Workflow Scripting in Enterprise Architect

www.sparxsystems.com

Table of Contents

Introduction

High-end development of large scale applications requires tight control on getting work completed in a series of interactions between team members. When these interactions are defined as an ordered flow of connected steps it is referred to as **workflow**. Using workflows you can set the order of work to be performed by members of the team and ensure that specified outcomes are obtained on completion of the workflow routine.

When working in a model driven development environment there can be numerous workflow processes operating in the design and the development of a project.

Enterprise Architect offers a number of tools to facilitate a workflow process ranging from using Gantt chart views, team-review interactions and model mail through to a fully scripted workflow process.

The purpose of this document is to provide an illustration of the set-up and features of workflow scripting in Enterprise Architect, to help you work out your own solution guided by the examples provided. The intended audience is those administering the overall project management of a design and development process.

What you will need

For this introduction to workflow scripting you will need the Corporate edition or above of Enterprise Architect version 10 or later.

The example model referenced in this paper can be downloaded from:

http://community.sparxsystems.com/white-papers

The model has security enabled. To open this model use:

User Name: *admin*

Password: *password*

What you will learn

In this discussion of Enterprise Architect's workflow capabilities you will be guided through steps to enable workflow using scripting. These steps include:

- Setting up teams and team-members using Enterprise Architect's security facility
- Using Profiles for adding to an element a set of details that will be used in the workflow process
- Using workflow scripting to define user-specific views of tasks
- Using workflow scripting for defining validation rules for user input
- Using the debug facilities for debugging and testing workflow scripts

What is a workflow process?

A workflow process involves a team of users interacting to achieve a common objective. This process starts with prompting a team-member to action a task presented to them. When a team-member is prompted to action they can initiate a process involving other team-members and resources, the outcome of this process being determined by various factors and the current status in the process. After a team-member has acted upon their allotted task they then assign the task to the next recipient. This process is continued until a final outcome is reached.

Overview of workflow usage

Enterprise Architect's workflow scripting, in its simplest form, can be used for keeping track of workflow items such as issues or change requests as they are being processed, as well as providing a means of validating user-input.

Using these scripts you can provide a team-member, assigned with the task of resolving issues, with a set of items grouped by the action that needs to be taken. This grouping of items is provided in Enterprise Architect's Model Search view.

The grouping of these items can be based on administrator-defined attributes (Tagged Values). Although the Model Search view of the workflow items is key, there are numerous other resources that can be incorporated, such as the Project-Management Gantt chart view and the Testing view.

For an example of what workflow scripts can provide, open the example model supplied with this paper.

To use the example you will need to log in as different users to see the different workflow features that have been associated with the user-groups.

To quickly view different sets of tasks assigned to each user, use the group of scripts called *Logins* found in the scripting view (main menu: Tools | Scripting). Each script allows you to log-in as a different user. Figure 1 shows this group of scripts.

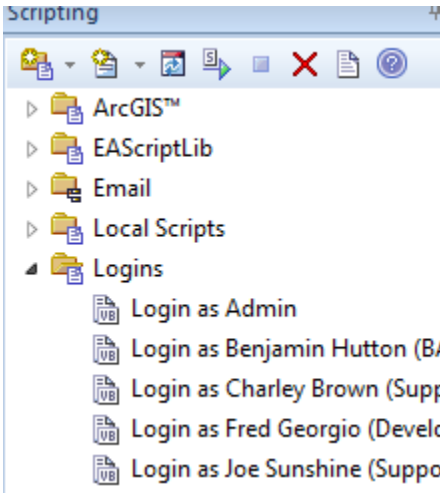


Figure 1: Login scripts in the example model

To view task related details open the Tagged Values view (Ctrl+Shift+6).

Each user can access their workflow details in one of two ways:

- By using the Personal Tasks view from the main menu: View | Personal Tasks > Workflow, as shown in Figure 2

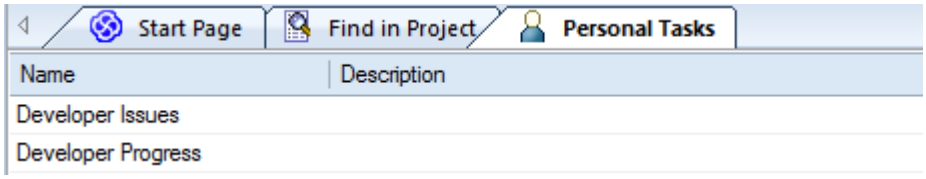


Figure 2: Selecting your Workflow listing from Personal Tasks

- By using the Model Search (Ctrl+F) and selecting their assigned workflow searches, as in Figure 3:

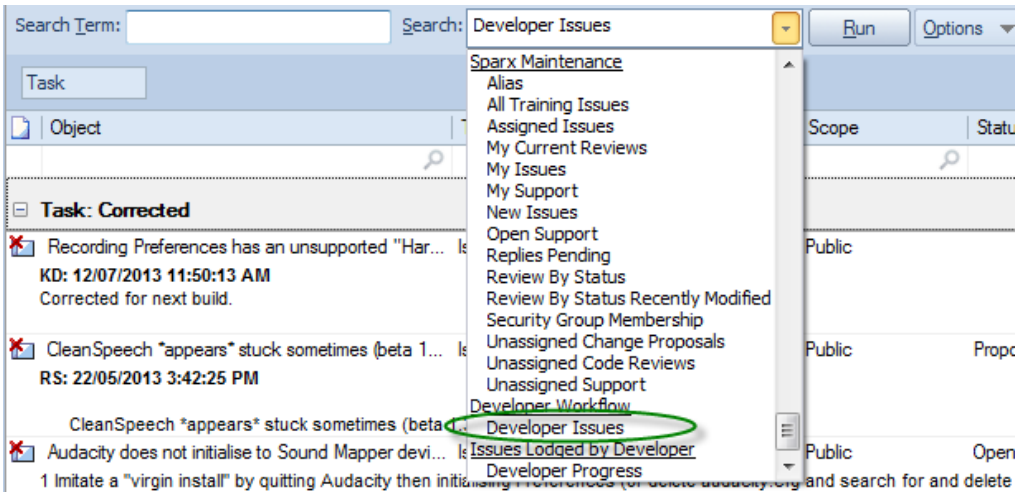


Figure 3: Selecting your Workflow listing from the Model Search

Figure 4 shows a list of tasks (Issue elements) assigned to the currently logged on user. The Tagged Values view displays the administrator-defined details specific to each issue (more details on how to set these Tagged Values are provided below).

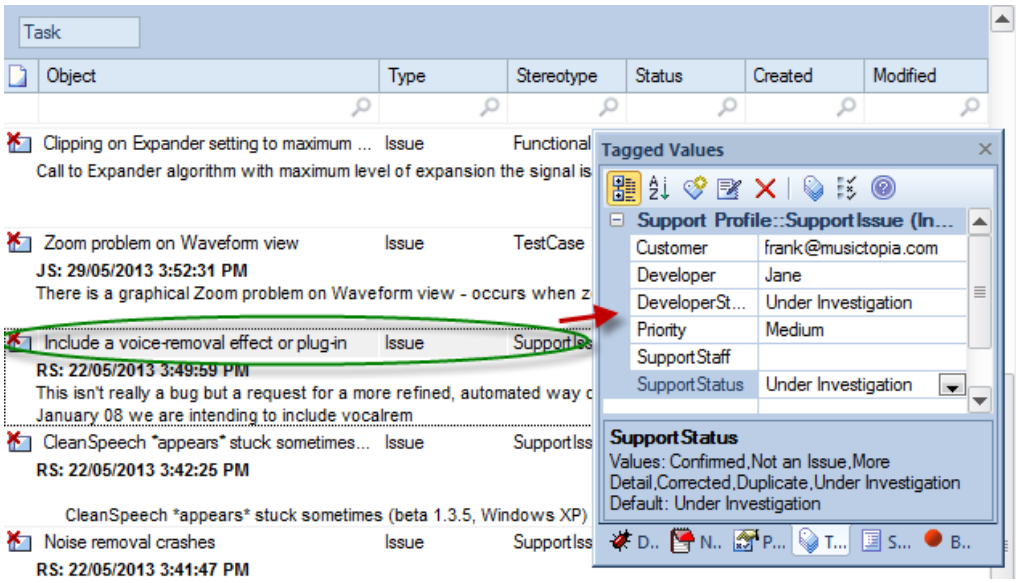


Figure 4: Viewing the details of an Issue in the Tagged Values view

The workflow scripting allows you to define how these issues are viewed by each team-member or team. The list of entries and their grouping can be based on things such as an issue element's status or information in one of its Tagged Values.

Scripting also supports defining validation rules and what ability the user has to alter information in Tagged Values.

Using Notes for tracking details of work carried out

The element.notes field can be helpful in logging team-members' actions. It can also be used for communicating a required action to the next recipient.

> ***Note:*** A recommended practice for each user's action is for each team-member to log their initials, the date, and some brief text in the Notes of the Issue elements. This also helps in tracking and reviewing the history of any processes performed.

On selection of an Issue, the notes text can be edited by double-clicking the element and opening the **Properties** dialog (or use the **Notes** view (Ctrl+Shift + 1)).

> **Tip:** After adding your initials, you can auto-generate a date by pressing the F5 key.

Figure 5 is an example of notes lodged against an issue being processed.

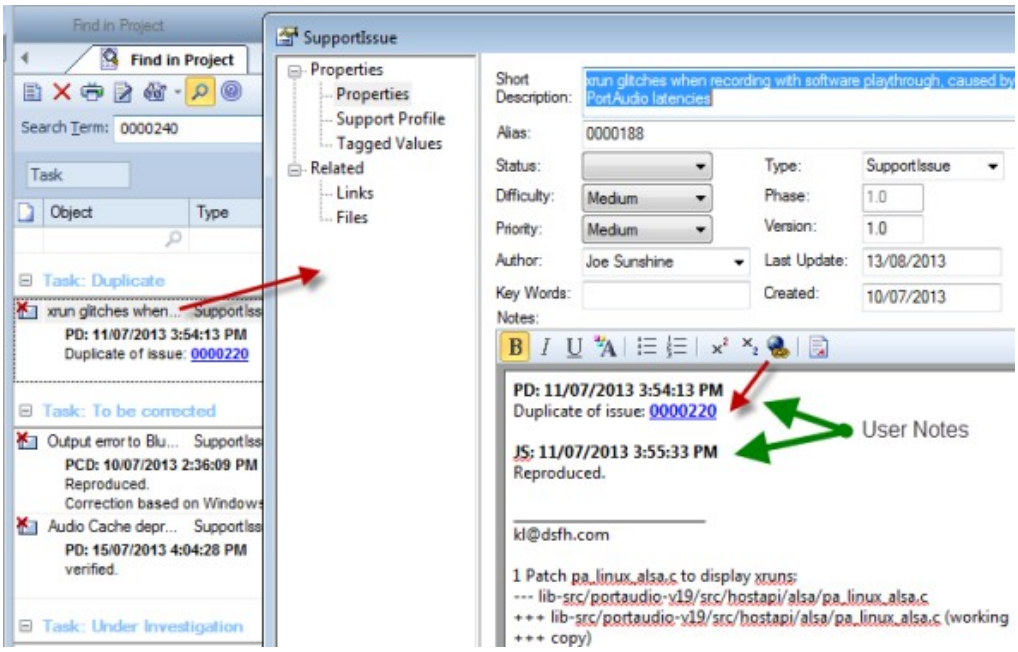


Figure 5: Viewing the process flow via notes added to an Issue

Figure 5 also includes a cross-reference to other issues inserted as a hyperlink to the related Issue element.

Using workflow scripting you can generate a listing of elements filtered by element features such as Project Management tasks. Figure 6 shows an example of a task allocated to a user using the Project Management Resource Allocation (Ctrl+Shift+7).

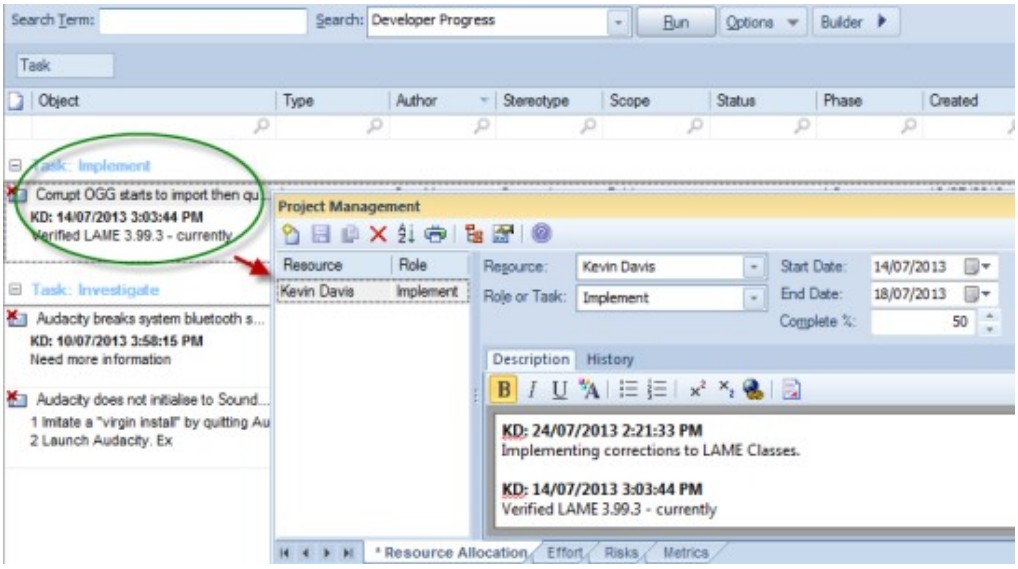


Figure 6: Example of a Workflow Search based on Resource Allocation

This can be further managed through the Project Gantt view as shown in Figure 7.

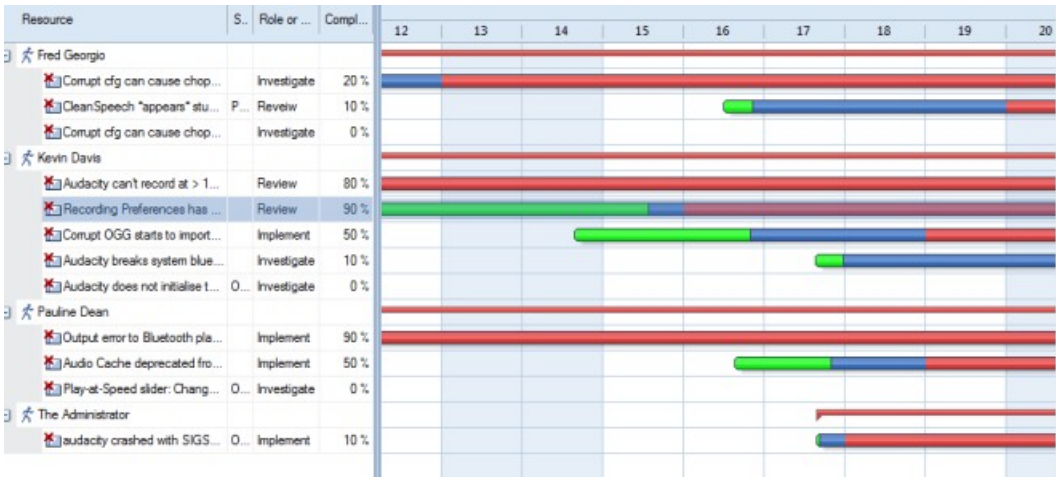


Figure 7: Gantt view of the Allocated Resources

Establish what you want to process

When defining a workflow process there are many notations that can be used. A common method is to use a BPMN business flow diagram to establish the path of the process. An analysis of the process involves identifying:

- Which users are involved
- What actions need to be recorded
- What the expected outcomes are

As an example of setting up a workflow process in your modeling environment we will discuss a common scenario; that of logging and processing an issue submitted by an end-user of a software product.

On analysis of this process we can identify that the key players involved form two groups:

- Support staff
- Developers

Figure 8 is a BPMN model for processing issues received by support:

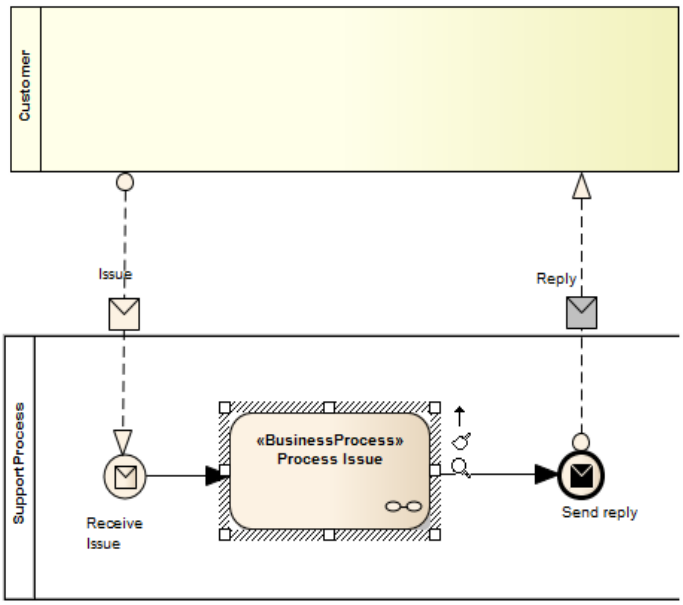


Figure 8: The main BPMN diagram of the Support Issue Processing

Figure 9 shows the detail for internal processing of an issue involving the two groups of users (support-staff and developers). The support staff receive the issues, process them and submit bugs or unresolved issues to developers.

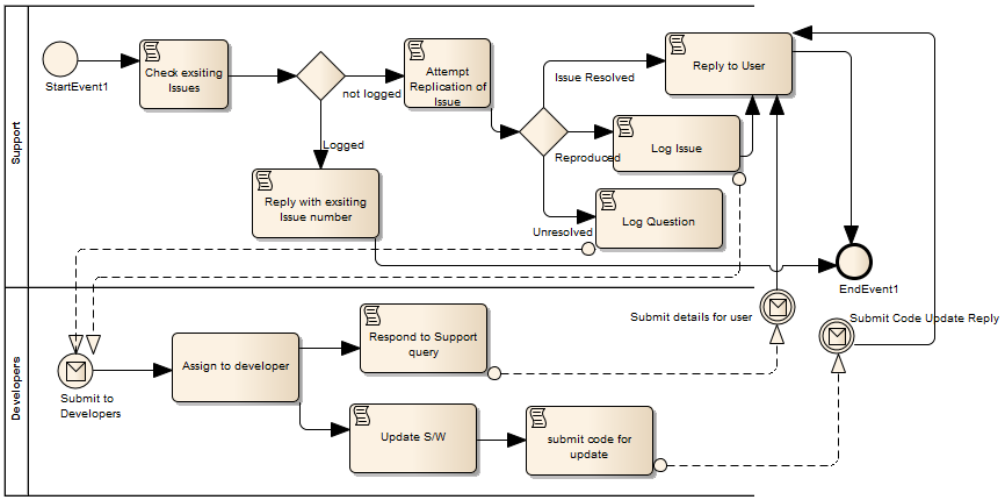


Figure 9: BPMN model for processing Issues

For access to these diagrams see the following package in the model associated with this paper:

Model.Support Issues Workflow.Support Issues Analysis

Rules for Workflow

With a workflow item you can impose a set rules on a team-member's actions. Once the teams and the general process have been defined, there are some team-specific requirements that also need to be stated. Using a workflow script you can create a set of validation rules to be imposed on a team-members actions, based on these requirements.

The following is a simple set of rules, parallel with the analysis, that will form the basis of the workflow validation process:

1. Support-staff and developers can create Issues
2. Developers can create Change elements – support-staff cannot
3. Support-staff cannot update an Issue's developer status
4. Support cannot assign an Issue to a developer
5. Setting the support-status to *complete* requires that the developer status is already set to *complete*

These rules are defined in the example model as a set of Requirements. The implementation of these rules in the workflow script is covered in more detail in the later section Workflow Scripting.

Given the above modeling we can now start to set up the various Enterprise Architect features needed to implement this.

System setup

In terms of how this workflow process will be implemented in Enterprise Architect, the key processes are:

- Storing issue details
- Assigning an issue
- Processing an issue

For each issue posted a new element is created and stored within the repository. This issue must include user-defined fields for posting details to be recorded in the workflow process. These user-defined fields can be set up using a Profile for a specific element-type. This will be covered in the later section, Setting up a Profile.

In order for issues to be passed between team members, each team-member must be logged-in under a unique ID, using Enterprise Architect's security. For more details on this see the next section, Defining Workflow Users.

Having set up the Profile and Security you can then start to implement the workflow scripts for processing issues, based on the fields defined in the Profile and the ability to identify which user is currently processing the issue. For details on this see the later section, Workflow Scripting.

Defining Workflow users

From the BPMN analysis in figure 9 you can see that this business process includes two groups:

- Support
- Developers

Using this analysis we can now define in Enterprise Architect's Security:

- The user-groups (teams)
- A set of users associated with each of these groups (team-members)

Once the users and their groupings are defined you can use code within the workflow script to identify the currently logged-in user and the group they belong to. How to do this using script is discussed in the Workflow Scripting section.

Figure 10 shows the security definition of a user and the group that they are assigned to.

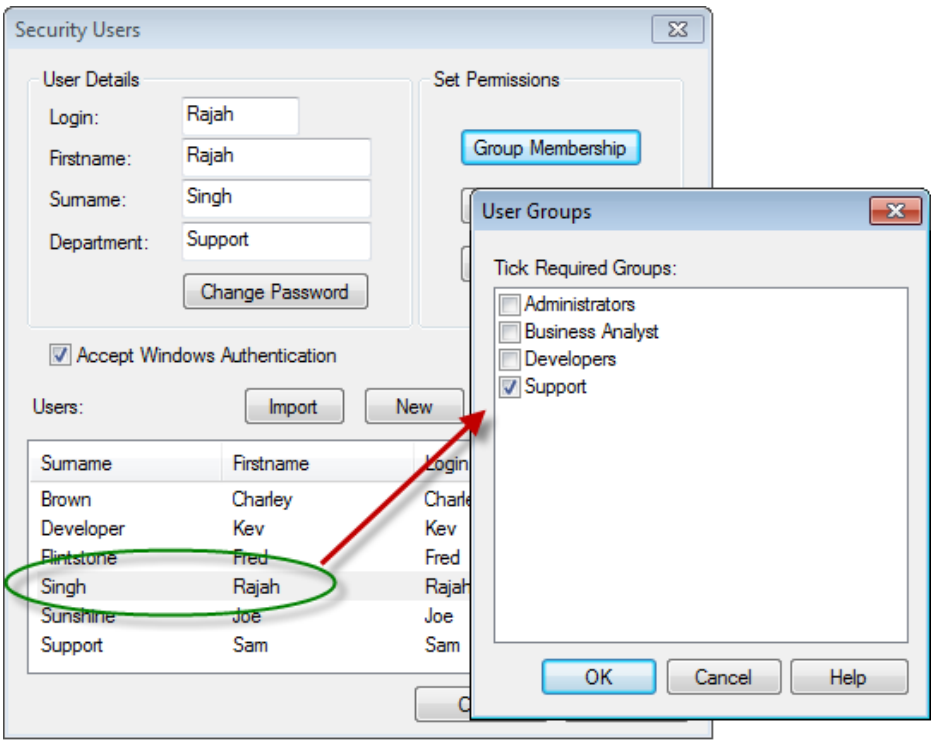


Figure 10: Setting Users in Security groups

– For more general information on configuring user-groups in security see the User Security Help topics.

Note: With Windows Authentication enabled each user's login can be based on the Windows login. For more details on using Windows Authentication see the Import User IDs From Active Directory Help topic.

To view the example model security:

- Log in as:

  User: a*dmin*
  Password: *password*

- Select from the main menu: Project | Security | Manage Users to view the details on users that have security access.

Defining resources

Based on our BPMN model, on receiving an issue you want to assign it to a staff resource. This selection of a staff member can be made from a Tagged Value. Using a Profile, these Tagged Values can provide a drop-down selection of people defined as either authors or resources in the repository. The reason the system authors and resources are used, rather than users listed under security, is that the set of users defined in security can be overly complex (with the different groups and predefined users like 'admin').

These staff resources can be defined by selecting from the main menu: Settings | Project Types | People. For more details see the People Help Topic.

Set up a Profile

Having set up the team members and their grouping in security, we then clarify what details need to be tracked in this process. For our model the details that need to be logged with an issue include:

- Customer email address
- Support person processing the issue
- Developer assigned to the issue
- Issue status in the support area
- Status in developer area
- Notes on each user's process

Using these details we can define an element type with Tagged Values that convey, in the simplest form, the issue's details at each stage of its processing.

To define a new element type we use a Profile, created using the Profile Helpers. For more detail on using these see the Using Profile Helpers Help topic.

Figure 11 shows a simple Profile for creating an element type that includes a group of Tagged Values that cover the support issue details outlined above.

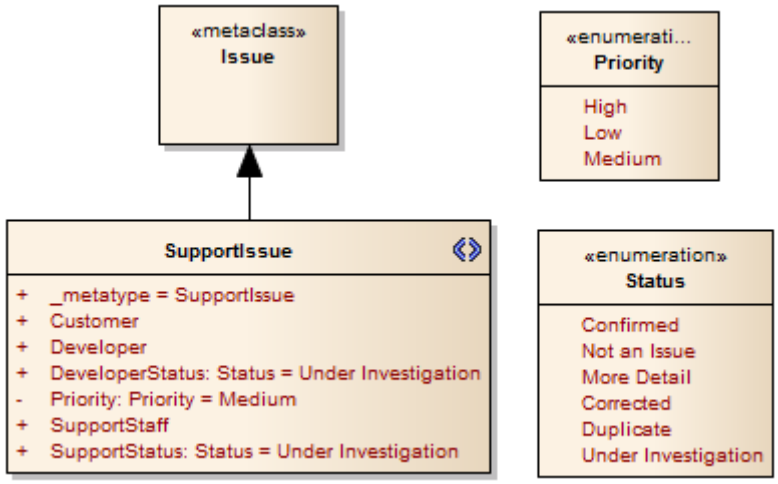


Figure 11: The definition Profile for SupportIssue

Note: This profile is available in the example model. It is located under the package:

Model.Profile Definition.Support Profile.Support Profile.Support Profile

This profile includes two Enumeration elements (**Priority** and **Status**). These are referred to within the Element.Attributes of **SupportIssue**. The Attributes referring to these enumerations support drop-downs in the final **SupportIssue** element. These are rendered as Tagged Values, as shown in Figure 12.

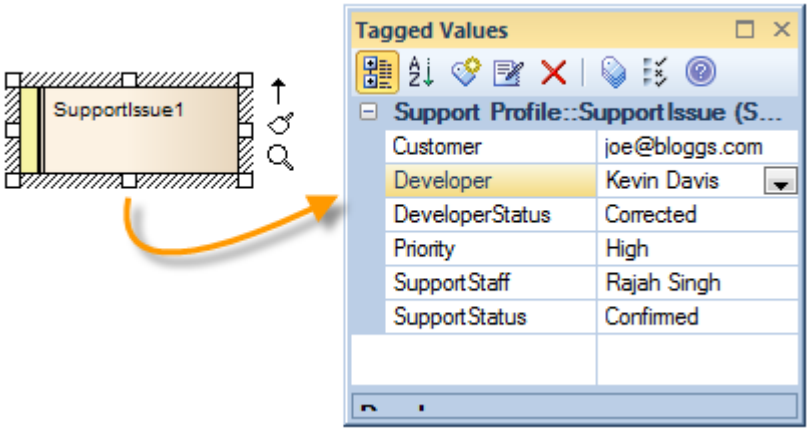


Figure 12: The Tagged Values of a SupportIssue

In the example .eap file supplied, for simplicity of use this Profile has been saved as a Resource Profile under: Resources | UML Profiles.

> **Note:** The use of Resources for posting Profiles is a redundant option and has limited features. Although it does support this simple example, it is not a recommended practice. In order to implement this Profile as a proper MDG technology, available to other models, you will need to create an MDG technology - see the Create MDG Technology File Help topic.

As this Profile has been installed in the example repository you can create elements of this type by:

- Opening the diagram toolbox – main menu: Diagram | Diagram Toolbox (Alt+5)
- Selecting **More Tools**
- Selecting **Support Profile** from the list of toolbox options

Using this toolbox you can add these elements to a package or a diagram.

As this element-type is used more in element listings rather than in diagrams it has not been associated with a specific diagram in the Profile.

Unique identifier

In addition to defining the Tagged Values we also want the element to be assigned a unique identifier. In the example model supplied, the *Issue* element type is set to automatically increment the *Alias* for each new element created – giving it a unique Issue number (for more details see the Auto Naming Help page).

Summary

In the above section we have covered the core prerequisites for creating a workflow based modeling environment. These requirements form a foundation for developing the workflow scripting. In summary the underlying features we have set above include:

- Defining a Profile
- Defining team-members using Security and model-users (Authors and Resources)
- Setting a unique identifier

Having completed the set up you can now start to work with the workflow scripting.

Workflow Scripting

To illustrate the set up of Enterprise Architect's workflow scripting we will use the workflow scripts supplied in the example model. Enterprise Architect's workflow script is a specific set of functions callable from within VBScript. For an overview see the Workflow Scripts Help topic.

A workflow script must be defined in Enterprise Architect's Scripts under a Script-group of type **Workflow** as shown in Figure 13.

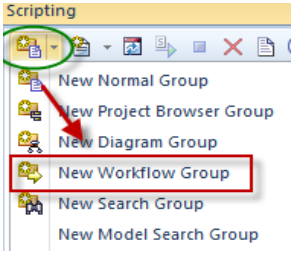


Figure 13: Creating a Workflow Script Ggroup.

The Workflow example model includes a number of scripts. In this section we will discuss only the scripts related to workflow. For details on the script for importing issues into the model from emails, see the Appendix: Importing Support Emails.

In the simplest form workflow scripts provide you with a set of options, based on the logged-in user's security identity. These options provide a means to:

- Validate user input
- Set accessibility to read and write to element fields and Tagged Values
- Create user-specific listings of elements

What is critical to using workflow scripting is the ability to debug any script defining these features.

Script development

Given that workflow scripting is based on an administrative process, the scripts do not directly support debugging by the user. What is provided in the supplied example model are scripts for running the workflow in a debug mode. These scripts are defined as standard VBscripts (not workflow scripts).

These workflow debug scripts create Classes that emulate the structure used in the workflow script and then call the workflow script itself. Although the script does not directly generate the workflow searches, it allows the developer to check for syntax errors and to view, in text form, the details of the search definitions to be created.

These generated details of the search definitions are written to the System Output view.

Using the debug scripts to call the workflow script, you can use standard debugging and the text description of the search definitions to verify your workflow scripts and check their operability.

To run the scripts you need to open the Scripting view by selecting Tools | Scripting from the main menu. The Scripting view is shown in figure 14.

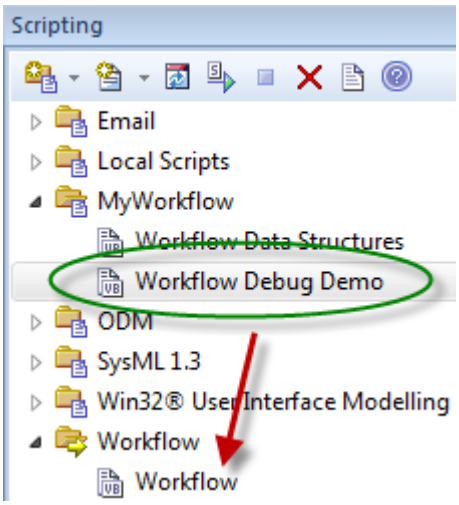


Figure 14: Scripting view showing the workflow debug script

- *Workflow Debug Demo* — The main routine to start the debugging, calling *Workflow Data Structures* to create the Classes and the *Workflow* script for debugging
- *Workflow Data Structures* — Defines the Classes to be used
- *Workflow* — The system script to run the workflow

To run the script you need to open at least:

- The **Debug** view (main menu: Analyzer | Debug)
- The **Locals** view (main menu: Analyzer | Locals)
- The **System Output** view (main menu: View | System Output or Ctrl+Shift+8)

Open the *Workflow Debug Demo* script and set a breakpoint close to the start of the workflow script.

Start your debugging by selecting the **Run** button in the **Debug** view shown in Figure 15.

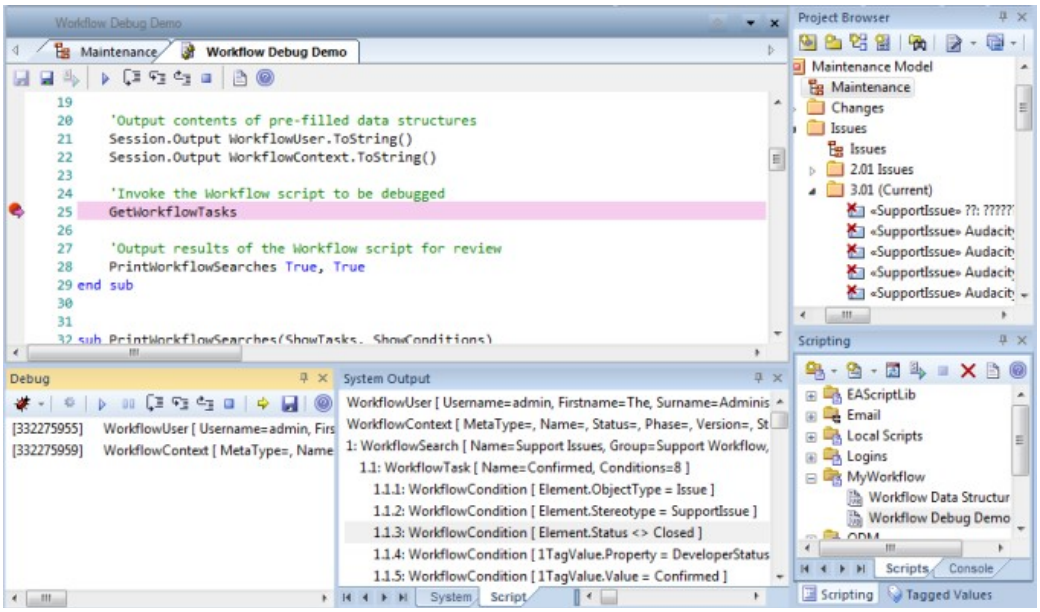


Figure 15: The Workflow Debug Demo script with a breakpoint set

On reaching your breakpoint you can control execution of the script using the debug options: Step Over / Step Out:  .

In the **System Output** view you can view the text representation of the Model Search condition statements created by your script.

You can now make alterations to the main workflow script: *Workflow* | *Workflow* and set breakpoints within this script.

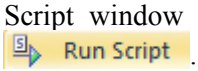
You can then check the alterations are syntactically correct by re-running the *Workflow Debug Demo* script in debug mode and checking your variables in the **Locals** view when breakpoints are reached.

Real-time run vs debug run

After stepping through the script in debug mode, to eliminate syntax errors, you can do an automated run of the *Workflow* script to check that the generated searches return the correct details.

Note that an automated run will not support debugging breakpoints, nor will it return code-errors. So correct any syntax errors prior to doing a direct *Workflow* run.

An automated run of the Workflow occurs on opening a repository or logging in as a different user. It can also be activated by clicking on the **Refresh Script Tree** icon.

To check the different sections of your script you will need to log on as users from both the support and developer user-groups. In the model supplied with this paper there is a set of scripts for logging-in under different user names, providing you with a simple means of changing between users. Figure 16 shows the *Logins* group of scripts in the Script window (Tools | Scripting). To log-in, right-click on a script and click on 

.

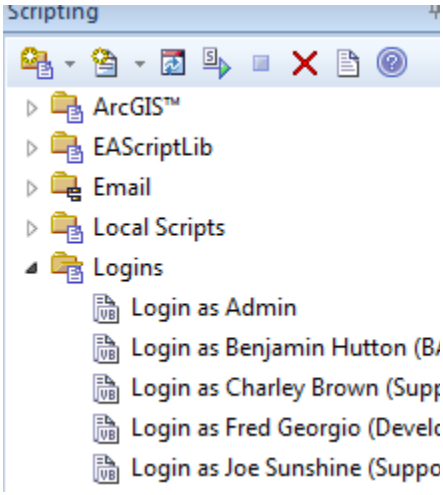


Figure 16: Login scripts in the example model

For each user, you can check:

- The searches available in the Model Search view – see Figure 3
- The details displayed in the Tagged Values view – see Figure 4
- (For Developers) the Task allocation in the Resource View – see Figure 6

> **Tip:** To avoid swapping log-ins between admin (for altering code), and a user (to check workflow actions for that user), you can open two instances of Enterprise Architect:
>
> Instance 1: log in as *Admin*
>
> Instance 2: log in as a user e.g. a Support user.
>
> This way you can update the *Workflow* script in instance 1, then in instance 2 refresh the *Workflow* script using the refresh Icon:  . Then re-run the Search to get the update from the code change.

Workflow options

With workflow scripting the system functions can be grouped as:

- Functions for creating Model Searches
- Functions for validation and control of user input

Although code using these functions can be combined under a single script, for clarity they are best grouped separately. In the model supplied they are named:

- Workflow (Model Searches)
- Validation (User input control)

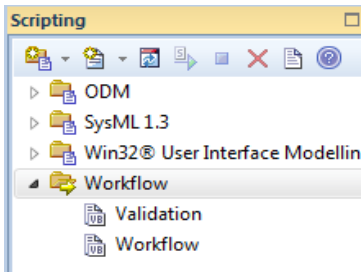


Figure 17: Workflow scripts supplied with the example model

In the following sections we will discuss some points on coding each of these two groups.

Creating Model Searches

When creating a set of workflow-based Model Searches the initial function that Enterprise Architect will call in the script is:

GetWorkflowTasks()

For more details see the Functions - Create a Search with User Task Help topic.

To view examples of code used for creating the Model Searches, see the script Workflow | Workflow, located in the Scripting window, of the example model (main menu Tools | Scripts).

In the example script the **GetWorkflowTasks()** function calls different routines for creating searches based on the security grouping of the currently logged-in user. For more information on working with the logged-in user details see the description of the **WorkflowUser** structure and **IsMemberOf()** function in the Filled Workflow Data Structures Help topic.

The routines called from **GetWorkflowTasks()** use the **WorkflowContext** object for creating Model Searches. The general structure of this can be seen in the script as:

```
WorkflowSearches(...).Tasks(...).Conditions(...).column
```

The Conditions parameter consists of Column, Operator and Value.

The column can be set for an Element field in the format: Element.field. For example:

"Element.Name"

For Element.Feature.Field (i.e. Element.TaggedValue.Name) the form is:

<ID><Feature>.<Field>

Where:

<ID>	is a numeric prefix used to group conditions
<Feature>	is the Element Feature such as "TaggedValue"; see the Element Package Help Topic
<Field>	is Feature.Field e.g. Resource.Role

For example:

"1TaggedValue.Property"

Where there are multiple conditions to be set against one Feature.field they are grouped using a numerical prefix (similar to an Alias in an SQL statement). For example:

```
NewCondition("1TagValue.Property", "=", "DeveloperStatus")
NewCondition("1TagValue.Value", "=", "Confirmed")
```

These two conditions apply to one Tagged Value. A similar scenario is used for other Element features such as Resources and Test Cases.

Validation – applying rules

In the introduction we covered a set of workflow rules to be imposed on actions taken by the support-staff and developers. For details on the functions that Enterprise Architect calls to validate and control user input into the model see the Functions - Validate and Control User Input Help topic.

The implementation of these rules in the form of workflow script is covered in the example script called Workflow | Validation. The following indicates where within the code these rules are implemented:

1. Support staff and developers can create Issues

   No script required for this rule.

2. Developers can create Change elements – support staff cannot

   See: `OnPreNewElement()`

3. Support staff cannot update an item's developer Status.

   (DeveloperStatus Tagged Value is set to read-only for Support users).

   `See: CanEditTag()`

4. Support staff cannot assign an issue to a developer.

   Allow modification status on the 'Developer' Tagged Value only for Only developer or Admin.

   `See: CanEditTag()`

5. Setting support status to completion requires developer status set to complete.

   See: `AllowTagUpdate()`

Summary

In the above section we have covered the key points for creating and debugging a workflow script. In summary, the points covered include:

- Debugging workflow scripting

- Scripting workflow Model Searches for a specific user-groups
- Defining a set of validation rules in the workflow script

Conclusion

This paper has discussed a common scenario that uses Enterprise Architect's workflow scripting, and worked through the key features that are required to implement this workflow process. This involved setting up a Profile and implementing Security, along with writing and debugging some workflow script.

Development of medium to large scale applications typically requires carefully managed work performed by interacting teams. Using the analysis of a common Workflow scenario, in this paper and with access to an example model, you have seen how to set the order of work to be performed by members of a team and how to ensure that specified outcomes are obtained on completion of a workflow routine using workflow scripting.

Appendix

Importing support emails

Given support issues are commonly logged via emails, there is a script supplied with the example model for importing email from an external email application (in this case it is specifically for importing emails from Microsoft Outlook 2003). This provides the basic mechanism for logging issues into the model directly from the original email source.

Using the script each email body is imported as text to a model element of stereotype *SupportIssue* (defined above). Any items attached to the email (including images), are attached to the issue element using the Element.File feature.

The main text of the email is imported into the Element.Notes field, along with references to any attached items (these are set as hyperlinks to file locations based on the Issue number stored in the Alias).

This script for importing email is left open to user alteration and can be debugged using the standard script debugging processes. It is accessible in the Script view under: Email | Get Outlook Email.

To ensure this script is operational you need to either create a file directory C:\temp or alter the `FileAttachmentPath constant in` the script to your own designated file folder. See the following code line in the Get Outlook Email script:

```
const FileAttachmentPath = "C:\temp\"
```

Note: given that the files referred to by the example issues are not supplied with the download, any existing links from the elements to these files will not operate.

Running the script:

To create an Issue element by importing an email from MS Outlook:

- Create a directory C:\temp (alternatively you can modify the script to point to your own file location)
- Open the email application MS Outlook
- Select an email in MS Outlook
- Select a package in Enterprise Architect's Project Browser
- Right-click on the package and select Scripts | Get Outlook Email
- MS Outlook will pause with a user prompt
- Bring Outlook into view and allow Enterprise Architect to access the email by selecting Yes in the message-box

After allowing the external access in Outlook, Enterprise Architect creates an Issue element under the currently selected package.

This will be displayed as a Model Search item. Double-click on this entry to open the Element Properties view.

In the Workflow model supplied there is a set of imported emails under the folder:

Model | Maintenance Model | Issues.3.01 (Current)