 2)
return @IsMember(fieldName, viewScope.get("waErrorFieldNames"))
}
[Script library waMinimum]


Workflow
Tab 1
Tab 2
Tab 3
Tab 4
Commentaires
Historique
Description
Value
Title
Total
Field 01
Field 02
Field01
Field02
Field
Name Title_E
Type Text
Computed
Font
Size
Style
Tw Cen MT Condens
Verdana
Viner Hand ITC
Vivaldi
Vladimir Script
Webdings
Wide Latin
Wingdings
Plain
Bold
Italic
Underline
Strikethrough
Superscript
Color
Title_E (Field) : Value
Run Client Formula
@If(@IsMember(@LeftBack(@ThisName; "_E"); FieldError); "I"; "")
NOTES

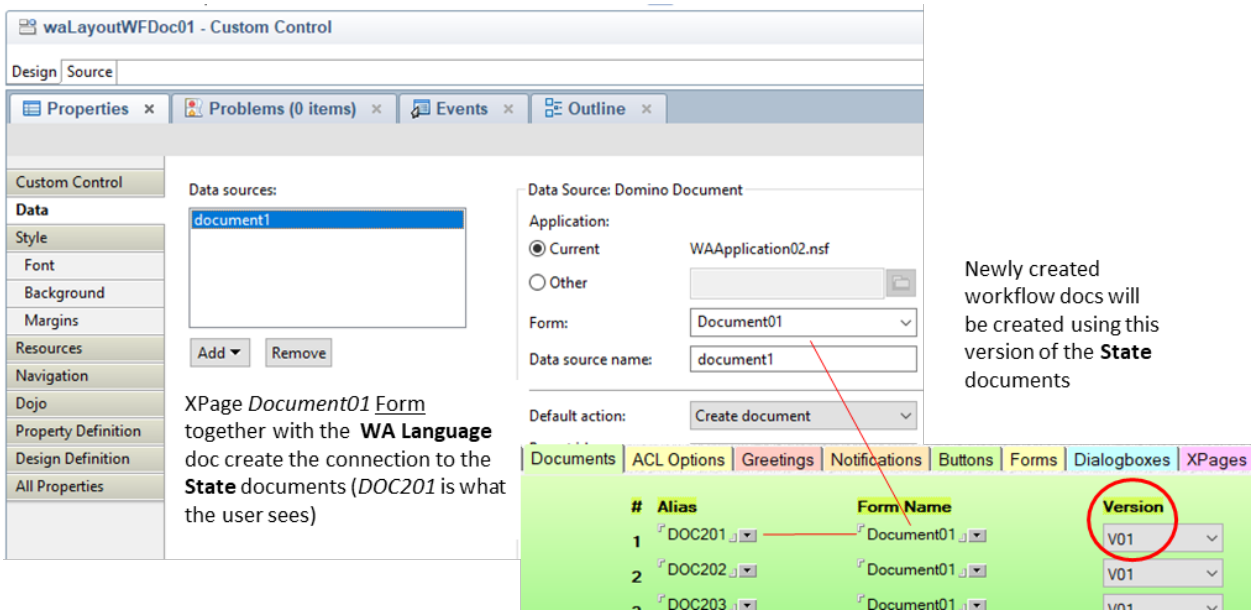
5.4) Panels / Sections





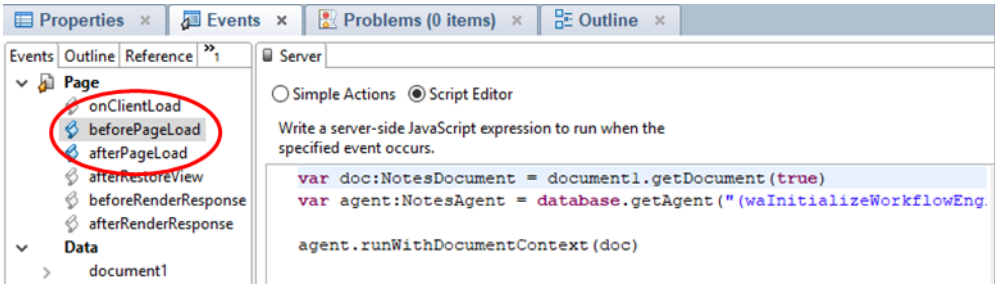
NOTES

5.5) Document Layouts

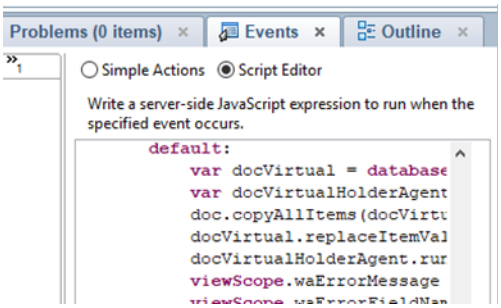


XPage *Document01* Form together with the **WA Language** doc create the connection to the **State** documents (*DOC201* is what the user sees)

Newly created workflow docs will be created using this version of the **State** documents



The workflow document is initialized here



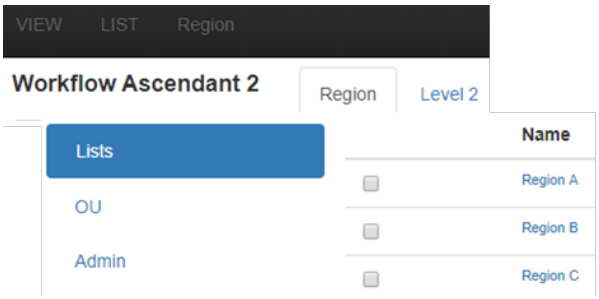
The workflow document is "sent" here

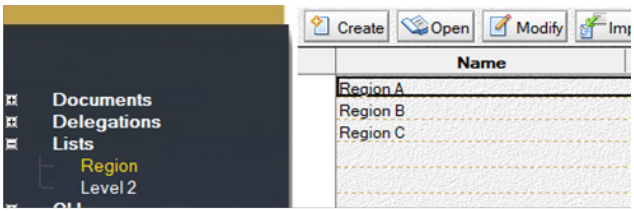
5.6) Simple List Creation

PURPOSE

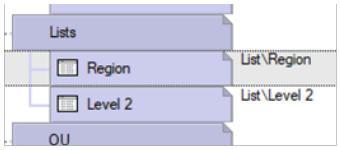
Providing lists for users to select from minimizes data entry error. Making those lists dynamic (i.e., easily modifiable) facilitates application maintenance. See Section 5.7 Simple List Usage for how to make these lists available for user selection in the application. This example converts generic "*List 1*" to "*Region*".

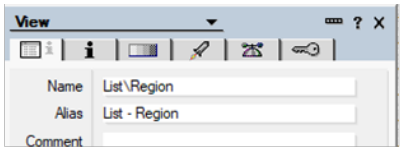
WHAT IT LOOKS LIKE





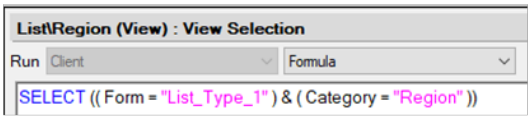
HOW TO MAKE IT HAPPEN

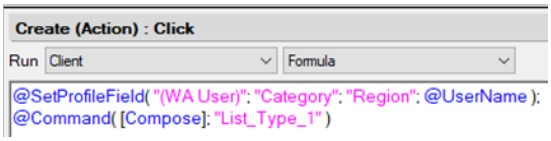


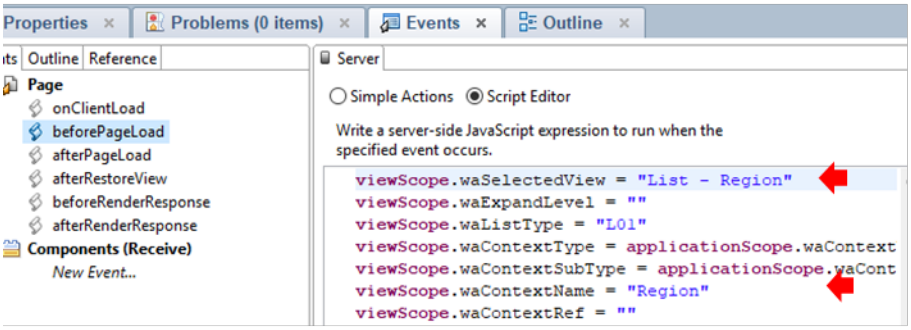


View
List – Level 1

Outline
WA Main

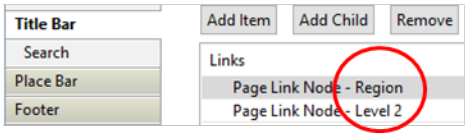




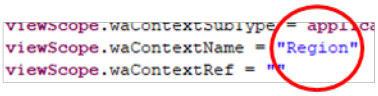


WALSViewL01

waLayoutLSView



WALSDocL01

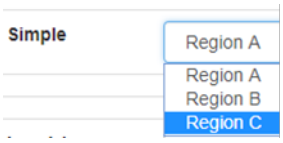


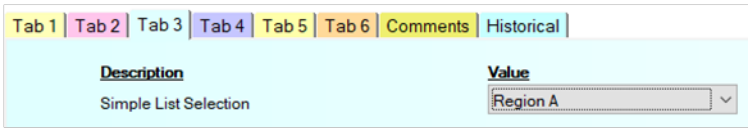
5.7) Simple List Usage

PURPOSE

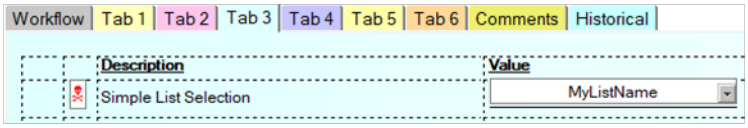
Providing lists for users to select from minimizes data entry error. Making those lists dynamic (i.e., easily modifiable) facilitates application maintenance and minimizes user data entry errors. See Section 5.6) Simple List Creation for how to create these lists. This can be seen in action by creating a document in process *DOC212* (*Document02 Tab 3*). This example converts generic "*List 1*" to "*Region*".

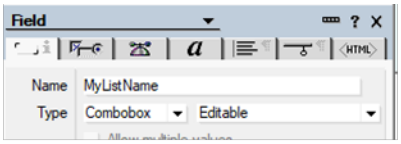
WHAT IT LOOKS LIKE

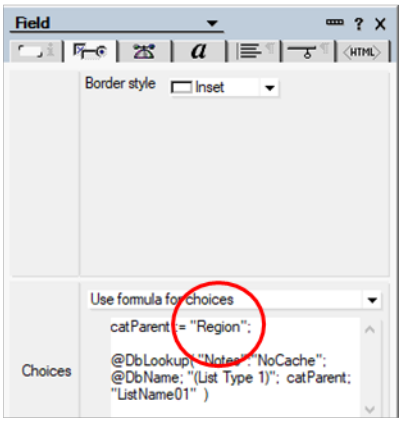


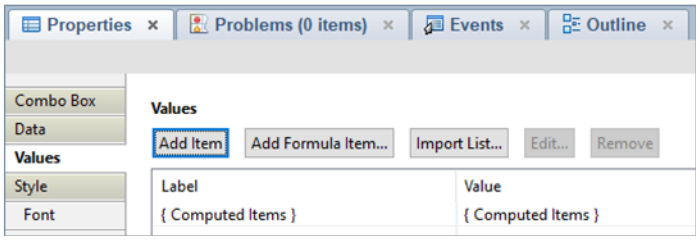


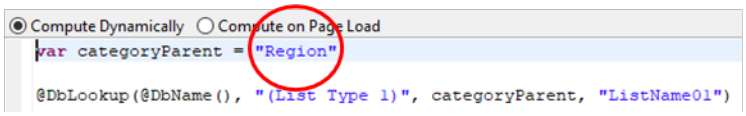
HOW TO MAKE IT HAPPEN









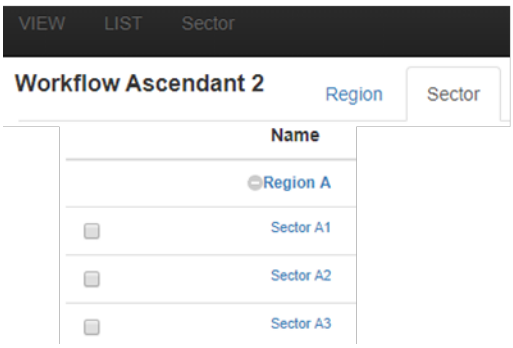


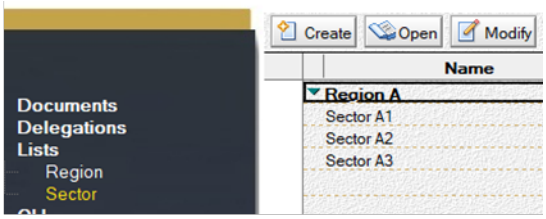
5.8) Tiered List Creation

PURPOSE

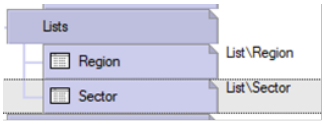
Providing lists for users to select from minimizes data entry error. Making those lists dynamic (i.e., easily modifiable) facilitates application maintenance. See Section 5.9) Tired List Usage for how to make these lists available for user selection in the application. This example converts generic "*List 2*" to "*Sector*".

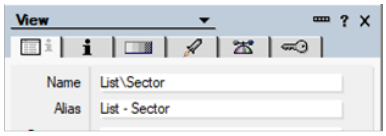
WHAT IT LOOKS LIKE





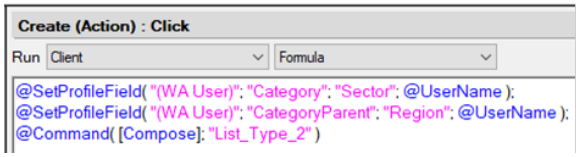
HOW TO MAKE IT HAPPEN

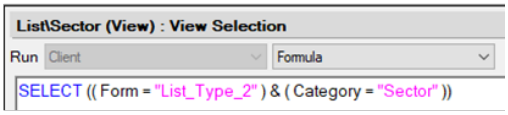


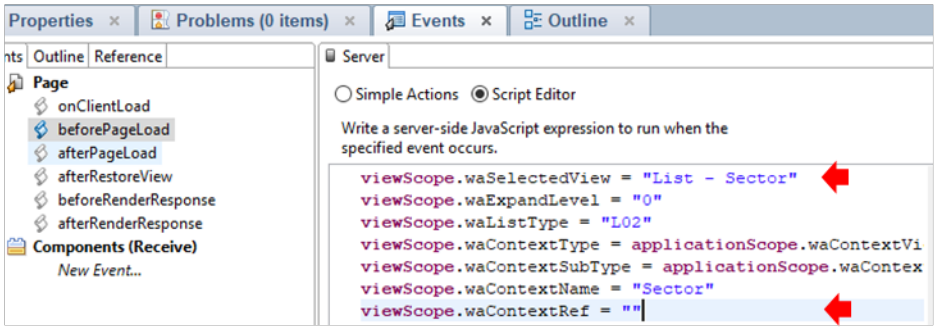


View
List – Level 2

Outline
WA Main

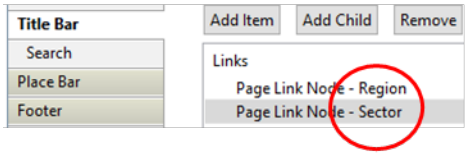




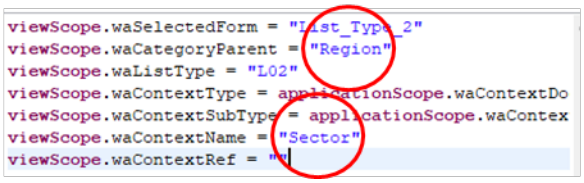


WALSViewL02

waLayoutLSView



WALSDocL02

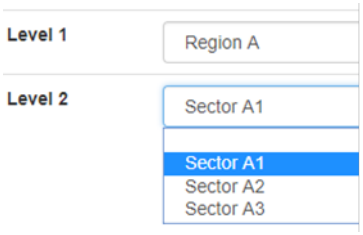


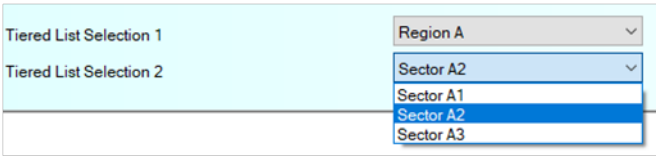
5.9) Tiered List Usage

PURPOSE

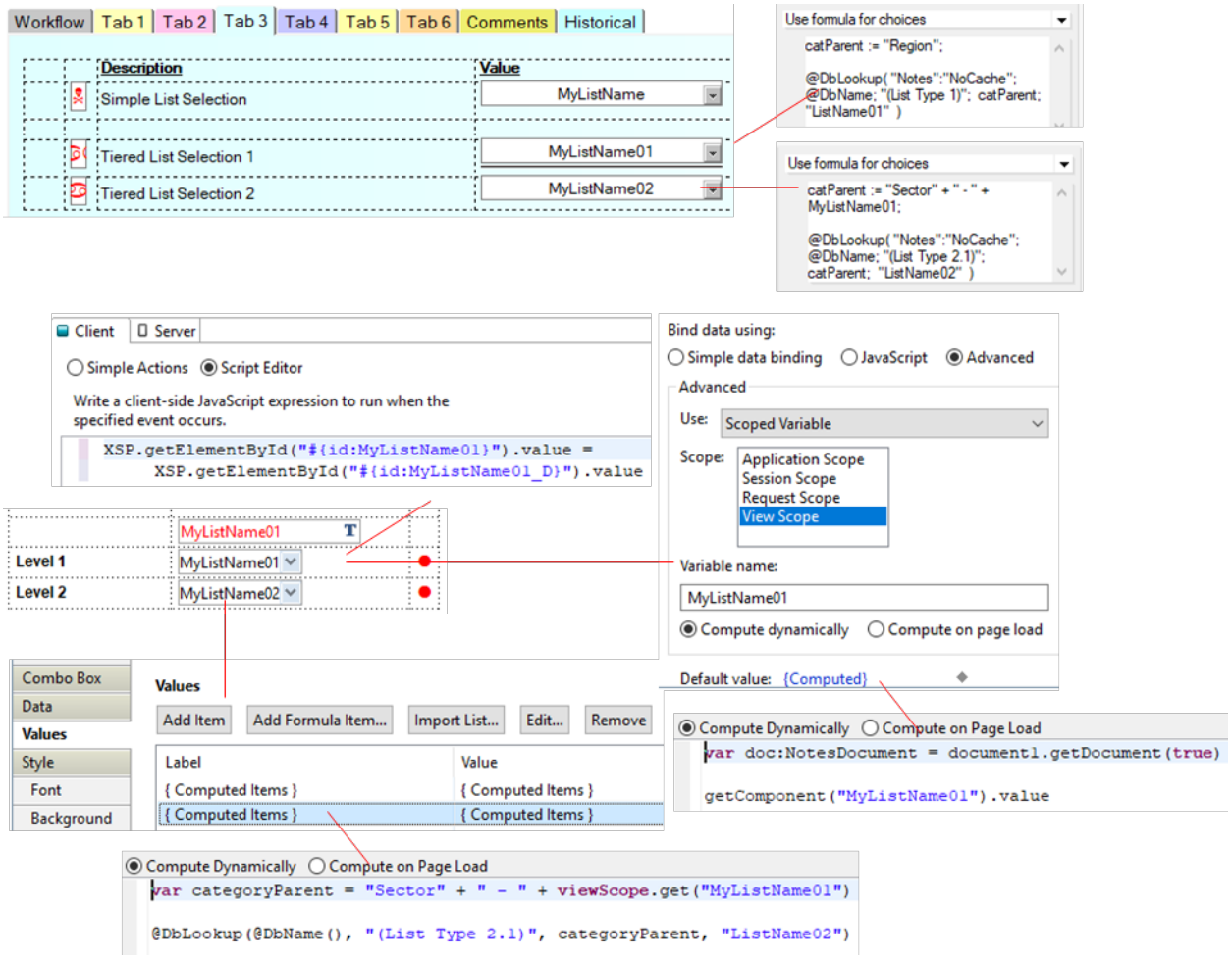
Providing lists for users to select from minimizes data entry error. Making those lists dynamic (i.e., easily modifiable) facilitates application maintenance and minimizes user data entry errors. See Section 5.8) Tiered List Creation for how to create these lists. This can be seen in action by creating a document in process *DOC212* (*Document02 Tab 3*). This example converts generic "*List 2*" to "*Sector*".

WHAT IT LOOKS LIKE





HOW TO MAKE IT HAPPEN

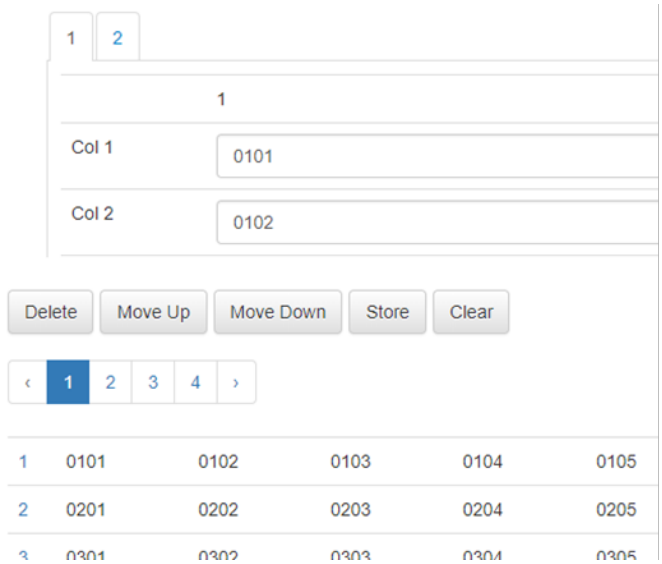


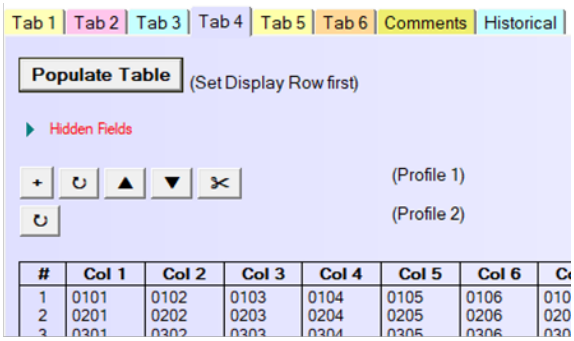
5.10) Tables

PURPOSE

Workflow Ascendant provides a convenient way to insert tables into applications. Insert the matching Custom Control and Subform into the desired panels/tabs of the application and set the configuration fields to display the table accordingly. To note that obligatory fields must be of the format "*T01_DBField04*" for table *1*, column *4* (example). This can be seen in action by creating a document in process *DOC211* (*Document01 Tab 6*).

WHAT IT LOOKS LIKE

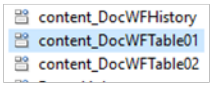




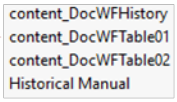
Populate Table is a mechanism to automatically fill in the table (for testing purposes only)

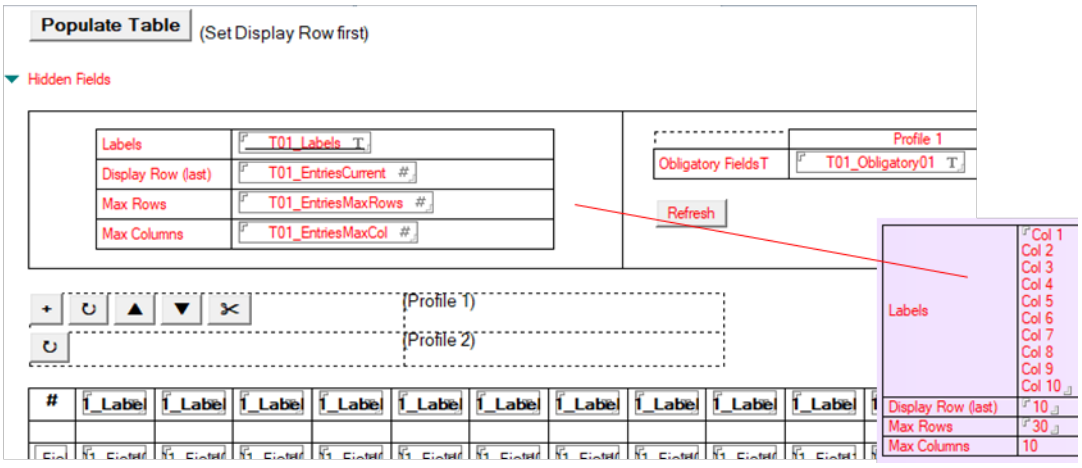
HOW TO MAKE IT HAPPEN

Custom Control *content_DocWFTable01*



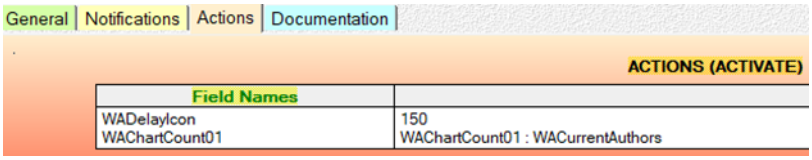
Subform *content_DocWFTable01*



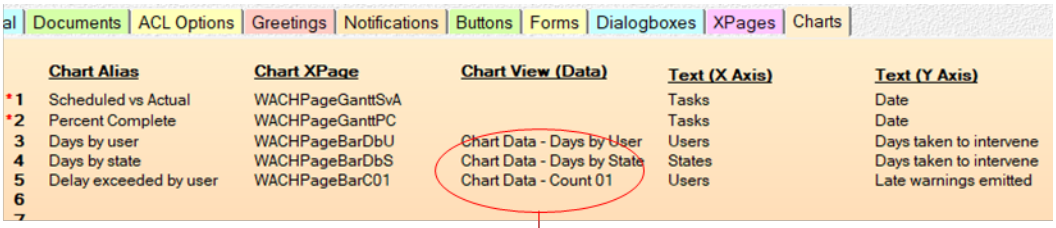


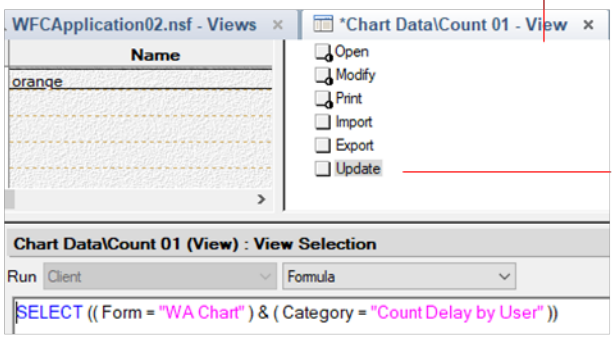
5.11) Bar Charts

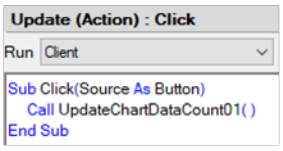
Bar Charts is a generic mechanism to display statistics graphically. The example below provided by default will produce a bar chart displaying statistics on users who exceed their allotted time to intervene in workflow documents. Intermediate documents are created in view *Chart Data\Count 01* from the workflow documents which, in turn, are used to generate the graphics (this to optimize performance for end users). Agent *WA Update Chart Docs Count01* creates those documents nightly, but they can also be updated manually by *[WA Manager]*. Create additional statistics by adding the relevant information to the WA Language document, creating the new view and adding the relevant routines (corresponding to *UpdateChartDataCount01* and *ChartSetCount01*) in script library *WA Chart*.



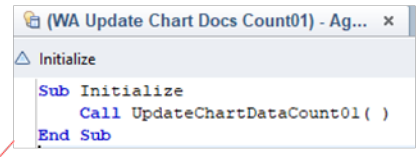
This Time Trigger Action adds all the current authors of a workflow doc to field *WAChartCount01* when the allocated time has been

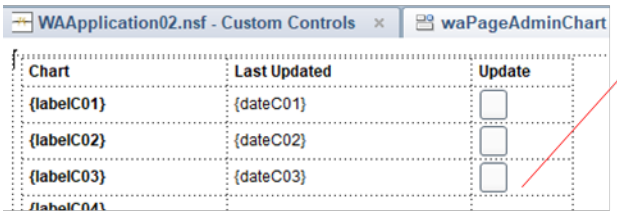




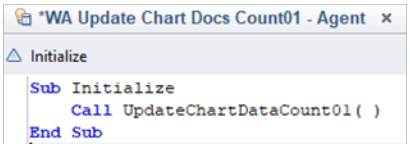


Run as Web user





Scheduled

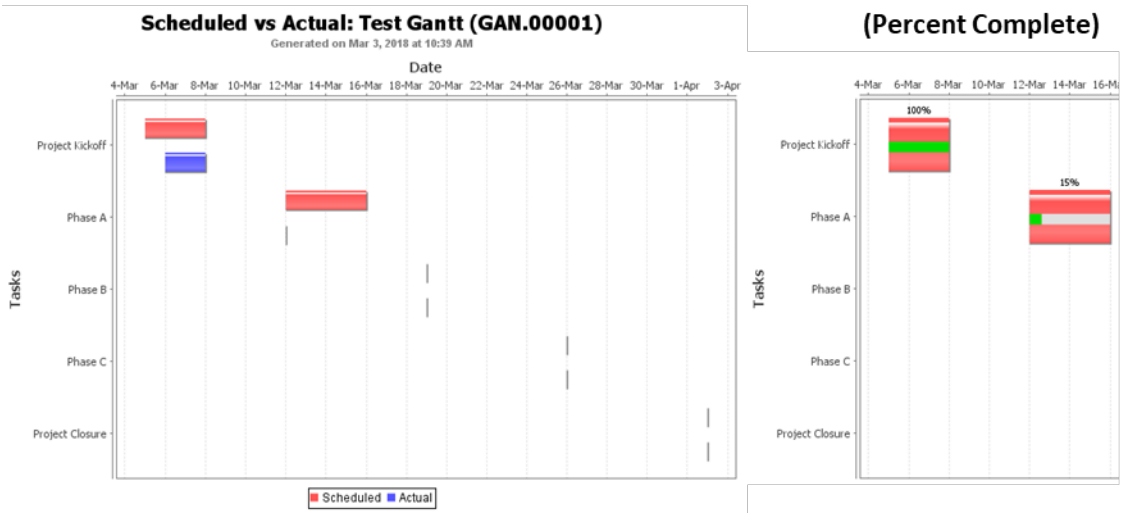


5.12) Gantt Charts

PURPOSE

Gantt Charts is a plug-and-play mechanism to display project milestone progress graphically. To that end two different charts are provided: by date and by % progress. In the example below, a document in process *DOC213* (*Document02* – reference the Language document) is configured to produce the below Gantt chart.

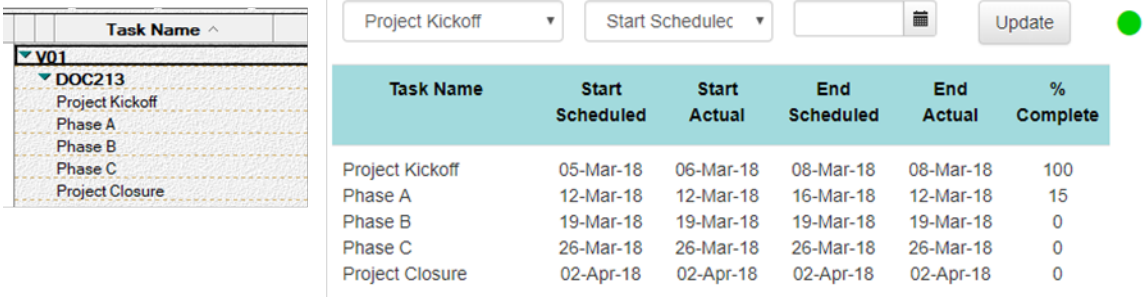
WHAT IT LOOKS LIKE



HOW TO MAKE IT HAPPEN

To render this functionality operational, the following needs to take place:

- Create the Gantt Task Name documents in view *Workflow - Gantt Tasks*
- Insert Subform *content_DocWFGantt* into the Notes form (*Document01* for example)
- Insert Custom Control *content_DocWFGantt* into the desired panel of *content_DocWF01* (example)
- In state *$created$* (example), insert the initialization routine *#waInitGanttFields*.
- Use the provided input mechanisms to modify all default dates (*01-Jan-00*) accordingly - this in turn will change the ready indicator from yellow to green which allows the chart to display



Task Name	Start Scheduled	Start Actual	End Scheduled	End Actual	% Complete
Project Kickoff	05-Mar-18	06-Mar-18	08-Mar-18	08-Mar-18	100
Phase A	12-Mar-18	12-Mar-18	16-Mar-18	12-Mar-18	15
Phase B	19-Mar-18	19-Mar-18	19-Mar-18	19-Mar-18	0
Phase C	26-Mar-18	26-Mar-18	26-Mar-18	26-Mar-18	0
Project Closure	02-Apr-18	02-Apr-18	02-Apr-18	02-Apr-18	0

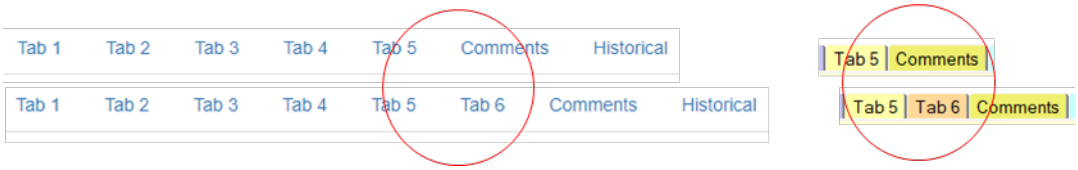
ADDITIONAL NOTES / REFERENCES

5.13) Panel/Tab Visibility

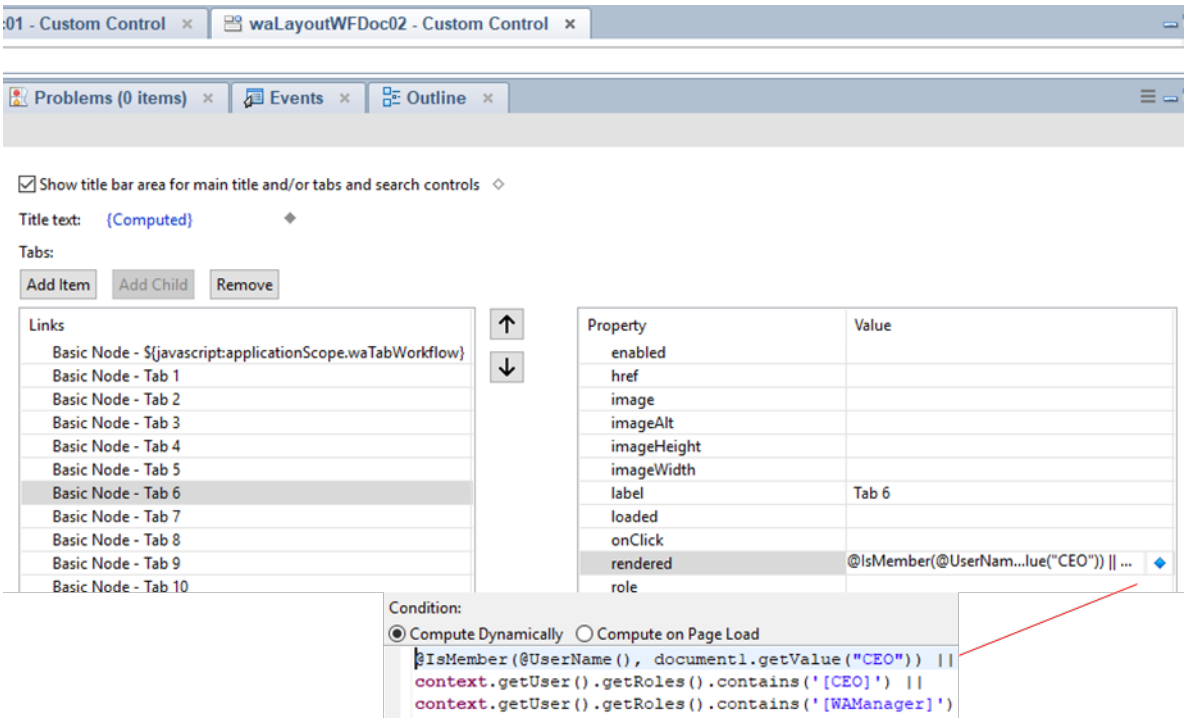
PURPOSE

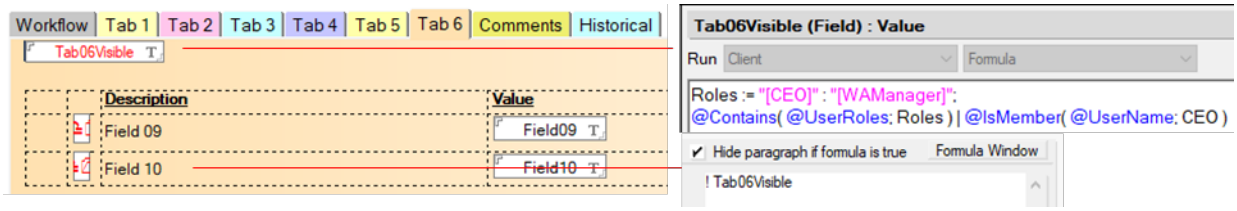
There are times when some portions of a document need to be selectively displayed depending on the user's profile. To note that the mechanisms for a Web client and the Notes client are independent. In example *DOC214* below, only *CEO* and *[WAManager]* can view *Tab 6*.

WHAT IT LOOKS LIKE



HOW TO MAKE IT HAPPEN





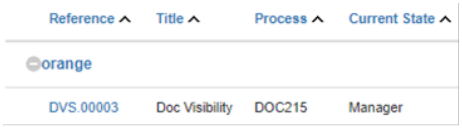
ADDITIONAL NOTES / REFERENCES

5.14) Document Visibility

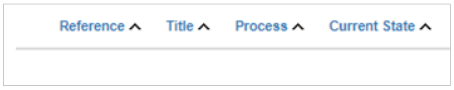
PURPOSE

There are times when some documents need to be selectively displayed depending on the user's profile. To note that there is one mechanism that works for both a Web client and the Notes client. In example *DOC215* below, users can only see documents that are assigned to them and that progressively as documents advance in the process. Note that *[WAManager]* and *[WASupervisor]* can see all documents at all times due to the initial assignment in state *$created$*.

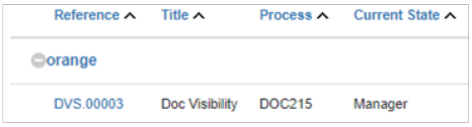
WHAT IT LOOKS LIKE



User *a1* creates the document

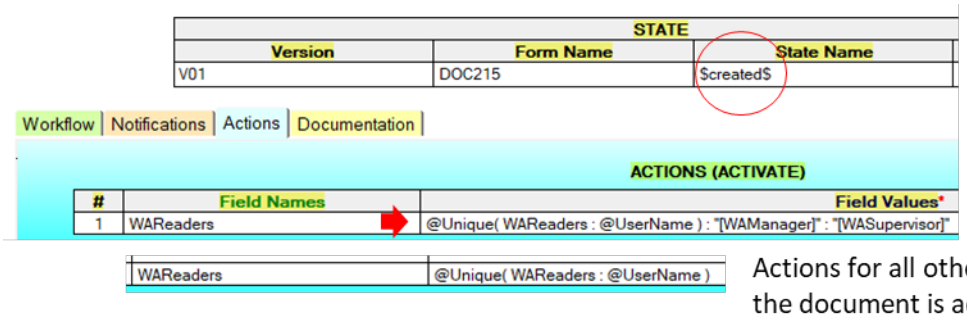


Another user (*b1*) can't see the document
Director (*red*) can't see the document yet



Manager (*orange*) advances the document
Director (*red*) can now see the document

HOW TO MAKE IT HAPPEN



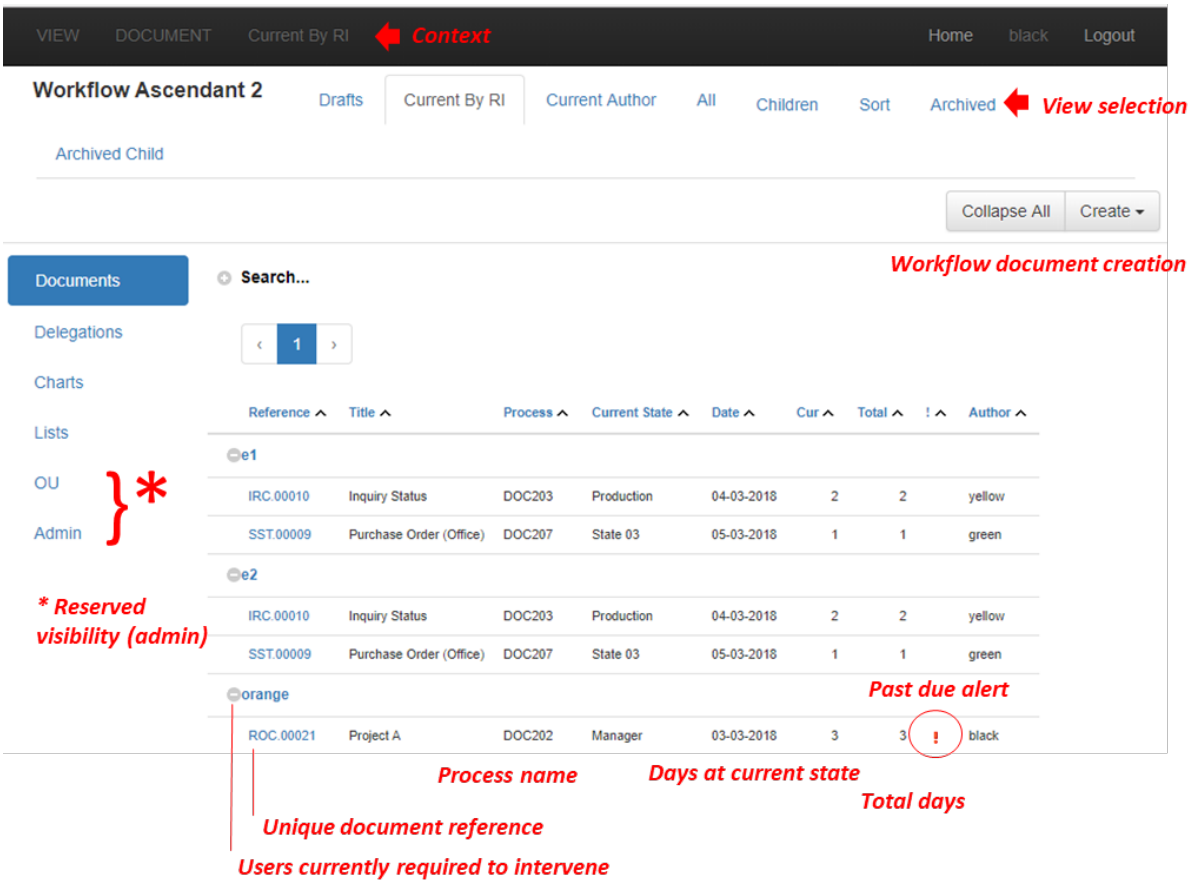
Actions for all other states when the document is advanced

ADDITIONAL NOTES / REFERENCES

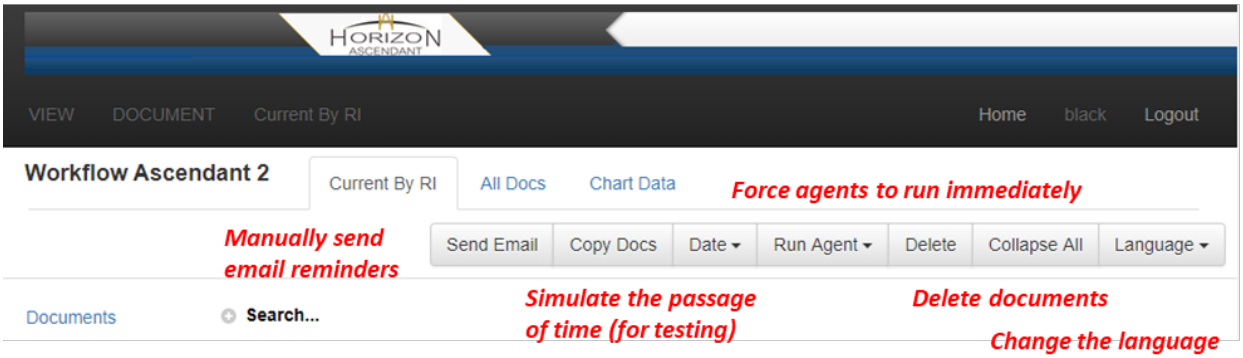
Reference Section 2.14 for explications regarding specifying Actions in State documents.

Chapter 6 – General References

6.1) General Web Interface

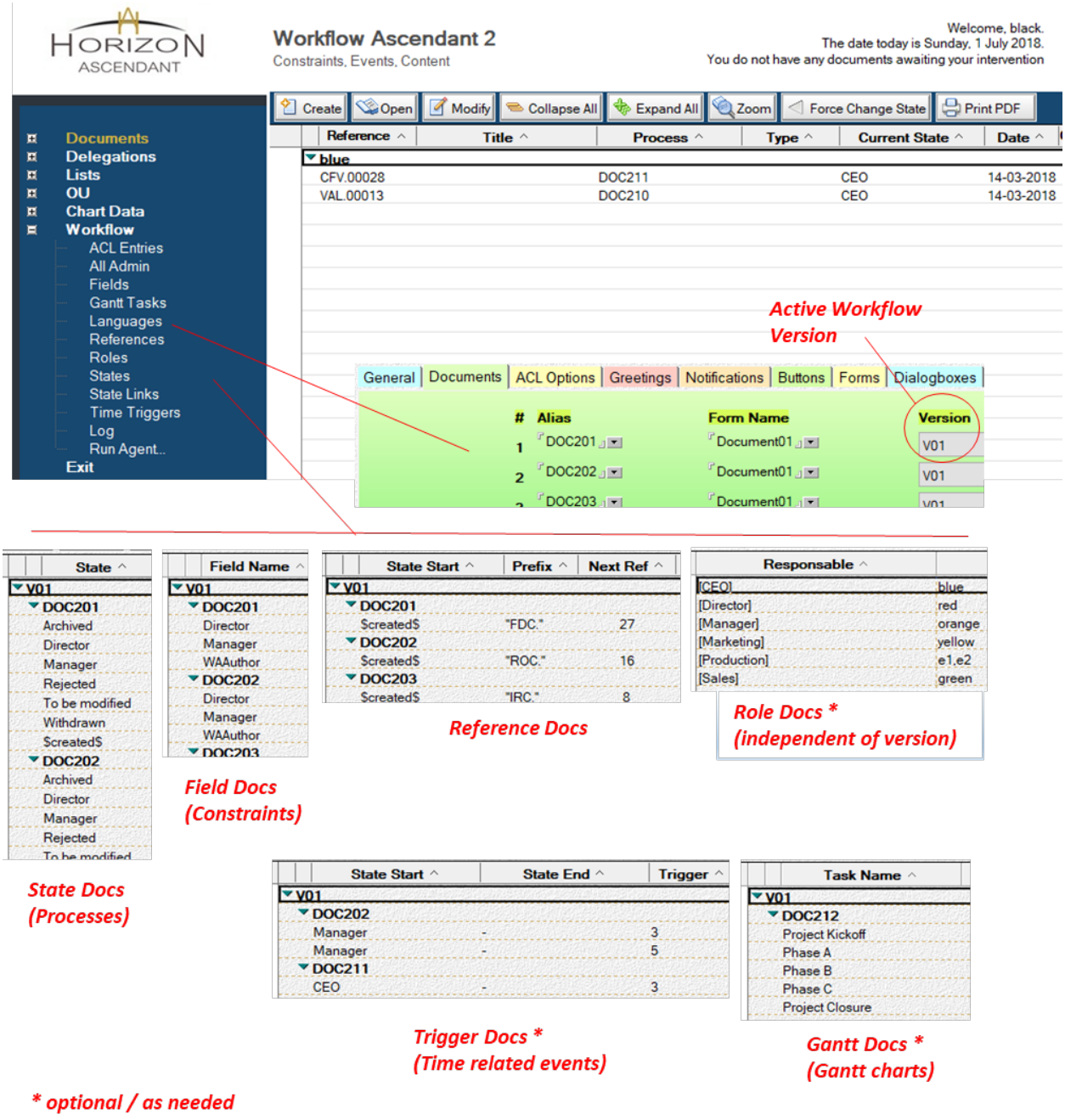


ADMIN INTERFACE

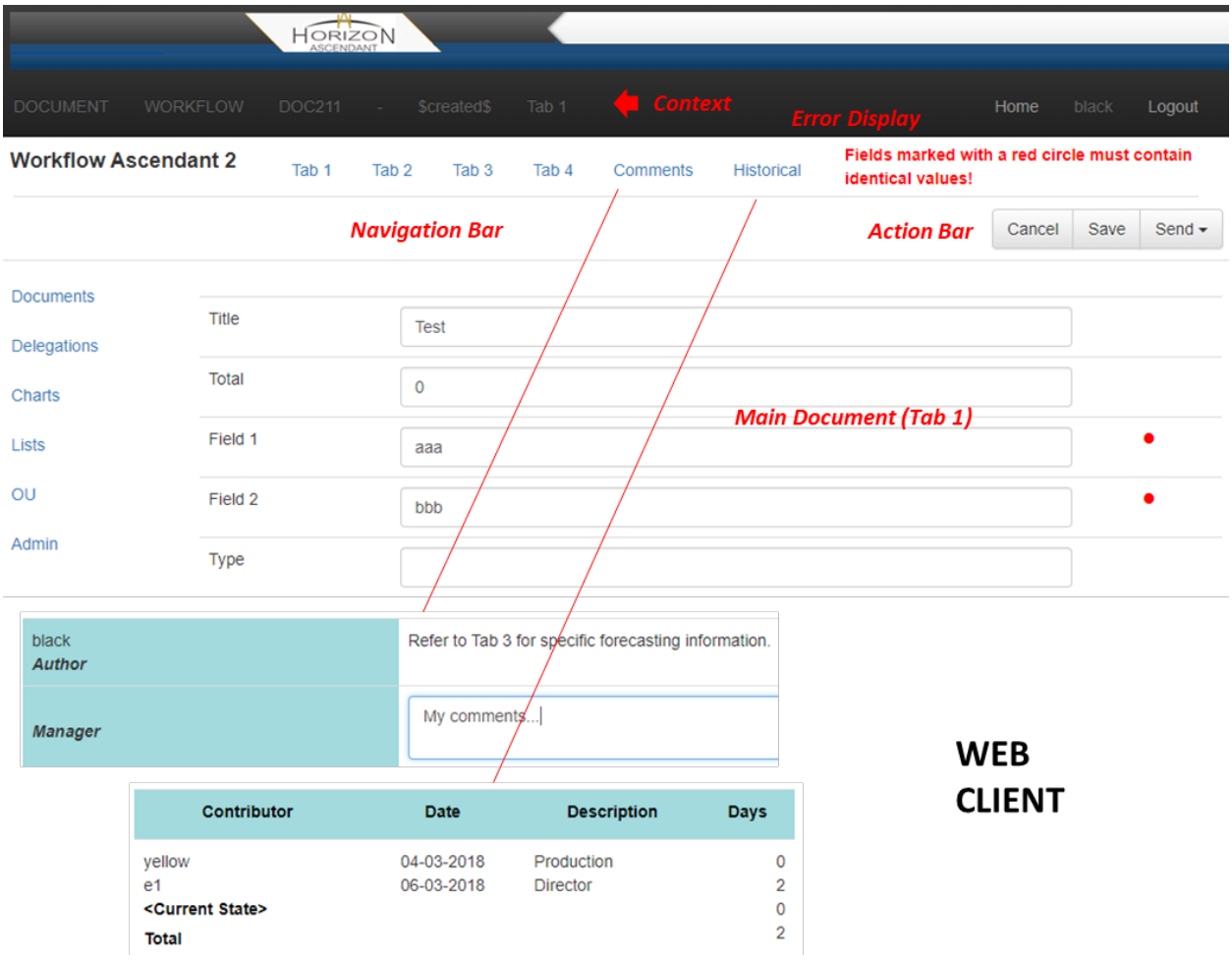


6.2) Notes Admin Interface

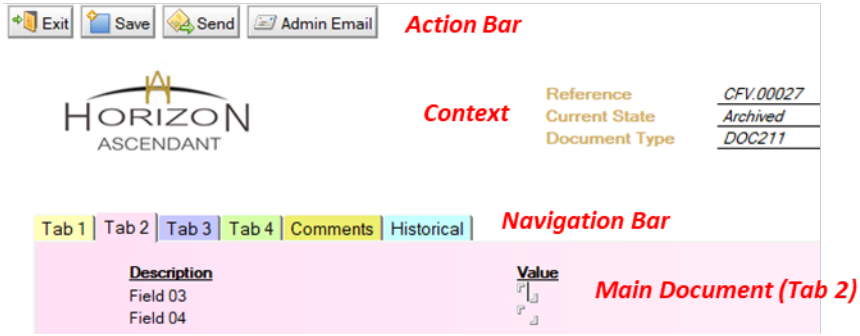
The various graphics below present the development administration interface as it relates to managing the process.



6.3) Workflow Document



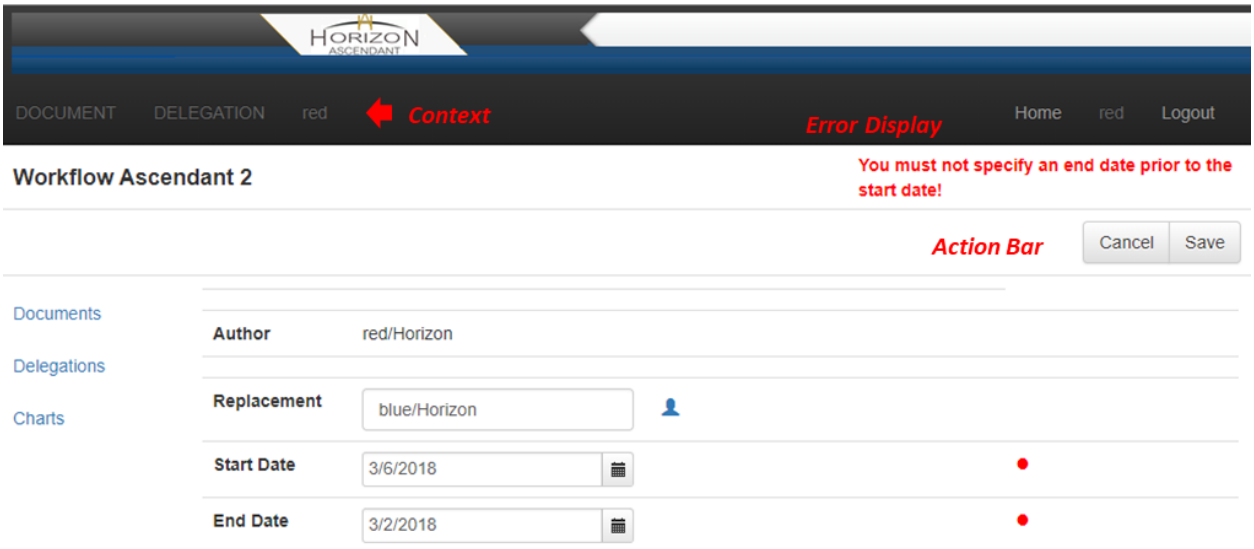
WEB CLIENT

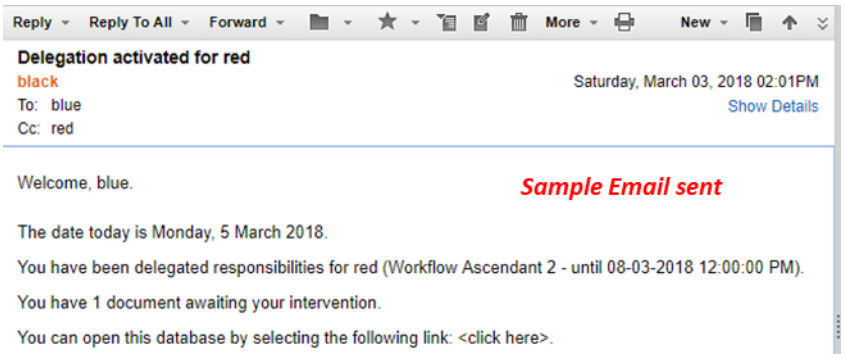


NOTES CLIENT

6.4) Delegation Document

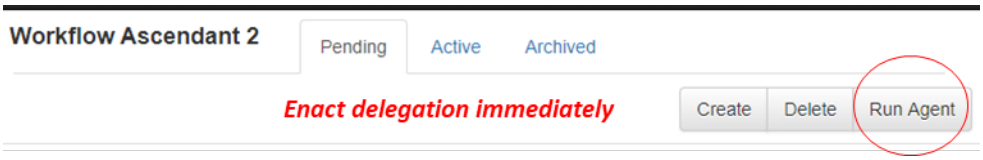
Workflow Ascendant provides real-time delegation. Users create a Delegation document for themselves indicating the time frame of their absence along with the person to be designated to. The database manager can create Delegation documents for others. When the delegation becomes effective, the delegated to party receives an email to that effect (and vice-versa when the delegation period has passed).





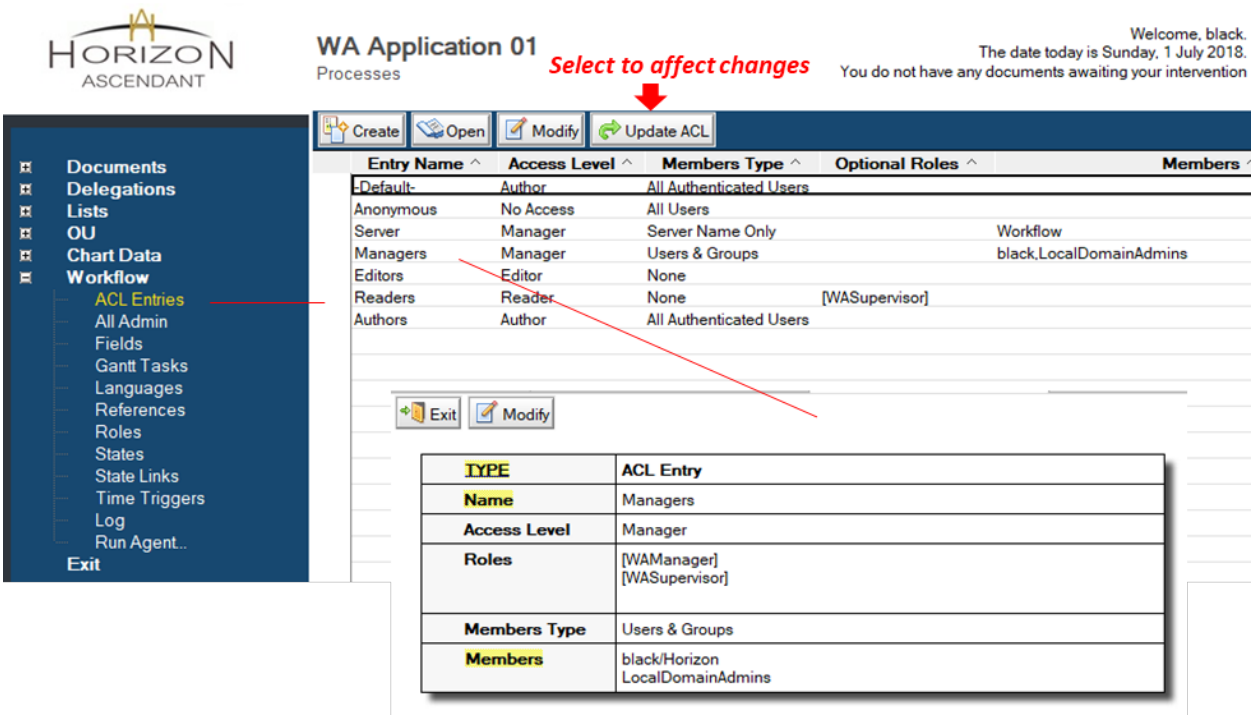
Note that *[WAManager]* can modify the *Author* field (above) to create Delegation documents for other users

While the agent to enact delegation normally executes at night, the database manager can activate this manually to take immediate effect



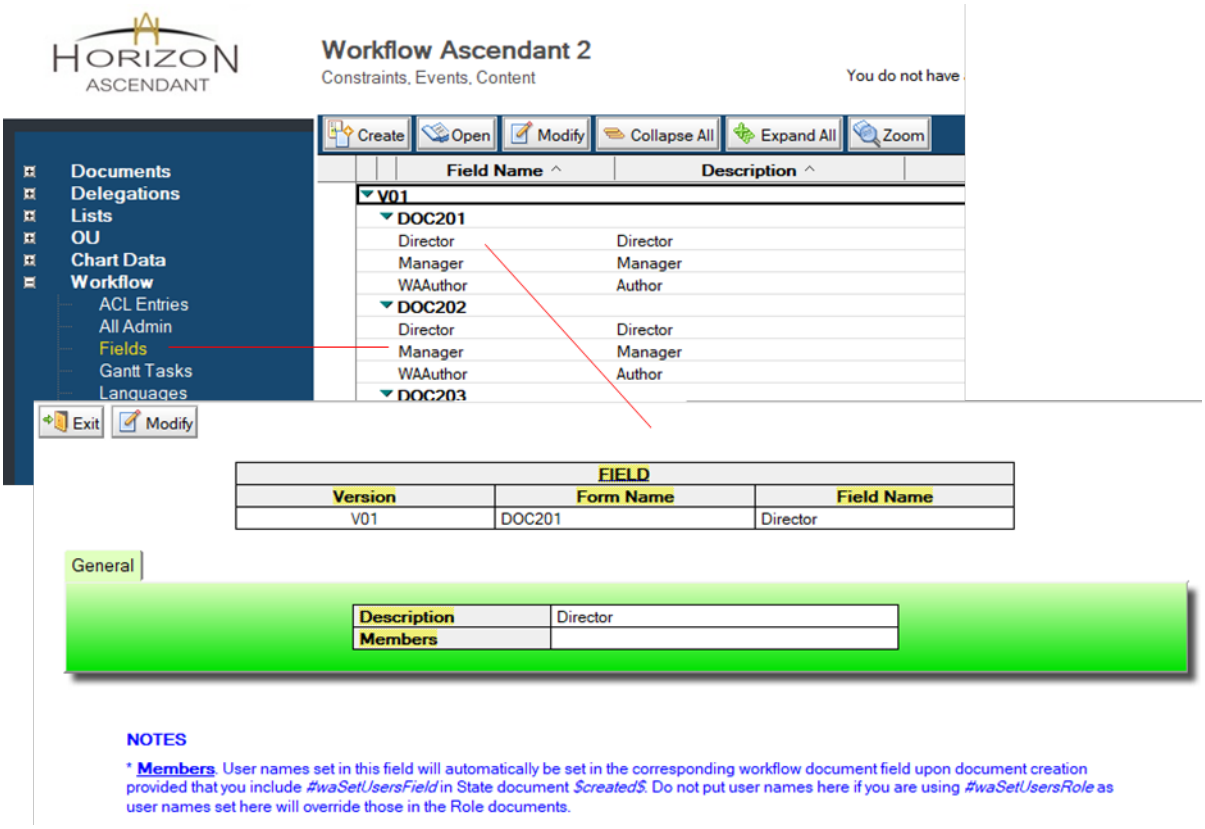
6.5) Admin ACL Entry Document

ACL Entry documents provide a centralized access to the database global access rights. Any changes you make here should be followed by selecting the *Update ACL* button. You will want to set the *Server* document to the name of your server and the *Managers* document with the names of all those required to manage the application. For application specific roles, reference Section 6.10. To note for Workflow Ascendant to properly control who can modify a document at a given point in the process, those who intervene must be set access level *Author* (the default as set below).



6.6) Admin Field Document

Each constraint used in a given workflow must contain a corresponding Field document for the delegation function to work properly. As in most Workflow Ascendant Admin documents, these are defined by *Version* and *Form Name*. For processes already in production, any changes to this list should only be done in the context of a new workflow version (e.g., *V02*).

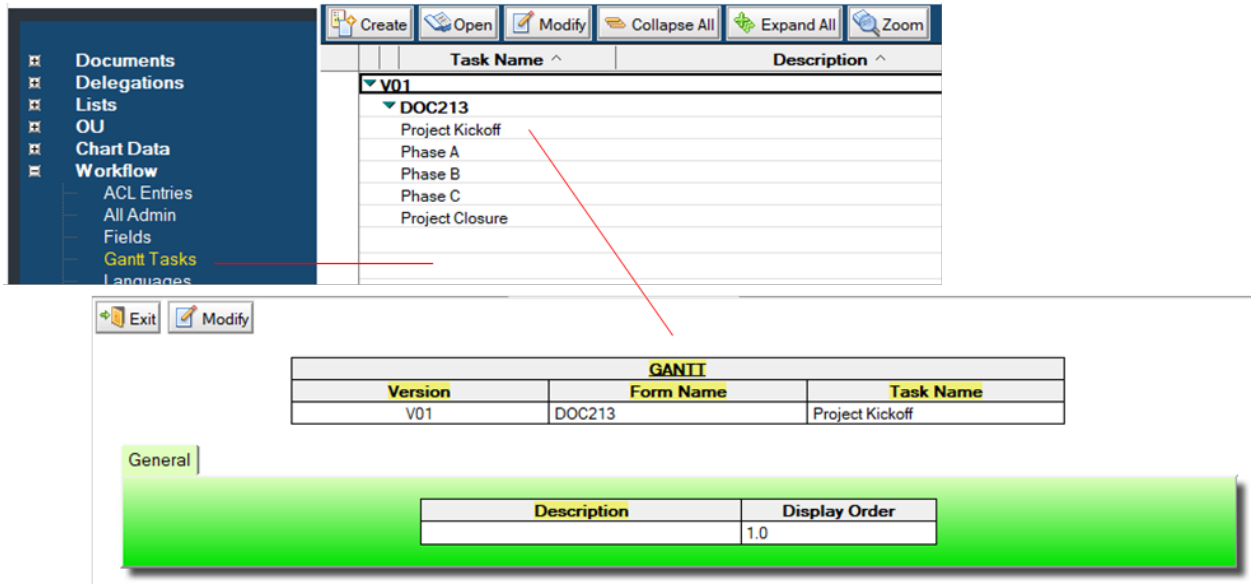


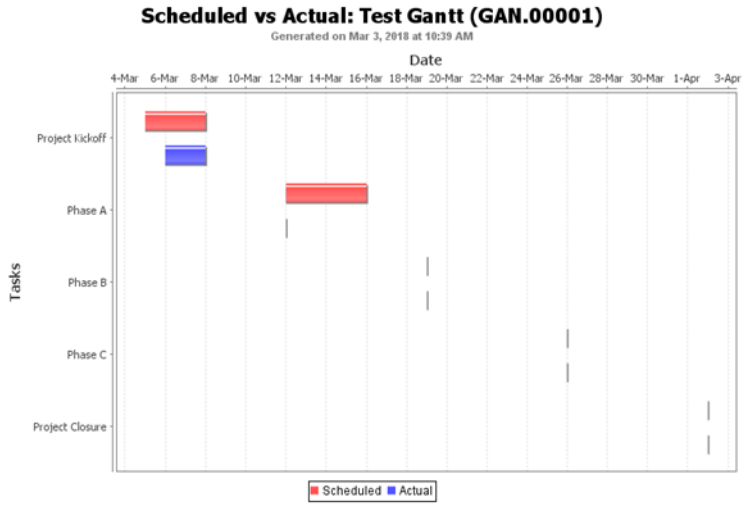
Use *Members* in conjunction with *#waSetUsersField* to initialize your constraint fields (i.e., who performs which roles in the

Changes made to Field documents while workflow docs are in production (same version, same process) may result in application failure.

6.7) Admin Gantt Task Document

Gantt Task documents form the task list to be used in creating Gantt charts. Use these in conjunction with Subform *content_DocWFGantt* and Custom Control *content_DocWFGantt*. For a chart to be rendered displayable, all default dates (*01-01-2000*) must be set to a different value (which will turn the ready indicator from yellow to green). As in most Workflow Ascendant Admin documents, these are defined by *Version* and *Form Name*. As these documents are only accessed upon workflow document creation, any changes to this list will not require enacting a new workflow version (e.g., *V02*).

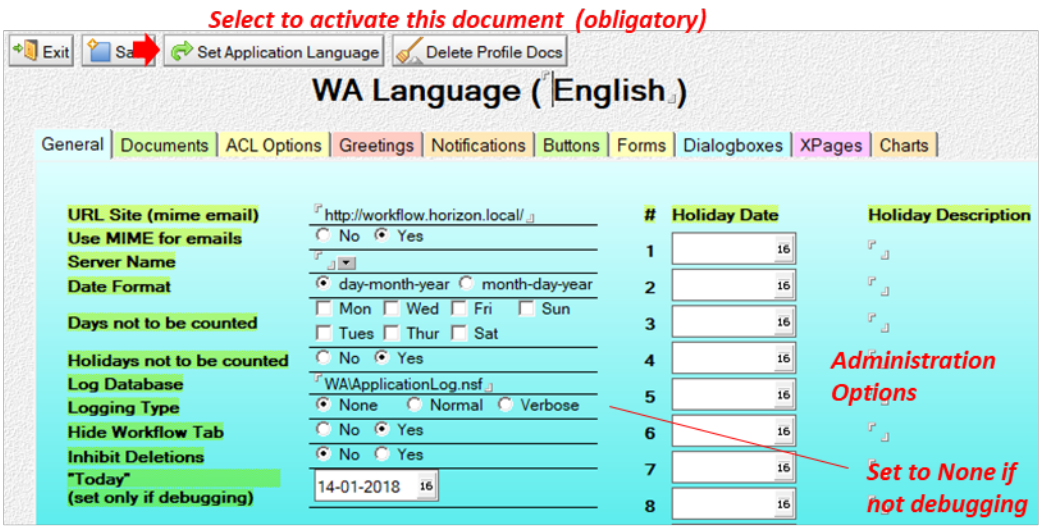


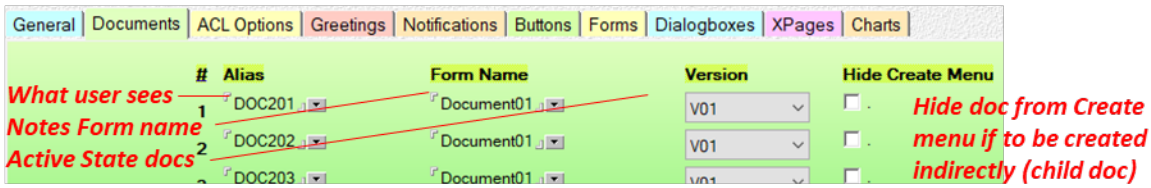


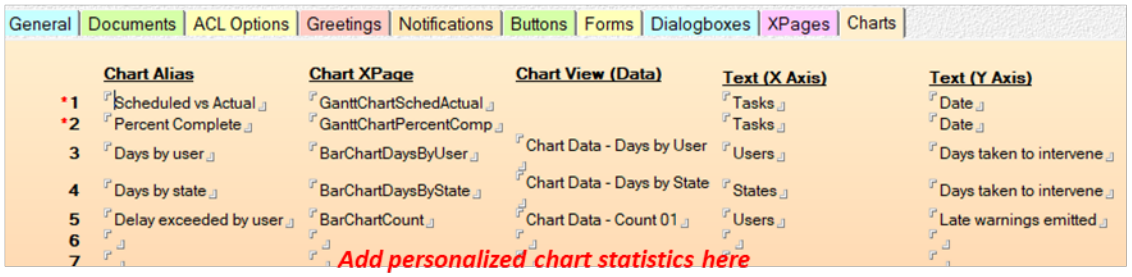
Use *Display Order* to specify the order in which tasks will be displayed in the view and in the corresponding Gantt chart. *Description* is solely included here for documentation purposes.

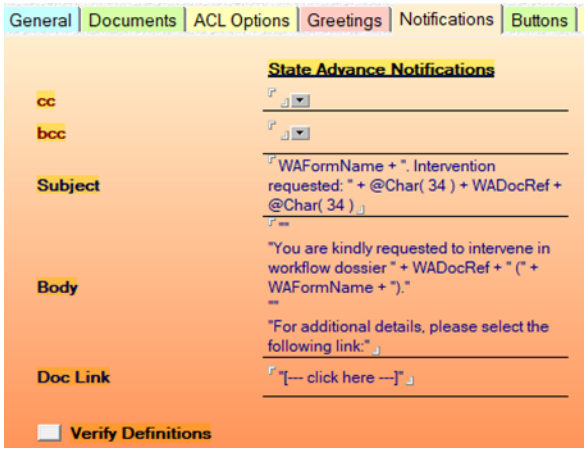
6.8) Admin Language Document

The active Language document dictates much of the behavior of Workflow Ascendant. The graphics included below illustrate options that affect basic operations. All other sections here provide you the opportunity to rename objects displayed in the application - or even create a new Language document to display all in the language of your choice.





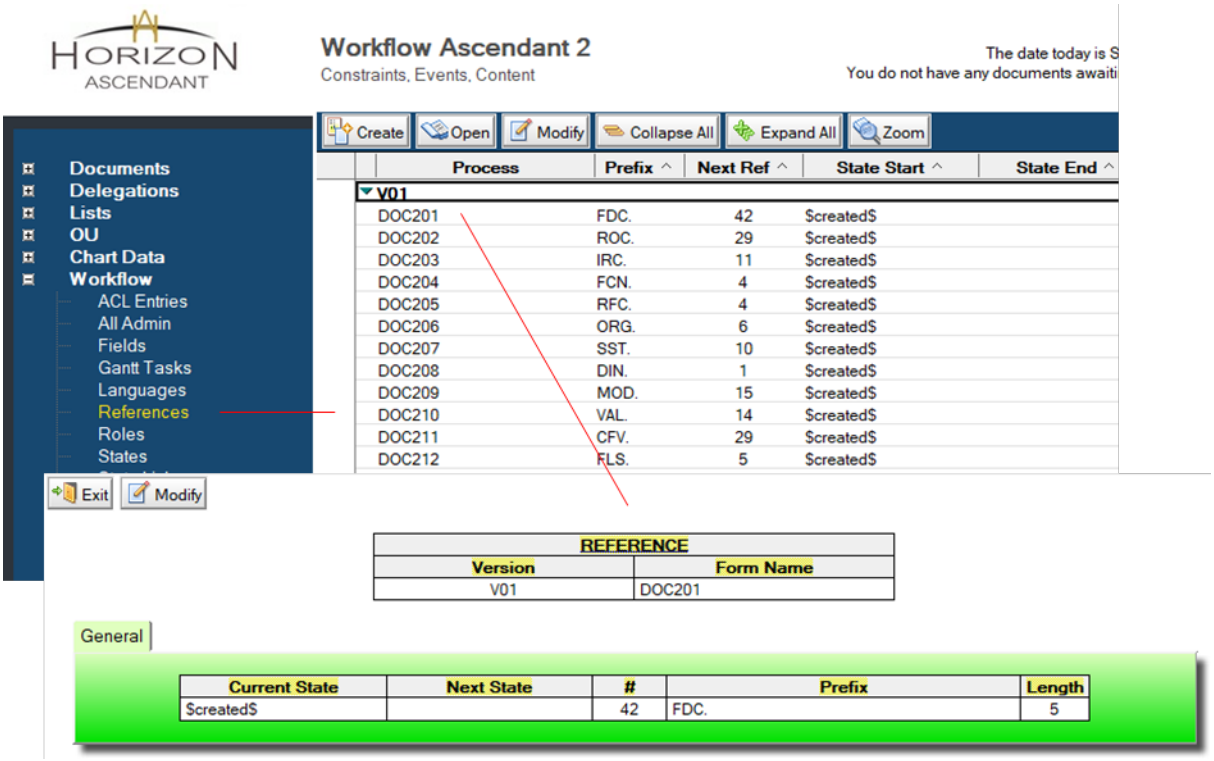


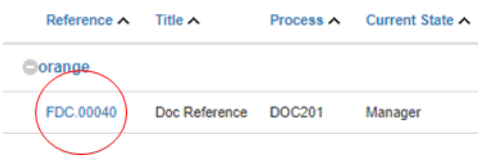


Information from the *Notifications* tab will be included in emails where you leave one or more of these fields empty in the State document. To note that text in blue is interpreted (i.e., these are executable statements). Select *Verify Definitions* to ensure that what you have specified is syntactically correct.

6.9) Admin Reference Document

Reference documents provide a means to allocate a unique and sequential reference to workflow documents. This ensures a consistent way to identify documents and verify that none are missing (note that a workflow document should *never* be deleted – only archived). References are virtually always required to be assigned to a workflow document when it is first launched into a process (as in the example below) but you can specify a different state if desired.





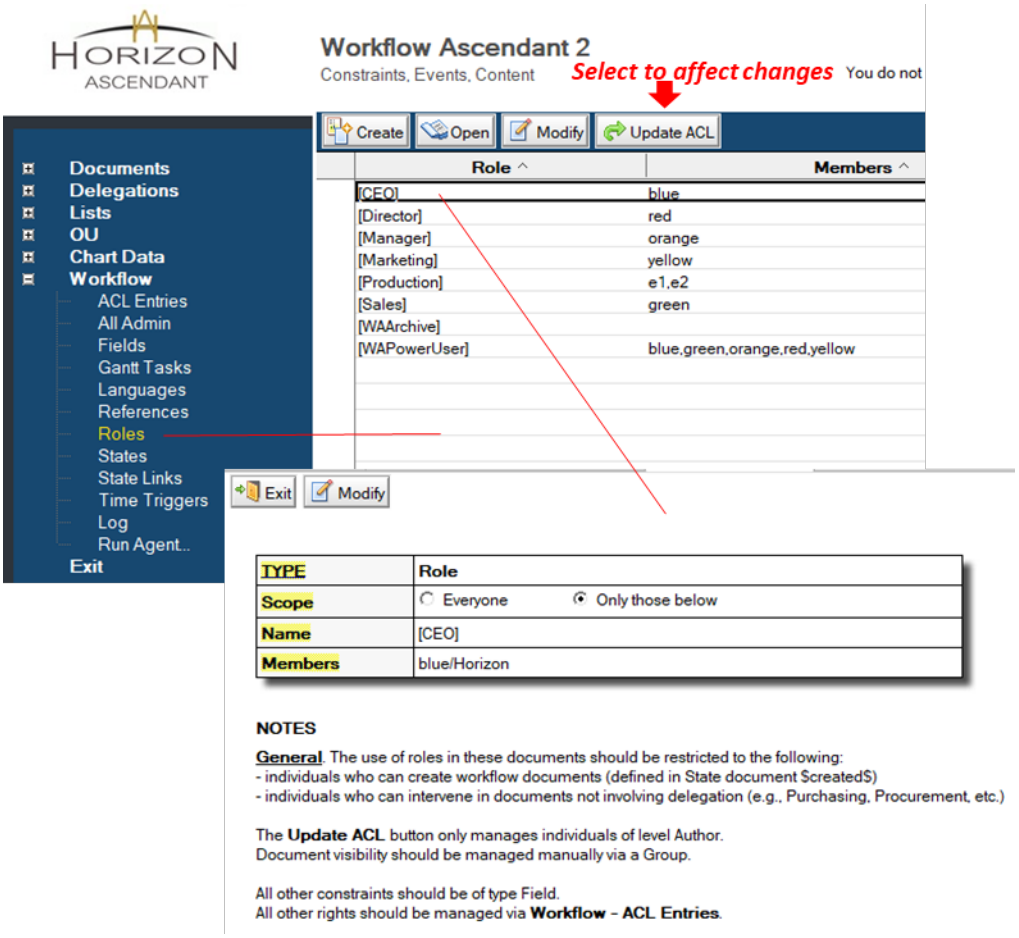
In the above example, the next *DOC201* workflow document to be created and sent into the workflow will be allocated reference *FDC. 00041*.

6.10) Admin Role Document

Role documents should generally be used in conjunction with Field documents. With the exception of the initial state, all constraints in the workflow must resolve to fields containing user names. To properly implement this Role-Field system, include *#waSetUsersRole* in initial State document *$created$* and name the following identically:

- Field Doc. Example: *CEO*
- Role Doc. Example: *[CEO]*
- Field in the workflow document: *CEO* (multivalued: *Name* type in Notes, *Text* type in XPages)

In this example then, field *CEO* will automatically be set to *blue/Horizon* when the document is created, keeping in mind however that this information is taken from the Role document – not the ACL. Updating the ACL allows for proper handling of the initial state only.



Of course you can use roles to show/ hide various portions of your

[*] indicates all users in State

6.11) Admin State Document

A State document dictates how a workflow document is to respond at that state. A collection of State documents represents the application process. In the example below, the document is initialized by routine *#waInitModFields* when *Manager* opens the workflow document. In selecting the button *Send*, there are 3 choices to choose from: *To be modified*, *Rejected* and *Approved*. If the latter is selected and field *Total* in the workflow document is set to *1000* and routine *#waCntMandFields* does not flag an error (rule *b*), then the workflow document is updated from the *Actions* tab (field *CommentsLabel01A*, …), the default email from the Language document is sent (as the *Subject* and *Body* fields are left blank) to those responsible to intervene in the next state (**N = Next*), and the document is routed to state *CEO*.

Exit | Modify | Verify

STATE			
Version	Form Name	State Name	State Type
V01	DOC104	Manager	(•) Standard ( ) Virtual

Workflow | Notifications | Actions | Documentation

GENERAL CONSTRAINTS

	The following RIs can intervene...	And forward to the following states...	Which will be displayed as...	Parallel Operation	Extended Operation	Inclusive Operation	View Display	Historical
1	Manager	To be modified	<state>	-	-	-	<user>	<state>
2	Manager	Rejected	<state>	-	-	-	<user>	<state>
a	Manager	Director	Approved	-	-	-	<user>	Approved
b	Manager	CEO	Approved	-	-	-	<user>	Approved

STOCKED PROCEDURES / AGENTS

	The document will be initialized with the following stocked subroutine(s)...	The document will be blocked from advancing if one of the subroutines signals an error...
1	#waInitModFields	-
2	#waInitModFields	-
a	#waInitModFields	#waCntMandFields
b	#waInitModFields	#waCntMandFields

1, 2, … - Manual Doc Routing
a, b, … - Automatic Doc Routing

AUTOMATIC ROUTING RULES

	The following state will be routed to automatically...	If the following condition is met...
a	Director	Total < 500
b	CEO	@True

CHILD DOCUMENT CREATION (not shown)

Notifications | Actions | Documentation

EMAIL NOTIFICATION RULES

#	To	cc	bcc	Subject*	Body*
1	*N				
2	*A	WAHistoryAuthors		WAFormName + ". Dossier declined: " + @Char(34) + WADocRef + @Char(34)	"" "For information only. The dossier " + WADocRef + " (" + WAFormName + ") has been archived without further action." "" "For additional details please select the following link:"
a	*N				
b	*N				

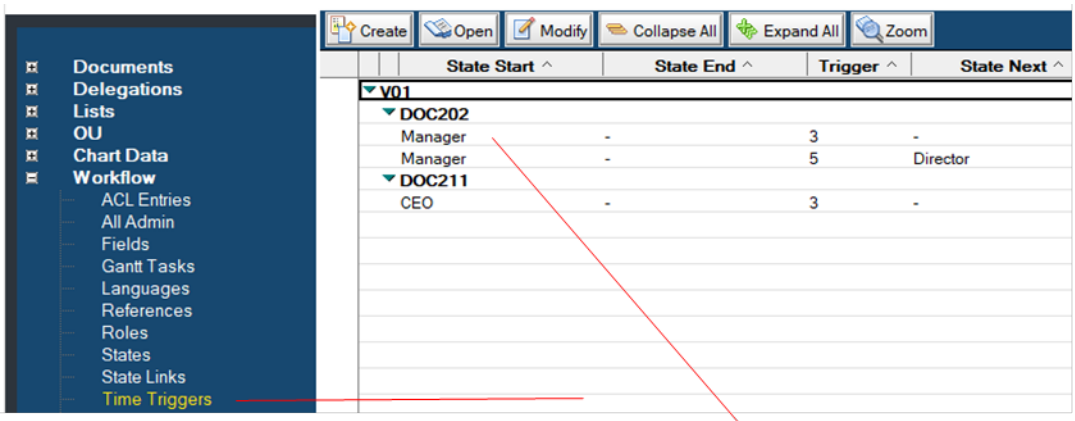
**A – Author*
**N – Next RIs to intervene*
(executable statements)

b	CommentsLabel02B	!! Director
	CommentsLabel01A	@Name([CN]; @UserName)
	CommentsLabel01B	"Manager"
	SectionComments03	CEO : "[WAManager]"
	CommentsLabel03A	""
	CommentsLabel03B	"CEO"

Actions Tab
(example: Field CommentsLabel01B := "Manager")

6.12) Admin Time Trigger Document

Time Trigger documents define actions to be taken when a document resides in a state (or in a series of states) for a designated amount of time. In the example below, if a workflow document belonging to process *DOC202* remains in any state for *3 days*, then an email is sent to those who haven't yet intervened (contained in field *WACurrentAuthors*) with *Director* on *cc*, an alert is set in the view (view document field *WADelayIcon*) and *WAChartCount01* is updated with the names of the "guilty parties" (a statistics monitor – you can create your own).



Exit | Modify

TIME TRIGGER			
Version	**Form Name**	**Document #**	**Description**
V01	DOC202	1.0	

General | Notifications | Actions | Documentation

CONTEXT

Start State	End State	State Change	Condition
Manager	-	-	@True

DETAILS

Trigger Type	Trigger Value	Agent to launch	Historical (User)	Historical
Days	3	-	n/a	n/a

Notifications | Actions | Documentation

EMAIL NOTIFICATIONS

To	cc	bcc	Subject	Body
WACurrentAuthors	Director		WAFormName + ". Intervention requested (PAST DUE): " + @Char(34) + WADocRef + @Char(34)	"" "You have exceeded the intervene in this docume therefore requested to d earliest possible conveni "" "For additional details, pl following link:"

(executable statements)

ACTIONS (ACTIVATE)

Field Names	Field Values
WADelayIcon	150
WAChartCount01	WAChartCount01 : WACurrentAuthors

ACTIONS (DEACTIVATE)

Field Names	Field Values
WADelayIcon	0

Actions Tab

Appendix A – Licensing

Workflow Ascendant is the proprietary software owned by Horizon Ascendant Inc, including any Documentation and any Support and Maintenance releases of the same Software. This software tool is used to create Business Process Management (BPM) applications. Your use of this Software and derived applications from the Software is subject to the terms and conditions of the **Workflow Ascendant Master Subscription License Agreement** (located on the Horizon Ascendant web site: *www.horizonascendant.com*). Please review the terms of this document carefully. By using the Software and/or derived applications, you agree to all of the terms contained therein.

This section is solely intended to highlight the financial conditions regarding the use of Workflow Ascendant.

Applications developed with Workflow Ascendant are licensed on a subscription basis. A separate license must be purchased for each nsf file which uses code from Workflow Ascendant: **95€ / month**.

Appendix B – Administration

TASKS

- **Modifying workflow documents data**. The Database Manager should be assigned role *[WAManager]* with access level *Manager*. Provided the application was developed following Workflow Ascendant recommendations (contained in this document), this user can modify any modifiable field in the document similar to the designated user in the workflow. Keep in mind however that the Database Manager assumes by default all roles in the process, which means in the case where there are multiple users intervening simultaneously at a given state, this can have unexpected/unintended results.
- **Modifying data fields**. The Database Manager can modify any field in documents by selecting the menu *Actions - WA Modify Field* (Notes client only). Select either specific documents or none to affect all documents in the database.
- **Changing a user name**. Change a user name for workflow documents by selecting the menu *Actions - WA Change User Name* (Notes client only). This can be particularly useful if someone leaves the company and active workflow documents need to be assigned to a different user (preferable to creating a long-term delegation).
- **Creating Delegation documents**. The Database Manager can create a delegation document for another user. Surprising how often this happens. Or not. Note that new delegations are taken into account during the early morning hours of the following day when the scheduled agent which handles this is launched.
- **Forcing delegations**. The Database Manager can force delegations to be applied immediately (rather than wait until the following day). This can be performed most easily from the *Delegations* view by selecting the button *Run Agent*. Reference the Admin View in General Web Interface.
- **Deleting workflow documents**. The Database Manager can delete workflow documents directly from the *Admin* view by selecting the document(s) to be deleted followed by the *Delete* button. It is highly recommended however ***not*** to delete any workflow documents. If it is the most recent workflow document created, consider resetting the corresponding *Reference* document in order not to have any "holes" in the document numbering.
- **Updating chart data**. Similar to *Delegation* documents, chart data (statistics) is automatically updated during the night. Should you need to have the latest, up-to-the-minute data, you can do update this data from the *Admin* view - *Chart Data* screen.
- **Forcing state changes**. The Database Manager can force the state of a workflow document by selecting it followed by the *Force Change State* view action button (Notes client only). This operation is particularly useful when a user mistakenly archives a document.
- **Logging events**. You can keep track of various events by enabling logging in the active Language document. This option should generally be turned off as during production as the log can become quite large (particularly on the *verbose* setting).

CONFIGURATION

- In the active Language document, you can configure Workflow Ascendant in a number of different ways including: changing the language, changing the date format, exclude weekends and holidays from the day counter, change how buttons and messages are displayed, how default emails are presented and so on.

Appendix C – Debugging

GENERAL

Domino Designer is equipped with a powerful debugging tool to debug your LotusScript code. Wherever possible, consider first putting your code in a Notes client button and verify the code there before executing it from an agent.

- **Your application specific code**. Application specific code to be executed when a workflow document is either opened or when the *Send* button is selected by the user should be referenced in the appropriate place in the State documents and defined in script library *WA Application Specific* (the calls to which to be included in subroutine *waExecuteStoredProcedure*). These subroutines can update document data and/or block a document from advancing in the workflow. All such code is executed via agents ***(waInitializeWorkflowEngine)*** and ***(waInitializeWorkflowEngine)***. Should you need to pass arguments to these routines, use designated fields to that effect in the workflow document.
- **Computed Form field values**. Computed formulas and default values will take effect as the *computeWithForm* option is set to *both* (referring to the *onload* and *onsave* events) in XPage *Document#* by default.
- **(waInitializeWorkflowEngine)**. This LotusScript agent is launched from the XPage *Document# beforePageLoad* event. This agent in turn calls the *PostOpenDoc* (*WA Workflow* script library) which in turn executes any routines you have referenced in the *State* documents and defined in script library *WA Application Specific* (the calls to which to be included in subroutine *waExecuteStoredProcedure*).
- **(waRunWorkflowEngine)**. This LotusScript agent is launched from the XPage *Document# postSaveDocument* event. This agent in turn calls routines you have referenced in the *State* documents and defined in script library *WA Application Specific* (the calls to which must be included in subroutine *waExecuteStoredProcedure*).

Workflow Ascendant is equipped with a complete set of logging features. Provided you have configured the log correctly (creating the *Agent Log* database and referencing it in the active Language document), you can put the following command in your application specific code in order to view variables / field values in the log:

Call a_LogWrite("Got Here")
Call a_LogWrite(doc.MyField(0))
Etc.

JAVASCRIPT

This will concern any specific transactions that take place *within* the context of a given state. Workflow Ascendant largely limits the usage of javascript to pulling values from the XPage and passing them on to LotusScript agents. Should you use that approach, the *a_LogWrite* command can first be used to verify that you're getting the values correctly into your agent. For any debugging to be done prior to (or after) that, you can use the following command to output variables to the Domino console:

print("Got Here")

There are also downloadable tools available on the web to display scoped variables.

Appendix D – @Formulas and Reserved Field Names

MOST COMMON @FORMULAS

- **+**. Separator used to concatenate strings. Keep in mind that Actions and Notification *Subjects* must be contained in a single line (unlike a Notification *Body* which can consist of several lines).
- **<, <=, =, !=, >=, >**. Comparative operators which can be used in automatic routing rules as well as Actions.
- **@Char(34)**. Translates to the " character.
- **@If(*condition*; *true*; *false*)**. Conditional statements which can be used in Actions.
- **@Name([CN]; @UserName)**. The current user name – used in Notifications and Actions.
- **@Text(@Today)**. The current date – used in Notifications and Actions.
- **@Today**. The current date – used in Actions where the date format needs to be preserved.
- **@Unique(WAReaders : @UserName)**. Used to set the *WAReaders* field to progressively provide visibility to a workflow document (be sure to include also *[WAManager]* and *[WASupervisor]* initially to provide those users visibility).
- **@UserName**. The current user name – used in Actions where the full formal name needs to be preserved.
- **And**. Logical operator which can be used in automatic routing rules.
- **Or**. Logical operator which can be used in automatic routing rules.
- **Not**. Logical operator which can be used in automatic routing rules.

FIELD NAMES

- **WAAuthor**. The user name of the individual who created the workflow document.
- **WACurrentAuthors**. The list of user names who can intervene in the document at that moment in time.
- **WACurrentAuthorsDisplay**. The list of current user names displayed to the end users in the various views.
- **WACurrentState**. The name of the workflow document's current state.
- **WADocRef**. Contains the unique reference allocated to the workflow document.
- **WAErrorMessage**. Contains any error messages which block advancing the document in the workflow.
- **WAFormName**. The process name displayed to the end users.
- **WAHistoryAuthors**. The list of users who have intervened in the workflow document to date.
- **WAReaders**. The list of users who can visualize the workflow document (in addition to those who have or who can modify the workflow document).
- **WAVersionRef**. The version of the workflow document.

Appendix E – List Classes

DECLARATION / INITIALIZATION

Dim listL As ItemList	Declares *listL* as a list (class).
Dim listL As New ItemList	Declares and initializes *listL* as a list (class).
Call List_Initialize (listL)	Initializes list *listL*.
List_SetListFromDocField	Initialization is not necessary for

BASIC FUNCTIONS

Call listL.DeleteNthItem(index)	Deletes the item at position *index* in list *listL*.
Call listL.InsertNthItem(index, newItem)	Inserts a new item at position *index* in list *listL*.
myItem = listL.PopOffItem	Pops off the last item in list *listL*.
Call listL.MoveDown(index)	Moves the *listL* entry at position *index* down in the list.
Call listL.MoveUp(index)	Moves the *listL* entry at position *index* up one in the list.
Call listL.PushOnItem(newItem)	Adds a new item to list *listL*.
Call listL.ReplaceNthItem(index, newItem)	Replaces the item at position *index* in list *listL*.

ROUTINES

Call List_CopyItems(item | listL, resultL)	Appends an *item* (or items in *listL*) to list *resultL*.
Call List_GetIndexesFromItems(item, listL, index)	Returns the position *index* of *item* in *listL* (*0* if not present).
If List_IsMember(item, listL) Then...	Returns True if *item* is found in *listL*.
Call List_OrderList(listL, resultL)	Alphabetically orders *listL* in *resultL*.
Call List_SetDocFieldFromList(doc, "Field01", listL)	Sets *doc* field *Field01* (multi-value field!) with *listL*.
Call List_SetListFromDocField(doc, "Field01", listL)	Sets *listL* with the contents of document *doc* field *Field01*.
Call List_SetStringFromList(listL, ".", result)	Converts *listL* to a string in variable *result* using separator ".".

LOOP OPERATION

```
For index = listL.GetFirstIndex To listL.GetLastIndex
        currentItem = listL.GetNthItem( index )
Next
```

STACK OPERATION

```
While Not listL.IsListEmpty
        currentItem = listL.PopOffItem
Wend
```

Appendix F – Script Library (WA Application Specific)

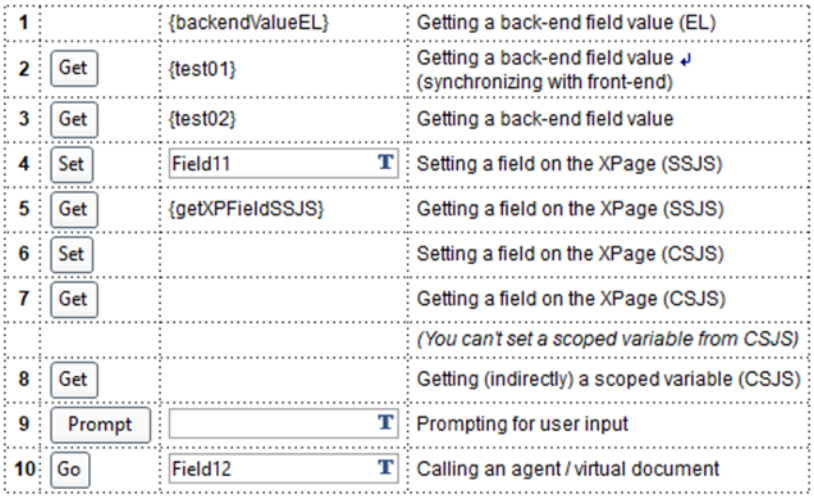
It is highly recommended that you do not modify subroutines beginning with "*wa…*". The following is a list of these subroutines (at the time of this writing) along with a description as to their purpose.

- **waCntMandFields**. Controls mandatory fields (see routine *waSetErrorFlags* for details). Used with "red circle" error indicators.
- **waExecuteStoredProcedure**. Contains a list of routines referenced in the State documents. Any new routines accessed by the State documents must be added here in order to be executed.
- **waExecuteStoredProcedureChild**. Contains a list of routines referenced in the State documents. Any new routines accessed by the State documents must be added here in order to be executed
- **waInitGanttFields**. Initializes the fields used in a Gantt chart.
- **waInitModFields**. Sets which fields in the workflow document the current user can modify.
- **waInitModTabs**. Sets which tabs in the workflow document the current user can modify.
- **waSetErrorFlags**. Returns true if mandatory fields are not set and updates the workflow document with the following: *WAErrorMessage* (contains the error message to display) and *FieldError* (contains a list of fields to display the corresponding red circle error indicator).
- **waSetFieldListFromFieldDocs**. Internal routine which sets a list of field names derived from the Field documents.
- **waSetFieldUserListFromRoleDocs**. Internal routine which sets a list of field names derived from the Role documents.
- **waSetListModifiable**. Internal routine which gets the list of modifiable fields for the current user.
- **waSetRedCirclesNotes**. Sets the red circle indicators for Notes clients.
- **waSetUsersField**. Copies all users from the Field documents to the Workflow tab of a workflow document provided that the corresponding fields in the workflow document are empty.
- **waSetUsersMngr**. Copies all managers of the current user from the OU documents to the workflow document. Generates an error if the manager of the user is undefined.
- **waSetUsersRole**. Copies all users from the Role documents to the Workflow tab of a workflow document provided that: 1) There are corresponding Field docs for those roles; and 2) The corresponding fields in the workflow document do not contain a value.

Appendix G – JavaScript

Custom Control *zcontent_DocCode* contains a sample of some commonly used JavaScript code. In addition, the following links connect to web pages containing sample code for the various JavaScript commands available for XPages:

- NotesDocument Sample JavaScript Code for XPages
- NotesView Sample JavaScript Code for XPages
- NotesDatabase Sample JavaScript Code for XPages



1		{backendValueEL}	Getting a back-end field value (EL)
2	Get	{test01}	Getting a back-end field value (synchronizing with front-end)
3	Get	{test02}	Getting a back-end field value
4	Set	Field11	Setting a field on the XPage (SSJS)
5	Get	{getXPFieldSSJS}	Getting a field on the XPage (SSJS)
6	Set		Setting a field on the XPage (CSJS)
7	Get		Getting a field on the XPage (CSJS)
			(You can't set a scoped variable from CSJS)
8	Get		Getting (indirectly) a scoped variable (CSJS)
9	Prompt		Prompting for user input
10	Go	Field12	Calling an agent / virtual document

1	
2	var doc:NotesDocument = document1.getDocument(true) viewScope.test01 = doc.getItemValueString("Field11")
3	viewScope.test02 = document1.getValue("Field11")
4	getComponent("Field11").setValue("Server Side JS")
5	viewScope.getXPFieldSSJS = getComponent("Field11").getValue()
6	var element = XSP.getElementById('#{id:Field11}') element.value = "Client Side JS"
7	var myVar = document.getElementById("#{id:Field11}").value var element = XSP.getElementById('#{id:Field11}') alert(myVar) alert(XSP.getFieldValue(element))
8	var myVar = document.getElementById("#{id:backendValueEL}").innerHTML alert(myVar)
9	var userResponse = prompt("Please enter your response:","<response>") XSP.getElementById("#{id:userResponse}").value = userResponse
10	var doc:NotesDocument = document1.getDocument(true) var docVirtual = database.createDocument() var agent:NotesAgent = database.getAgent("(sampleAgentCallFromXPages)") docVirtual.replaceItemValue("TestField", doc.getItemValue("Field12")) agent.runWithDocumentContext(docVirtual) document1.setValue("Field12", docVirtual.getItemValue("TestField"))

Appendix H – Methodology

More than 90% of the development in Workflow Ascendant is done in classic IBM Notes. While following are the basic steps, the best approach is generally to copy/paste/adapt existing generic State documents which you have already tested. The XPage layer is (optionally) added at the end of the process once you have an operational application for the Notes client. For your very first application, you will need to sign the model database, configure the basics in the Language document, set the basic *Workflow - ACL Entries* and create an agent log (optional).

1. **Notes database**. Make a Notes copy of the database according to your requirements (including modifying Page *WA Left* to reflect the name of your database).
2. **Language document**. In the *Documents* tab of the active Language document, set the *Alias* (how the process will be known to your end users) and corresponding *Notes Form Name* for each of the processes to be included in the database. *Logging Type* should be set to *Verbose* and *Hide Workflow Tab* to *No*.
3. **Field and Role documents**. Create these documents once you have identified the various profiles of those who will intervene in the various processes. Creating these first allows you to select these values from a list in the State documents (although that is strictly optional). Consider setting default values (*a1/Horizon*, *blue/Horizon* …) in these fields for subsequent testing.
4. **Notes Forms**. Modify the existing Notes Forms (one per process) from existing ones forms:
    - **Workflow Tab**. Create all the Constraint fields defined in the previous step. Consider setting default values (*a1/Horizon*, *blue/Horizon* …) in these fields for initial testing.
    - **Fields**. Create fields in the form which will affect the process (i.e., how documents are routed).
5. **State documents**. Create the State documents (always using the button *Verify* to verify that your definitions are coherent), concentrating on the basic manual and automatic routing rules:
    - **The following RIs can intervene…**
    - **And forward to the following states…**
    - **Which will be displayed as…**
    - **The following state will be routed to automatically…**
6. **Notes workflow testing**. Select the *Verify* button in the view to ensure that the process is coherent before testing. At the end of this step the basic workflow should be operational (note that the users placed as default values in Step 4 will be used, not those in the Field and Role documents).
7. **Notes initialization testing**. Include in each state document *$created$* (one for each process) the routines you plan to invoke when a workflow document is created (e.g., *#waInitModFields*, *#waInitModTabs*, …) in field *The document will be initialized with the following stocked subroutine(s)…* Remove the default values from Step 4 and verify that the expected users from Step 3 are correctly inserted into the appropriate fields on the Workflow tab (there is no need to test the workflow at this point).
8. **Finalize Notes Forms**. Create the rest of the fields in the Notes forms not completed in Step 4.
9. **Control routines**. Add any custom control routines to script library *WA Application Specific* and reference these (as well as any of those provided by default) in the various State documents.
10. **Window dressing**. Complete the State document Notifications and Actions tabs, as well as customizing presentation related information (*Historical Description* …).

OPERATIONAL NOTES CLIENT WORKFLOW APPLICATION

1. **XPages components**. Copy/paste XPages and Custom Controls to correspond to the Notes application created above.
2. **XPages fields**. Copy/paste fields into the appropriate Custom Controls.
3. **Visibility**. Apply mechanisms to selectively hide documents and/or portions of the workflow documents.
4. **Deployment**. Once you deploy applications, be sure to use design templates, naming them accordingly to facilitate design roll-back (e.g., *MD.Promo.2018.02.15.ntf*).

OPERATIONAL WEB CLIENT WORKFLOW APPLICATION