Version

**IDENTIFICATION SECTION**

|  |  |  |  |
| --- | --- | --- | --- |
| **Document Reference** | | | |
| **Client Name:** |  | | |
| **Project Name:** |  | | |
| **Client Contact:** |  | | |
| **Assignment Partners:** |  | | |
| **Project Manager:** |  | **Quality Assessor:** |  |
| **Author:** |  |  |  |
| **Project File Ref:** |  | **Version:** |  |
| **Status:** |  | **Date:** |  |
| **Filename:** |  | | |
| **Template:** |  | | |

|  |
| --- |
| **Approvals** |
| Approval ()  *(signature)* (*date*) \_\_\_\_\_\_\_\_\_\_  Approval ()  *(signature)* (*date*) \_\_\_\_\_\_\_\_\_\_ |

|  |  |  |  |
| --- | --- | --- | --- |
| **References** | | | |
| Ref No. | **Document Title** | **Version** | **Date** |
|  | Global Tax DMS High Level Design | v65gs.3.0 | 29-May-2009 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Version History** | | | |
| Version | **Date** | **Author(s)** | **Comments** |
| 67s.0.1 | 20-Jul-2012 | Jia Yu | Intitial design for SER456 based on v67.2.3 |
| 67s.0.2 | 23-Jul-12 | Aivin John | Updated for SER 442 |
| 67s.0.3 | 30-Jul-2012 | Jia Yu | Updated for SER456 |
| 67s.0.4 | 31-Jul-2012 | Jia Yu | Updated after internal review |
| 67s.0.5 | 31-Jul-2012 | Aivin John | Updated after internal review for SER 442 |
| 67s.0.6 | 1-Aug-2012 | Aivin John | Updated after 2nd review |
| 67s.0.7 | 2-Aug-2012 | Jia Yu | Updated after internal review |
| 67s.1.0 | 7-Aug-2012 | Sam Buttery | Marked as Ready for Client Review. |
| 67s.1.1 | 23-Aug-2012 | Aivin John | Updated after business review |
| 67s.1.2 | 03-Sept-2012 | Jia Yu | Updated after Client Review. |
| 67s.1.3 | 6-Sep-2012 | Aivin John | Updated for DA 472 |
| 67s.1.4 | 15-Apr-2012 | George Kaki | Updated for DA 497 |
| 67s.1.5 | 29-May-2013 | Aivin John | Updated for Global SER 496 |
| 67s.1.6 | 05-Apr-2013 | Aivin John | Updated after initial review. |
| 67s.1.7 | 17-Apr-2013 | Aivin John | Updated after Second review. |
| 67s.1.8 | 28-Jun-2013 | Aivin John | Updated for Global SER497 - Registration Query performance |
| 67s.1.9 | 17-Jul-2013 | Aivin John | Updated after initial review. |
| 67s.1.10 | 9-Aug-2013 | Aivin John | Updated with new method after review. |
|  | 12-Aug-2013 | Aivin John | Updated after review. |

# Table of Contents

1 Table of Contents 3

2 functional design 5

2.1 Description 5

2.2 Registration Terms and Definitions 5

2.2.1 Registration Group and Restricted Folders 5

2.2.2 Registrations 5

2.2.3 Product Specific Access 6

2.2.4 Registration Approvers 6

2.3 Registration Common Logic 6

2.3.1 Registration Task Permissions 7

2.3.2 Valid Registration Approvers 8

2.4 Assumptions 9

2.5 Business Rules 9

2.5.1 Registration submission 9

2.5.2 Registration extension 9

2.5.3 Set Auto-Approvers 10

2.5.4 Move Registrations 11

2.5.5 Get Registrations 15

2.5.6 Get Auto Approver Folders 15

2.5.7 Get Approvers 15

2.5.8 Get Approvers and Registrations 15

2.5.9 Get Automatic Approvers 15

2.5.10 Remove Approver 15

2.5.11 Remove Delegated Approver 15

2.5.12 Add Delegated Approver 16

2.5.13 Copy Registrations 16

2.5.14 Remove Registration 16

2.5.15 Get Registration Group 16

2.5.16 Mark Subclient/Job/Project Restricted 17

2.5.17 Unmark Subclient/Job/Project Restricted 18

2.5.18 Who can Mark/Unmark Restricted Subclient/Job/Project folders 18

2.5.19 Get Nominated Approvers 18

2.5.20 Add Nominated Delegated Approver 18

2.5.21 Remove Nominated Delegated Approver 19

2.5.22 Reassign Registrations 19

2.5.23 Who can Modify the Client Default Group 19

2.5.24 Update Client Default Group 19

2.5.25 Create Client Default Group Registrations 19

2.5.26 Who can Modify Approver Registration Access 19

2.5.27 Modify Approver Registration Access 19

2.5.28 Who can Set/Unset Job for Product Specific Access 20

2.5.29 Set Product Specific Access on Job 20

2.5.30 Unset Product Specific Access on Job 20

2.5.31 Can Job have Product Specific Access 20

2.5.32 Update Product Group Registrations 20

2.5.33 Update Product Specific Access Exclusions 20

2.5.34 Logging 21

2.5.35 Clean Up Deleted Folder 21

2.5.36 When can an Existing Registration be Updated 21

2.6 Outstanding Issues 22

2.7 Added / Modified / Removed Business Functionality 22

2.8 Configuration 22

2.9 Business Layer Messaging 24

2.10 System Requirements Checklist 25

2.11 Design Decisions 27

3 Technical Design 28

3.1 Overview 28

3.2 Queries 30

3.3 External Dependencies 46

3.4 TlsRegistration 47

3.4.1 Superinterfaces 47

3.5 Custom objects 48

3.6 TlsRegistrationImpl.java 50

3.7 TlsRegQuery.java 75

3.8 tls\_regb01\_manage\_Registrations 105

3.8.1 ManageRegistrations.java 105

3.8.2 RegistrationLib.java 106

3.8.3 RegistrationLibImpl.java 106

3.8.4 AddRegistration.java 112

3.8.5 RemoveRegistration.java 122

3.8.6 RegistrationApproverLib.java 128

3.8.7 ConfidentialLib.java 138

3.8.8 RegistrationLog.java 146

3.9 ModifyDefaultGroupCommand 150

3.10 CreateClientDefaultGroupRegCommand 151

3.11 ModifyRegApproverAccessCommand 152

3.12 UpdateProductGroupRegCommand 153

3.13 SetUnsetProductSpecificAccessCommand 154

3.14 UpdateProductSpecificAccessExclusionCommand 156

# functional design

## Description

This component includes the required business logic to process ‘Registrations’ within the DMS application. Registrations are used to provide users with access to documents within the Client cabinet(s) within the DMS.

This component also contains the business logic for marking or unmarking a folder Restricted within the DMS. Registrations can be managed at the Restricted folder level allowing for specific access to be granted.

## Registration Terms and Definitions

### Registration Group and Restricted Folders

A Registration group is a group which includes users who have access to the object which have Registrations created against it. It is through the adding and subtracting of users to these Registration groups that security is maintained in the system.

Registration groups are set at either the Head Client level or a Restricted folder level. At the Head Client level all users within the group have access to the Subclients, Jobs and Projects which are not marked as Restricted.

At the Restricted folder level, users who are within the group have access to the following:

* The Restricted folder and its documents.
* Sub-folders of the Restricted folder which are not marked Restricted and their documents.
* Parent folders of the Restricted folder, but not the documents contained in these parent folders.

The following folder types can be marked/unmarked Restricted within the DMS;

* Subclients
* Jobs
* Projects

### Registrations

A Registration is used to grant a user access to a folder and its associated documents within the DMS.

There are four main types of Registrations within the DMS:

#### Exipring (Default) Registration

Expiring Registrations are the default type of Registration given to users of the DMS. They provide access for a user to a given folder or Unrestricted folder tree. Expiring Registrations are given for a defined period of time at which point they will expire.

A user with an Expiring Registration is referred within this document as a Registrant.

Expiring Registrations will be created by users within the DMS.

#### Auto Approver Registration

Auto Approver Registrations are given to the Partners/Managers of Clients, Jobs or Projects to provide them with the ability to manage Registrations for Client, Job or Project they are responsible for. These Registrations will be created automatically by the System, and they will not have an expiry date.

#### Non-Approver, Non-Expiring Registration

Non-Approver, Non-Expiring Registrations are given to Partners/Managers of Unrestricted Subclients, Jobs or Projects, which are located under a Restricted Subclient or Job. These users will have Registrations to the Restricted Subclient or Job so they can view the Unrestricted Subclient, Job or Project they are responsible for. They will not be capable of creating Registrations to the Restricted Subclient or Job. These Registrations will be created automatically by the System, and will not have an expiry date.

#### Delegated Approver/Nominated Delegated Approver Registration

Delegated Approver/Nominated Delegated Approver Registrations are given to users who have been given the ability to manage Registrations to a Client, Job or Project by an Auto Approver. Delegated Approver/Nominated Delegated Approver Registrations will not have an expiry date.

### Product Specific Access

Jobs within the DMS can be configured to use Product Specific Access. If a Job has been assigned Product Specific Access, an additional group of users will be assigned access to the Job. There will be a single group for each configured Product Code within the DMS, and this group of users will be assigned to all Unrestricted Jobs under a given Product.

### Registration Approvers

A Registration Approver is a user that has the ability to grant and modify Registrations for a folder within the DMS. An Approver will only be able to modify Expiring Registrations.

There are four different ways a user can become a Registration Approver within the DMS:

#### Auto Approver

Auto Approvers are automatically given their Auto Approver status by the System for locations where they are the Partner or Manager (e.g. Client Manager) within the DMS.

#### Delegated Approver

Delegated Approvers are given Registration Approver access for a single folder within the DMS by an Auto Approver.

#### Nominated Delegated Approver

Nominated Delegated Approvers are given their Registration Approver status by an Auto Approver (Nominator) within the DMS. A Nominated Delegated Approver’s Registration Approver access is linked to the Nominator (a person) rather than a specific location.

Everywhere in the DMS that the Nominator has Auto Approver access the Nominated Delegated Approver will also gain Registration Approver access.

#### Registration Administrator

These are members of the Security Admin Group (imp\_tls\_security\_admin).

## Registration Common Logic

The Registrations sub-system allows the submission of ‘Registrations’ and after affirming that the user is a valid ‘Approver’ will issue a ‘Registration’ for the requested amount of time.

The management of Registrations is achieved by a number of utility functions such as extend a Registration, expire a Registration and the setting of ‘Auto-Registrations’ in which system defined roles (e.g., Project Partner, Project Manager) are given non-expiring access.

Another primary function of this component is the ability to mark or unmark a Subclient, Job or Project as ‘Restricted’. Marking a Subclient, Job or Project as ‘Restricted’ will set up a new set of Registrations for that folder. Only users who are directly registered to the Restricted folder will have access to the folder, sub-folders and contents.

Client Default Group and Product Specific Access Registrations are managed by the REG\_B05 component, and cannot be managed by a Registration Approver.

There are a number of ways in which Registrations can be created:

* If a user is a valid Registration Approver they can add users and set a time period which that user has access to the object.
* When a Client, Job or Project is created, a set of ‘Auto-Registrations’ occur which are system defined and do not expire.
* Registrations can be added by moving and re-assigning Clients/Jobs.
* When the properties of a Client, Job or Project are changed (e.g., the Project Partner is changed) then a new Registration will be created.
* When a user is added as a Nominated Delegated Approver through the Nominate Delegated Approver window (REG\_W06), this will not overwrite that user’s existing Delegated Approver or Auto Approver access.
* When a nominator is given Auto Approver access to a folder the Nominated Delegated Approver for the user will be given a registration to the folder as well, this will not overwrite that user’s existing Delegated Approver or Auto Approver access.
* Registrations will be added for all users in the group assigned as a Client Default Group on Client creation via REG\_B05 - Update Registrations for Client/Product Access Groups.
* Registrations will be added to Product Specific Access groups when a user is added to the configured Product group. Registrations will be updated via the ‘REG\_B05 – Update Registrations for Client/Product Access Groups’ component.

There are a number of ways in which Registrations can be removed or modified:

* The component Expire and Clean up Registrations (REG\_B02) is a scheduled job which removes all expired Registrations.
* When the properties of a Client, Job or Project are changed, a call to update the Approver should be made which changes the Registration for the user who was removed to be a default time rather than non-expiring. The default time can be configured using ‘/Registration/DefaultDuration’.
* Security Registrations (REG\_W01) can extend a Registration or remove a Registration.
* When a user is removed as a Nominated Delegated Approver for an Auto Approver.
* When a nominator of the Nominated Delegated Approver loses Auto Approver access to a folder.
* Registrations for the Client will be deleted when a Client or Subclient is deactivated. The deactivated Client’s Auto Approver and Non-Expiring registrations to folders above the Client for the Client Manager, Client Partner will also be removed if the Client Manager, Client Partner are not listed as an Auto Approver for other Subclients, Jobs or Projects.
* Registrations will be removed when a user expires registrations via ‘My Registration’ page of security registration window.
* Registrations will be modified when a user extends registrations via ‘My Registration’ page of security registration window.
* Registrations will be modified for all users in the group assigned as a Client Default Group via REG\_B05 - Update Registrations for Client/Product Access Groups.
* Registrations will be removed from Product Specific Access groups when a user is removed from the configured Product group. Registrations will be updated via the ‘REG\_B05 – Update Registrations for Client/Product Access Groups’ component.

### Registration Task Permissions

The following table details which user levels can perform Registration tasks.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Task \ User** | **Expiring Registration** | **Delegated Approver** | **Nominated Delegated Approver** | **Auto Approver** | **Registration Admin** |
| **Modify Expiring Registrations** | Only if Registration Approver Access is Registrants | Yes | Yes | Yes | Yes |
| **Modify Delegated Approvers** | No | No | No | Yes | Yes |
| **Modify Nominated Delegated Approvers** | No | No | No | Yes | No |
| **Mark Client/Job/Project Restricted** | On Creation | On Creation | On Creation | Yes | Yes |
| **Unmark Client/Job/Project Restricted** | No | No | No | Yes | Yes |
| **Set Product Specific Access on Job** | No | No | No | Yes | Yes |
| **Unset Product Specific Access on Job** | No | No | No | Yes | Yes |

### Valid Registration Approvers

The following users are considered valid Registration Approvers at the appropriate folder levels:

The members of the Security Admin Group (imp\_tls\_security\_admin) will be called Registration Administrators.

##### Head Client:

1. Client Partner [or Tax Client Partner if configured] for the Head Client.
2. Client Manager [or Tax Client Manager if configured] for the Head Client.

**Note:** Throughout this document Client Partner and Client Manager will be interchangeable with Tax Client Partner and Tax Client Manager depending on whichever is configured for use based on ‘/CAT\_B77/UseTaxPartnerManager’. Only one of either Partner/Manager or Tax Client Partner/Tax Client Manager will be used, never both. The Tax Client Partner/Manager can be utilised when the Client Partner/Manager are not within the Tax LoS to ensure that there is a Tax resource managing security for a Client.

1. Auto Approvers of Subclients, Jobs and Projects which aren't Restricted or within a Restricted folder. These are Client Partners, Client Managers for Unrestricted Subclients, Job Partners and Job Managers for Unrestricted Jobs, and Project Partners and Project Managers for Unrestricted Projects.
2. Delegated Approvers at the Head Client level.
3. Registration Administrator.
4. Registrants can be Registration Approvers if the Auto Approver has granted access for the registrants to add Registrations.

**Restricted Subclient:**

1. Client Partner [or Tax Client Partner if configured] for the Subclient.
2. Client Manager [or Tax Client Partner if configured] for the Subclient.
3. Delegated Approvers for the Subclient.
4. Registration Administrator.
5. Registrants can be Registration Approvers if the Auto Approver has granted access to the registrants to add Registrations.

##### Restricted Job:

1. Job Partner.
2. Job Manager.
3. Delegated Approvers for the Job.
4. Registration Administrator.
5. Registrants can be Registration Approvers if the Auto Approver has granted access to the registrants to add Registrations.

##### Restricted Project:

1. Project Partner.
2. Project Manager.
3. Delegated Approvers for the Project.
4. Registration Administrator.
5. Registrants can be Registration Approvers if the Auto Approver has granted access to the Registrants to add Registrations.

## Assumptions

1. The security layer will be responsible for the setting of security permissions for the Registrations group.
2. The user will not directly interact with this component. Other components will call this component to perform functionality.
3. Calling components will be responsible for the removing of automatic Registrations. For instance; Project Properties will need to call this component passing the Project Manager and/or Project Partner that was removed.

## Business Rules

The following sections describe the business rule for the operations that this component implements.

### Registration submission

A Registration submission is the responsibility of other modules (e.g., REG\_W01), they are generally AUTO generated (e.g., the creation of a new Project will trigger a set of automatic non-expiring Registrations), USER generated (e.g., REG\_W02 - Add Registrations) or SYSTEM generated (e.g., Copy Registrations).

The following occurs when a registration is submitted:

1. Validate that the requestor is a valid Registration Approver (as defined in section 2.3.2 Valid Registration Approvers) or, SYSTEM generated.
2. If the folder is a Restricted folder then the request will go against that folder. Otherwise it will go against the Head Client folder.
3. For an AUTO generated Registration, if a user already has a Registration against the object then this request will overwrite the old request.
4. For a USER generated Registration, if the user already has a Registration against the object then the Registration will be updated with the new information unless the old Registration was an AUTO Registration or Non Expiring Registration.
5. AUTO generated Registrations will be given Auto-Approver status and no expiry date.
6. Non Expiring Registrations will have an Approver Status of ‘NONE’ and no expiry date.

### Registration extension

A Registrations extension is the responsibility of other modules (e.g., REG\_W04 Manage Registrations).

In common logic the following is performed:

1. Validate that the requestor is a valid Registration Approver for the object and that there is an existing Registration for the object.
2. Update the time period of the Registration to be today’s date plus the amount of days requested. NOTE: If user has an EXPIRED Registration then the user will have its Registration re-instated for the period given.
3. Will not update Non-Expiring Registrations.

### Set Auto-Approvers

This function sets the Auto-Approvers for an object depending on if it has been marked as Restricted and what type of object it is.

If the object is Unrestricted, this method will also give the object’s Auto Approvers either:

1. Non-expiring registrations to the first Restricted parent above it.
2. Or if no Restricted parent exists above the object, Auto Approver registrations to the Head Client.

If the object is Restricted, the method will give the object’s Auto Approvers Auto Approver registrations to the object. It will then remove these Auto Approvers’ registrations from the first Restricted parent above the folder (or the Head Client) if they are no longer Auto Approvers for other Unrestricted objects underneath this parent. Note that these registrations will be removed not by giving the Auto Approver registration to the parent for the default time period, but by completely deleting the registrations at the parent.

See the following sections for what auto-Approvers are set up for each object type.

#### Set Auto-Approvers for Head Client

This method will be called to establish Approvers for the given Head Client. It will add Auto Approver registrations to the Client for, Client Manager and Client Partner.

#### Set Auto-Approvers for Subclient

When a Subclient is created or modified this method will be called as it will establish Approvers for the given Subclient. This is achieved by adding the Client Partner and Client Manager to the Registrations and setting them as Approvers.

If the Subclient is Restricted this method will give Auto Approver registrations to the Client, and remove their Auto Approver registrations at the Head Client if they are no longer Auto Approvers for Unrestricted folders beneath this. If the Subclient is Unrestricted it will give the Auto Approvers, Auto Approver registrations to the Head Client.

#### Set Auto-Approvers for Job

When a Job is created or modified this method will be called as it will establish Approvers for a given Job. A Job’s Auto Approvers are the Job Manager and Job Partner.

If the Job is Restricted this method will give these Auto Approvers Auto Approver registrations to the Job, and will remove their Auto Approver registrations at the Head Client/restricted parent if they are no longer Auto Approvers for Unrestricted folders beneath this. If the Job is Unrestricted and is within a Restricted Subclient, it will add the Auto Approvers’ non-expiring Registration to Restricted Subclient. If the Job is Unrestricted and is not within a Restricted Subclient it will add an Auto Approver registration to the Head Client.

#### Set Auto-Approvers for Project

When a Project is created or modified this method will be called as it will establish Approvers for a given Project. A Project’s Auto Approvers are the Project Partner and Project Manager.

If the Project is Restricted this method will give these Auto Approvers Auto Approver registrations to the Project, and will remove their Auto Approver registrations at the Head Client/restricted parent if they are no longer Auto Approvers for Unrestricted folders beneath this. If the Project is Unrestricted and is within a Restricted parent, it will add these Auto Approvers’ non-expiring registration to the first Restricted parent above it. If the Project is Unrestricted and is not within a Restricted parent, it will add an Auto Approver registration to the Head Client.

### Move Registrations

Move Registrations accepts as parameters the old Restricted parent id (i.e. Head Client or Restricted parent folder) and the object that has been moved.

It does the following when a Client, Job or Project is moved:

1. Removes the Auto Approvers from the old Head Client / Restricted parent folder (e.g., Job Manager and Job Partner if moving a Job). Detailed business rules are covered in the section **Remove Approver**.
2. Adds the Auto Approvers of the moved folder and of all contained subfolders (e.g. Jobs and/or Projects) to the new Head Client or Restricted parent folder.
3. If the folder being moved is Restricted then remove the Restricted group from the Read Only group of the old parent. This is necessary to ensure that users authorized to access the moved folder will no longer be able to access the old parent.
4. If the object being moved has a Read Only group, then remove the Read Only group from the Read Only group of the old parent. This is necessary to ensure that users authorized to access Restricted subfolders of the moved folder will no longer be able to access the old parent.
5. If the new parent object is not Restricted then copy the old Head Client / Restricted parent registrations to the new Head Client / Restricted parent. Registrations without an expiration date e.g. approver registrations for other folders that are not being moved will be assigned the configurable default expiration period when copied.
6. Restricted groups as well as read-only groups of the moved folder and of all contained subfolders will be added to the corresponding groups of the new Head Client / Restricted parent by the component Security & ACLs (CFG\_C04).

The following tables state the applicable business rules for each possible case.

#### Moving Clients

Note that for all cases, the following business rule also applies, but has not been included for brevity:

If the Client being moved is a Subclient and it has a Read Only group, remove the Read Only group from the Read Only group of the old Head Client.

| **Original Client Type (Pre-Move)** | **New Client Type (Post-Move)** | **Applicable Business Rules** |
| --- | --- | --- |
| Head Client | Unrestricted  Head Client / Subclient | • Auto Approvers of the moved Head Client and Auto Approvers of all contained Jobs and Projects will be granted Auto Approver registrations for the new Head Client. Business rules from the section **Set Auto-Approvers** will apply.  • Non-approver registrations of the Head Client being moved will be reassigned to the new parent i.e. Head Client. |
| Unrestricted  Subclient | Unrestricted  Subclient | • Auto Approvers of the moved Subclient and Auto Approvers of all contained Jobs and Projects will be granted Auto Approver registrations for the new Head Client. Business rules from the section **Set Auto-Approvers** will apply.  • Auto Approvers of the moved Subclient and Auto Approvers of all contained Jobs and Projects will be removed as Auto Approvers from the old Head Client. Business rules from the section **Remove Approver** will apply. |
| Unrestricted  Subclient | Head Client |
| Restricted Subclient | Head Client | • Registrations will not be changed since they are specific to the Subclient that is being moved.  • Restricted group of the moved Subclient will be removed from the read-only group of the old parent i.e. Head Client.  • Restricted group will be added to the new parent read-only group when the new client type is a Subclient. |
| Restricted Subclient | Restricted Subclient |

#### Moving Jobs

Job registration changes will be contained to within the same Client between different products. The functionality for moving Job registrations will remain as stated below even if not fully utilised. Those scenarios that are currently used in the Global DMS are marked with an asterisk (\*) and shaded.

Note that for all cases, the following business rule also applies, but has not been included for brevity:

If the Job has a Read Only group, remove the Read Only group from the Read Only group of the old parent.

| **Used in Global DMS** | **Job Restricted?** | **Moved From Restricted Subclient?** | **Moved To Restricted Subclient?** | **Moved to same Head Client?** | **Applicable Business Rules** |
| --- | --- | --- | --- | --- | --- |
| \* | N | N | N | Y | *CASE 1*  • Nothing needs to be done as Unrestricted user group at Head Client level will have access to Job before and after the move. If moved to a Product using Product Specific Access the Job will inherit the Client security plus gain the Product Specific Access. If moved from a Product using Product Specific Security the Job will be changed to inherit from the new parent Client. |
| \* | Y | N | N | Y | *CASE 2*  • Nothing needs to change as the restricted group of the job will remain assigned to the job in the new location and will still remain in the read-only group of the parent Client. |
|  | N | N | Y | N | *CASE 3*  • Valid Approvers of the Job being moved will be given registrations for a non-expiring period of time to the new Restricted parent, but they will not have approval authority. In this instance valid Approvers are the Partners and Managers at the Job and Project levels. |
|  | N | Y | Y | N |
|  | N | N | Y | Y |
|  | N | Y | N | Y | *CASE 4*  • Valid Approvers of the Job being moved will be given registrations for a non-expiring period of time to the Head Client (which in turn gives them access to the new Unrestricted parent). In this instance valid Approvers are the Partners and Managers at the Job and Project levels. |
|  | N | N | N | N | *CASE 5*  • User registrations from Unrestricted group of first Head Client will be added to Unrestricted group of second Head Client. • Only the users who are members of first Head Client Unrestricted group and not members of second Head Client Unrestricted group will be added. |
|  | N | Y | N | N | *CASE 6*  • User registrations from Restricted group of first Subclient will be added to Unrestricted group of (new) Head Client. • Only the users who are members of Restricted group of first Subclient and not members of (new) Head Client Unrestricted group will be added. |
|  | N | Y | Y | Y |
|  | Y | N | Y | N | *CASE 7*  • User group with access to Restricted Job will be removed as sub group of first Head Client Unrestricted group.  • User group with access to Restricted Job will be added as sub group of second Subclient Restricted group. • The Restricted group of the Job being moved will be removed from the old parent's read only group. |
|  | Y | N | Y | Y |
|  | Y | N | N | N | *CASE 8*  • User group with access to Restricted Job will be removed as sub group of first Head Client Unrestricted group. • The Restricted group of the Job being moved will be removed from the old parent's read only group. |
|  | Y | Y | N | Y | *CASE 9*  • User group with access to Restricted Job will be removed as a sub group of first Subclient Restricted group.  • The Restricted group of the Job being moved will be removed from the old parent's read only group. |
|  | Y | Y | Y | N |
|  | Y | Y | N | N | *CASE 10*  • User group with access to Restricted Job will be removed as sub group of first Subclient Restricted group. • The Restricted group of the Job being moved will be removed from the old parent's read only group. |
|  | Y | Y | Y | Y |

#### Moving/Copying/Rolling over Projects

The following table covers the scenarios when Project is moved. The business rules are as stated in the above table and referenced via case number so they are not repeated.

Project registration changes will be contained to within the same Client between different products. The functionality for migrating Project registrations will remain as stated below even if not fully utilised. Those scenarios that are currently used in the Global DMS are marked with an asterisk (\*) and shaded.

| **Used in Global DMS?** | **Project Restricted?** | **Moved from Restricted Job?** | **Moved to Restricted Job?** | **Moved from Restricted Subclient?** | **Moved to Restricted Subclient?** | **Business Rule (Case # from above)** |
| --- | --- | --- | --- | --- | --- | --- |
|  | Y | Y | Y | Y | Y | 10 |
|  | Y | Y | Y | Y | N | 10 |
|  | Y | Y | Y | N | Y | 10 |
| \* | Y | Y | Y | N | N | 10 |
|  | Y | Y | N | Y | Y | 10 |
|  | Y | Y | N | Y | N | 10 |
|  | Y | Y | N | N | Y | 10 |
| \* | Y | Y | N | N | N | 10 |
|  | Y | N | Y | Y | Y | 7 |
|  | Y | N | Y | Y | N | 8 |
|  | Y | N | Y | N | Y | 7 |
| \* | Y | N | Y | N | N | 8 |
|  | Y | N | N | Y | Y | 7 |
|  | Y | N | N | Y | N | 8 |
|  | Y | N | N | N | Y | 7 |
| \* | Y | N | N | N | N | 8 |
|  | N | Y | Y | Y | Y | 3 |
|  | N | Y | Y | Y | N | 3 |
|  | N | Y | Y | N | Y | 3 |
| \* | N | Y | Y | N | N | 3 |
|  | N | Y | N | Y | Y | 4 |
|  | N | Y | N | Y | N | 4 |
|  | N | Y | N | N | Y | 4 |
| \* | N | Y | N | N | N | 4 |
|  | N | N | Y | Y | Y | 3 |
|  | N | N | Y | Y | N | 3 |
|  | N | N | Y | N | Y | 3 |
| \* | N | N | Y | N | N | 3 |
|  | N | N | N | Y | Y | 3 |
|  | N | N | N | Y | N | 4 |
|  | N | N | N | N | Y | 3 |
| \* | N | N | N | N | N | 1 |

### Get Registrations

This function will retrieve Registrations based on a filter condition.

### Get Auto Approver Folders

This function will retrieve folders for which the logged on user is an Auto Approver.

### Get Approvers

This function will retrieve Registration Approvers based on a filter condition. Auto Approvers that are only in the Security Admin group (imp\_tls\_security\_admin) shall not be returned.

### Get Approvers and Registrations

This function will retrieve Approvers and their registrations for a given folder. Auto Approvers that are only in the Security Admin group (imp\_tls\_security\_admin) shall not be returned.This method can return a list of All Approvers or only Active Approvers for the folder.

### Get Automatic Approvers

This method returns the automatic Approvers for a given folder.

### Remove Approver

This function will be called after updating the Client/Job/Project properties and changing their Approvers. The calling function will pass the old Approver. There are two scenarios:

1. Remove an Approver for a Restricted folder. In this case the user gets a new “Expiring” Registration for the default access time and has their Approver Registration removed.
2. Remove an Approver for an Unrestricted folder. In this case a check will be made to see if the removed Approver is a valid Approver based on other Clients/Jobs/Projects ( for instance a Project Partner may be removed from Project A but still be on Project B, in this case they should not be removed). If the removed Approver is still a valid Approver then the Approver will not be removed otherwise they will have their approver status removed and a default expiry date.

Note: The updated registration will not be AUTO and therefore able to be changed.

### Remove Delegated Approver

This removes a Delegated Approver from a folder. The method will check if the user being removed has a Delegated Approver Registration to the folder. If they do have a Delegated Approver Registration, it will be updated to a default expiring Registration. If the user does not have a Delegated Approver Registration, no update will be made.

### Add Delegated Approver

This adds a Delegated Approver for a given folder giving them non-expiring registration access. There is a configuration object that limits the number of Delegated Approvers which can be added to a single object. This configuration item has no upper limit and can be set appropriately by the business.

Add delegated approver can operate in a batch mode where a list of folders can be specified to add Delegated Approvers to.

### Copy Registrations

This function copies the Registrations of one object to another object.

When Registrations are copied during a Project Rollover or Project Copy procedure the following will occur:

1. Only registrations with a status of ACTIVE will be copied.
2. Delegated Approvers of the copied folder will be given access to the new folder as Delegated Approvers.
3. Auto Approvers of the copied folder will be given default access to the new folder. This will not remove the user’s Auto Approver access if they are the Partner, Partner or Manager of the new folder.
4. All other copied registrations will expire on the same day as the original registrations.

### Remove Registration

This function is used to remove a user from a Registration group. Multiple Registrations can be removed in a single request.

### Get Registration Group

This function will get the Registration group name for an object. The naming convention will be **reg\_<r\_object\_id>**

Groups beginning with ‘reg’ will be assigned to Restricted Clients, Jobs, Projects and the associated Unrestricted folders under the Restricted folder.

### Mark Subclient/Job/Project Restricted

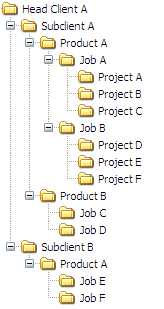
If the user is able to modify Restricted folders then when a folder is marked Restricted the following occurs:

1. The t\_confidential flag is set to true.
2. A new group is created which will contain users who have access to the ‘Restricted’ folder.
3. The members of that group are set up depending on what type of folder has been marked as Restricted. See ‘Set up auto-Approvers’ for further information.
4. The new Restricted group is set as a ‘Restricted’ group for the selected folder.
5. The Head Client group is removed from the selected folder.
6. If there are any Unrestricted folders beneath the ‘Restricted’ folder, they will have their existing group removed, and replaced with the new ‘Restricted’ folder group.
7. The Restricted folder group is added as a ‘Read only’ group to the folder group above the new ‘Restricted’ folder to allow navigation.
8. Until the Head Client folder is updated, or a parent folder already contains the ‘Read only’ group of the current folder, traverse up the tree, adding the ‘Read only’ group of each folder to the ‘Read only’ group of its parent folder.

For example, below all folders’ Read only groups have been updated after Project A has been marked Restricted.

**Read-only\_Group\_HeadClient\_A**

* Read-only\_Group\_Subclient\_A



**Restricted\_Group\_Proj\_A**

* User A
* User B

**Read-only\_Group\_Job\_A**

* Restricted\_Group\_Proj\_A

**Read-only\_Group\_Subclient\_A**

* Read-only\_Group\_Job\_A

**NOTE:** Product A will share the read-only group of Subclient A.

1. If the configuration item '/REG\_B01/DeleteAutoApprovers/DefaultAccessEnabled' is set to false or does not exist, for each Unrestricted folder under the newly marked Restricted folder, delete the Auto Approvers’ registrations to their old Restricted parent if they are no longer Auto Approvers for Unrestricted folders under their old Restricted parent.
2. If the configuration item ''/REG\_B01/DeleteAutoApprovers/DefaultAccessEnabled' is set to true, for each Unrestricted folder under the newly marked Restricted folder, replace the existing Auto Approver registration with a default registration to the old Restricted parent if the user is no longer an Auto Approvers for Unrestricted folders under the old Restricted parent.
3. For each Unrestricted folder within the newly marked Restricted folder, call setAutoApprovers. This will add the required Auto Approvers to the newly Restricted folder.
4. Apply security to this folder.
5. Set the ability to add registrations to: ‘Approvers Only’.
6. If a Job has been set to use Product Specific Access, the Product Specific Access (t\_product\_group\_access) flag will be set to false.
7. After a Client, Job or Project folder is successfully marked Restricted and if the configuration item ‘/REG\_B01/IsMarkUnmarkRegistrationLogEnabled’ is set to TRUE the following information will be logged into Registration Log table:

The object id of the folder marked restricted.

The action (in this case ‘Marked Restricted’).

The user who performed the action.

The time when the user marked the folder Restricted.

For more information about Registration logging refer to section [2.5.34](#_Logging) of this document.

### Unmark Subclient/Job/Project Restricted

If the user is able to modify Restricted folders then when a Restricted folder is unmarked the following occurs:

1. The Restricted group associated with the Restricted folder is cleared. NOTE: Users who were registered only at the Restricted group level will be added as a new registration with default access to parent Restricted group. E.g., if a Restricted Project is unmarked Restricted then the Restricted Job will have registrations created for the removed users.
2. The Restricted group is removed from the ‘Read only’ group of the folder above it.
3. The t\_confidential flag is set to false.
4. The setAutoApprovers method is called on the folder and Unrestricted folders beneath it so that these Auto Approvers are added to the new Restricted parent.
5. Apply ACL is called.
6. Clear the ability to add registration setting so that inherits from parent.
7. For Clients, the Default Group will be cleared.
8. For Jobs, if it is under a Product configured to use Product Specific Access, the Product Specific Access (t\_product\_group\_access) flag will be set to true.
9. After a Client, Job or Project folder is successfully Unmarked Restricted and if the configuration item ‘/REG\_B01/IsMarkUnmarkRegistrationLogEnabled’ is set to TRUE the following information will be logged into Registration Log table:

a) The object id of the folder Unmarked restricted.

b) The action (in this case ‘Unmarked Restricted’).

c) The user who Unmarked the folder.

d) The time when the user Unmarked the folder Restricted.

For more information about Registration logging refer to section [2.5.34](#_Logging) of this document.

### Who can Mark/Unmark Restricted Subclient/Job/Project folders

The users who can modify a Restricted folder depends on the folder being selected:

1. At the Project level the Project Partner, Project Manager and the Registration Administrator
2. At the Job level the Job Partner, Job Manager and the Registration Administrator.
3. At the Client level the Client Manager, Client Partner and Registration Administrator. Client Partners/Managers will not be able to unmark a Client that has a Client Default Group assigned, unless the configuration item ‘REG\_B01/ClientApproverCanModifyDefAccessGroup‘ is set to ‘True’. Unmarking a Client with a Default Group, will result in the Default Group being removed.
4. Head Clients cannot be Marked/Unmarked Restricted within the DMS.

### Get Nominated Approvers

This will return list of users who are currently a nominated approver for the logged in user.

### Add Nominated Delegated Approver

This will create a nominated delegated approver registration to a folder for a given user. The registration will allow a user to modify and create normal registrations to the given folder. A nominated delegated approver will be able give delegated approver access to a folder .If a user already has an existing Auto Approver or Delegated Approver registration to the folder, the existing registration will not be modified. If they have an expiring registration to the folder, the registration will be updated to a nominated delegated approver registration.

The user who requested the nominated delegated approver will be listed as the approver for the registration, if a registration has to be created or modified. If multiple users have nominated the same user, which would result in the nominated user having access to a folder twice, they will only be given a single registration and only one user will be listed as the approver.

### Remove Nominated Delegated Approver

This will delete a nominated delegated approver registration. The Registration will be removed during the next execution of the scheduled Job (REG\_B04).

### Reassign Registrations

When a Pending Client is reassigned the following updates will be made to registrations:

1. If the Existing Subclient is Restricted, only the Auto Approvers of the Jobs, and Projects of the Pending Client will be given registrations to the Head Client, and Restricted Subclient.
2. If the Existing Subclient is Unrestricted, then the registrations will be copied from the Pending Client to the Head Client.
3. Normal users will get the same access they had to the Pending Client to the new Head Client.
4. Delegated Approvers will be made Delegated Approvers to the new Head Client.
5. Auto Approvers of the Pending Client will be given registrations for the default period of time to the new Head Client.

### Who can Modify the Client Default Group

This will validate who can modify the Client Default Group based on the following rules:

1. At the Client level if ‘/REG\_B01/ClientApproverCanModifyDefAccessGroup’ is True then the Client Partner and Client Manager as well as Registration Administrators can modify the Client Default Group, otherwise only the Registration Administrators can modify the Client Default Group.

### Update Client Default Group

Updates the Client Default Group attributes with a new group value.

### Create Client Default Group Registrations

Creates Registrations at a Client for all users in the group assigned as the Client Default Group (This will only be called during Client creation). Registrations will be created for length of time defined in the configuration item (‘/REG\_B01/DefaultGroupRegDuration’).

### Who can Modify Approver Registration Access

The users who can modify Approver Registration Access are:

1. At the Client level the Client Partner and Client Manager.
2. At the Job level the Job Partner and Job Manager.
3. At the Project level the Project Partner and Project Manager.
4. At all levels a Registration Administrator user.

### Modify Approver Registration Access

Updates the Approver Registration Access attributes on Clients, Jobs and Projects.

1. Updates the access setting to be one of either “Only Approvers” or “Registrants & Approvers”
2. Updates the GUID of the user who last modified the access setting to the requesting user.
3. Can only be updated if a user has permission to modify Approver Registration Access (as per 2.5.26)

### Who can Set/Unset Job for Product Specific Access

This will validate when a Job can be marked/unmarked for Product Specific Access based on the following rules:

1. The Job Partner and Job Manager can modify the Product Specific Access of a Job.
2. Registration Administrators can modify the Product Specific Access for a Job.

### Set Product Specific Access on Job

Jobs will be set to use Product Specific Access on creation or when set via the UI.

When a Job is set for Product Specific Access the following changes will occur:

1. Product Access Registration group ‘prd\_<product\_item\_id>’ will be created and assigned to the Job and Unrestricted sub-folders.
2. Registrations to the Product Access Registration Group will be maintained by the REG\_B05 - Update Registrations for Client & Product Access Groups component. Registrations already created for the Product in the system will gain access at Job creation.
3. The Product Access Registration group is added to the ‘Read only’ group of the parent Client.
4. The Product Specific Access (t\_product\_group\_access) flag of the Job will be set to true.

### Unset Product Specific Access on Job

Jobs can have Product Specific Access unset via the UI.

When a Job is unset for Product Specific Access the following changes will occur:

1. The Job’s security will revert to inherit from its parent Client.
2. The Product Registration group will be removed from the ‘Read only’ group of the parent Client.
3. The Product Registration group will be removed from the Job and Unrestricted sub-folders.
4. Users in the Product Group will NOT be given default Registrations to the parent folder.
5. The Product Specific Access (t\_product\_group\_access) flag of the Job will be set to false.

### Can Job have Product Specific Access

This will validate whether a Job can have Product Specific Access based on the following rules being meet:

1. Product Specific Access must be enabled, by setting the configuration item ‘/Registration/ProductSpecificAccess/Enabled’ to true.
2. The parent Client is not excluded from Product Specific Access via the REG\_W11 - Manage Default Group Exclusions component.
3. The Job’s Product is listed within ‘/Registration/ProductSpecificAccess/ProductGroups’
4. The Job is not Restricted.

### Update Product Group Registrations

Updates Registrations at a Product level for all users in the group assigned to the Product code in configuration (‘/Registration/ProductSpecificAccess/ProductGroups’).

1. Registrations will be created as Non-expiry Registrations for users within the group.
2. Registrations will be removed for users no longer within the group.

### Update Product Specific Access Exclusions

Updates the Clients that are currently excluded from using Product Specific Access..

1. Only members of the groups defined in ‘/REG\_W11/AuthorisedGroups’ can make changes to the Product Specific Access Exclusion List.
2. When adding a Client that already exists in the Product Specific Access Exclusion List, no updates will be performed, and no error will be displayed.
3. When removing a Client from the Product Specific Access Exclusion List, if the Client cannot be found, then no updates will be performed, and no error will be displayed.
4. When adding a Client, the following information will be added to the Product Specific Access Exclusion List:
   * 1. Object Id of the Client.
     2. Date the Client was excluded.
     3. User the Client was excluded by.
5. When removing a Client from the Product Specific Access Exclusion List, the entire row is removed from the list.
6. When adding a Client to the Product Specific Access Exclusion List, validation will be performed to ensure that a Client with Object Id exists within the DMS. If the Client does not exist no updates will be made and no error will be displayed.
7. When adding or removing a Client to/from the Product Specific Access Exclusion List an audit log entry will be added to the Registration log table.

### Logging

The following will be logged into a new custom table:

1. When a new Registration is added.
2. When a Registration is modified (NOTE: this includes the adding of Approver status to the Registration).
3. When a Registration is removed.
4. When a Client, Job or Project folder is Marked or Unmarked Restricted (if configured).

The following is logged:

1. The object id of the object which the Registration was against.
2. The action.
3. The user who initiated the action.
4. The user who the Registration was for (Blank when a folder is Marked/Unmarked restricted).
5. The date.

### Clean Up Deleted Folder

This function will clean up all registrations of the following folders:

* Deactivated Client or Subclient

Auto Approver and Non-Expiring registrations to folders above the Subclient for Client Manager, Client Partner will be also be removed if they are not listed as Auto Approvers for other Subclients, Jobs or Projects.

### When can an Existing Registration be Updated

Listed below are the Registration types used through out the DMS in their order of precedence:

* 1. Auto-Approver Registration.
  2. Delegated Approver/Nominated Delegated Approver Registration.
  3. Non-Apporver, Non-Expiring Registration.
  4. Expiring Registration.

Listed below are details as to when one type of Registration is allowed to replace an existing Registration. These rules will only apply if a user already has an existing Registration to a given folder, and this is being updated with another Registration request. If the new Registration is not allowed to overwrite the existing Registration, then the existing Registration will not be updated.

1. An Auto-Approver Registration can be overwritten by the following types of Registrations:
   1. None.
2. A Delegated Approver/Nominated Delegated Approver Registration can only be overwritten by the following types of Registrations:
   1. Auto-Approver Registration.
3. A Non-Approver, Non-Expiring Registration can only be overwritten by the following types of Registrations:
   1. Auto-Approver Registration.
   2. Delegated Approver/Nominated Delegated Approver Registration.
4. An Expiring Registrations can only be overwritten by the following types of Registrations:
   1. Auto-Approver Registration.
   2. Delegated Approver/Nominated Delegated Approver Registration.
   3. Non-Approver, Non-Expiring Registration.
   4. Expiring Registration.

## Outstanding Issues

1. Clarification required on whether the Registration Approver Access setting (“Authorized to create Registrations”) should be able to be modified by Auto Approvers and Registration Admin users or whether configuration should be created to define admins only.
2. The current Product specific security implementation should be re-implemented using web services in the future so Registration changes to groups can be submitted from 3rd party systems.
3. When removing or adding new Product Group configuration items a conversion script (or manual modification) will be required to modify existing folders in the DMS to either mark or unmark them for Product Specific Access and update their security. New Job folders created will use the new configuration. At this point in time the conversion script has not been created, and will be looked at outside of the initial D6.7 Registration release.

## Added / Modified / Removed Business Functionality

The following functionality has been added, modified or removed from the US implementation of the component:

**Added / Modified**

1. 2nd and 3rd level Client support.

**Removed**

1. Archive docbase support.
2. Copy / Delete / Move / Rollover Job
3. Delete / Move Project

**Note:** Removed functionality will be commented out from the code to allow future implementations if required.

## Configuration

The following configuration items are used by this component:

| Name | Item/Alternate Value | Description | New? |
| --- | --- | --- | --- |
| /Registration/DefaultDuration | 30 | The default duration (in days) for newly created registration if none specified when adding. | Yes |
| /Registration/RegistrationDurations | 7, 30, 90, 365 | List of possible durations (in calendar days) that registrations can be set for when adding. | Yes |
| /Registration/CleanUpJob | 90 | Number of days after which expired registrations will be cleaned up. | Yes |
| /Registration/DelegatedApproverLimit | 80 | Max limit of Delegated Approvers, will not let you add past this when returning from pavs to add user. | Yes |
| /REG\_B01/Approver | Approver | Value used in t\_create\_reg attribute when only Approvers can create registrations. | Yes |
| /REG\_B01/RegistrantAndApprover | All | Value used in t\_create\_reg attribute when Registrants and Approvers can create registrations. | Yes |
| /REG\_B01/ClientApproverCanModifyDefAccessGroup | False | Whether Client Partner and Client Manager can modify the default access group for Clients, if False only Registration Admin users can modify. | Yes |
| /CAT\_B77/UseTaxPartnerManager | True/False | Configuration that defines whether the Client Manager/Partner or **Tax** Client Manager/Partner are used for Auto Approvers. | Yes |
| /REG\_B01/DefaultGroupRegDuration | 14 | Number of days used as duration when initially creating Registrations for Client default group members. | Yes |
| /Registration/ProductSpecificAccess/ProductGroups | <Product Code> / <Group Name> | Contains a list of Product Codes (t\_product\_code) and associated Group (group\_name) for Product specific security. For each Product Code listed in this configuration item there must be a group name. | Yes |
| /Registration/ProductSpecificAccess/Enabled | True/False | Whether Product Specific Access is enabled in the DMS. | Yes |
| /REG\_W11/AuthorisedGroups | <imp\_tls\_security\_admin > | The groups of users who have access to this window and to modify the Product Specific Access Client Exclusion List. | Yes |
| /REG\_B01/DeleteAutoApprovers/DefaultAccessEnabled | True/False | Determines whether or not a user will be given a Registration for the Default time period when the user loses their Auto Approver access to a parent folder due to the current being marked as Restricted.  If set to TRUE, a default Registration will be given to the restricted parent /Head Client and has their Approver Registration removed.  If set to FALSE, the restricted parent / Head Client will completely loses their Approver Registration. | Yes |
| /REG\_B01/LogFolderRestrictionChange | True/False | When a folder is Marked or Unmarked Restricted the configuration item determines if the information about the action will be logged into the Registration Log table (t\_registration\_log). If the configuration is set to TRUE , the below information will be logged :   1. The user who carried out the action. 2. The date and time the action was carried out. 3. The folder the action was carried out on. 4. The action that was carried out (Marked Restricted OR Unmarked Restricted)   If the configuration item is set to FALSE if the configuration item is not found, the above information relating to Marking or Unmarking Resticted on a folder will not be logged into the Registration Log table. | Yes |
| /REG\_W01/RestrictedSearchAttribute | 'title'  'object\_name' | The configuration control if the Restricted level search on the Folder name is performed on the Folder 'title' or on the 'object\_name' | Yes |

## Business Layer Messaging

The following messages may be thrown to the calling component throughout the processing of this component:

| Message Id | Message | Situation |
| --- | --- | --- |
| 94352 | You do not have a registration to the Head Client and the unrestricted Subclients, Jobs and/or Projects within the Head Client so you cannot perform the requested action. If you require a registration to the Head Client, please contact an approver for the Head Client to be granted access. |  |
| 94351 | You are required to have a registration to the Head Client in order to be able to #P1#. If you require a registration to the Head Client, please contact an approver for the Head Client to be granted access. |  |
| Constant | Removed Registration as registration denied. | Message used in logging of Registration task. Logged when expiring a Registration. |
| Constant | Removed Registration as expiry date met. | Message used in logging of Registration task. Logged when system expires a Registration due to expiry date. |
| Constant | User has had delegated approver status removed. | Message used in logging of Registration task. Logged when removing a Delegated Approver. |
| Constant | User has had auto approver status removed. | Message used in logging of Registration task. Logged when removing an Auto Approver. |
| Constant | User removed as part of client import. | Message used in logging of Registration task. Logged when removing a Registration due to Client Import . |
| Constant | Non Expiring Non Confidential Registration Removed. | Message used in logging of Registration task. Logged when removing an Unconfidential AutoApprover from the Restricted parent or Head Client. |
| Constant | Removing nominated approver access. | Message used in logging of Registration task. Logged when removing a Nominated Approver. |
| Constant | Due to Marking folder confidential. | Message used in logging of Registration task. Logged when adding/modifying/removing a Registration when marking Restricted. |
| Constant | Due to Unmarking folder confidential. | Message used in logging of Registration task. Logged when adding/modifying/removing a Registration when unmarking Restricted. |
| Constant | Added as Delegated Approver. | Message used in logging of Registration task. Logged when adding a Delegated Approver. |
| Constant | Added as Auto Approver. | Message used in logging of Registration task. Logged when adding an Auto Approver. |
| Constant | Copied Registration from <Folder Id>. | Message used in logging of Registration task. Logged when copying a Registration. |
| Constant | Added non-expiry registration for auto-approver of a sub folder. | Message used in logging of Registration task. Logged when adding a Non-Expiry Registration. |
| Constant | Nominated delegated approver access granted. | Message used in logging of Registration task. Logged when adding a Nominated Delegated Approver. |
| 65002 | You do not have the required permission to set/unset Product Specific Access. | If the user does not have the required permissions to set or unset Product Specific Access. |
| 65003 | Product Specific Access cannot be enabled. | If attempting to Set Product Specific Access, and the Product that the Job us under has not been configured to use Product Specific Groups or the Client the Job is under has been configured not to allow Product Specific Groups or Product Specific Access is disabled. |
| 65004 | Cannot set Product Specific Access, as the Job already has Product Specific Access set. | If the user attempts to Set Product Specific on a Job that already has it set. |
| 65005 | Cannot unset Product Specific Access, as the Job does not have Product Specific Access set. | If the user attempts to Unset Product Specific on a Job that does not have it set. |
| 65050 | You do not have permission to modify Clients within the Product Specific Access Exclusion List. | Returned when the user is not a member of one of the groups contained within the configuration item ‘/REG\_W11/AuthorisedGroups’ |

## System Requirements Checklist

The following DMS 6.7 SER requirements are satisfied or partly satisfied by this component:

|  |  |  |
| --- | --- | --- |
| Req Id | Description | Comments |
| SER497 | Evaluate Performance of Major Security Registration Queries. | To improve perfomace the format of the Registration queries responsible for the below function are updated:   1. Query to populate Registration information on Manage Registration search window for both normal user and security admin. 2. Query to populate Registration information on My Registration window. 3. Query to populate Registration information on Add/Remove Delegated Approver window.   The changes are only technical and no existing Business rules are impacted. |

## Design Decisions

1. Using Registration groups, security is maintained simply and without complex processing.
2. Update of Approvers after the change in object properties to be done in real-time but asynchronously.
3. The current Product specific security solution is intended to be an interim solution until web services can be used to perform the required functionality fully within the Registrations model. This is to meet the business need of granting users temporary access to a Job so they can carry out required Tax Compliance/IAS style work. This functionality is intended to allow certain groups of users access to all Jobs under a given Product across all Clients within the DMS. In particular it is intended for Tax Compliance and IAS Product work, where there is a large centralised team that carries out this work across all Clients within the DMS. This work is generally high volume, but taking only a short period of time to complete (2 to 4hrs), and the overhead of managing Security manually would be unmanageable from a business perspective.
4. It will not be possible for end users (Registration Approvers) to manage Registrations for Product Specific Access. These will be managed by REG\_B05. This has been done to simplify the logic required for Product Restricted Access which is intended as a temporary step to assist a number of territories with moving to the Registration model. Only Registration Admins will be able to view Product Restricted Access Registrations, this is due to the fact that the Registrations are added against a Product so they can be shared within the DMS.
5. Non-UI components will be responsible for setting/unsetting the Product Specific Access value for Jobs but will call the verification methods to ensure that the change can be made.
6. A custom table will be used to store the Client Code of Clients that will not use Product Specific Access.
7. Who can “Nominate Delegated Approvers” (see section ) will be governed by configuration. The UI component REG\_W06 – Nominated Delegated Approver Window will enforce only the configured users can modify Nominated Delegate Approvers.
8. The ability to either assign a default access time Registration or delete the Approver Registrations when a folder being marked as Restricted will be governed by the configuration.

# Technical Design

## Overview

The following diagram is of the various technical components which made up this component.

TlsRegististration

-getInformationAboutRegistrations

-updateRegistrations

WDK COMPONENT

WEB-SERVICE INTERFACE

DESKTOP CLIENT COMPONENT

External Calling presentation components (Out of scope)

TLSRegistrationImpl

Business Object Framework

Server Method (for updates)

ManageRegistrations

SvSecurityLib

RegistrationLib

RegistrationLibImpl

AddRegistration

RegistrationApproverLib

ConfidentialLib

RemoveRegistration

RegistrationLog

Core Classes that Update Registration Information

Other Server Methods

Commands

ModifyDefaultGroup

CreateDefaultGroupReg

ModifyApproverAccess

UpdateProductGroupReg

SetUnsetProductSpecificAccess

UpdateProductSpecificAccessExclusionCommand

The SBO component is implemented by three Java classes. One for the interface, one as the implementation of the interface, and the final one to execute queries to get values that will be returned to calling components. These classes will be located in the com.pwcglobal.tls.delta.shared.common.sbo.tlsregistrationservice package.

| Component | Type | Custom | Description |
| --- | --- | --- | --- |
| TlsRegistration.java | Java Interface Class | Yes | This is the Java interface class that contains all public method declarations to the implementation class. |
| TlsRegistrationsImpl.java | Java Implementation Class | Yes | This is the Java implementation class that contains code that overload existing Documentum DfService methods. |
| TlsRegQuery.java | Java Class | Yes | This contains all the logic to carry out queries and return the results for the get methods within TlsRegistrationsImpl. |

The server component is implemented by nine Java classes.

These classes will be located in the com.pwcglobal.tls.delta.server.registration package.

| Component | Type | Custom | Description |
| --- | --- | --- | --- |
| ManageRegistrations | Java Class | Yes | This class holds the method wrapper to call RegistrationsLib. |
| RegistrationLib | Java Interface Class | Yes | This is the Java interface class that contains all public method declarations to the implementation class. |
| RegistrationLibImpl.java | Java Implementation Class | Yes | This is the Java implementation class of RegistrationLib which will call the correct Java class to carry out the server side operations as a super user. |
| AddRegistration.java | Java Class | Yes | This class holds the logic to create Registrations. |
| ConfidentialLib.java | Java Class | Yes | This class holds the logic to mark and unmark folders as Confidential. |
| RegsitrationApproverLib.java | Java Class | Yes | This class holds the logic to set the Auto Approvers for folders. |
| RemoveRegistration.java | Java Class | Yes | This class holds the logic to expire or remove Registrations. |
| RegistrationLog.java | Java Class | Yes | This class contains the logic to update the Registration Log. |
| ManageNominatedApproversJob.java | Java Class | Yes | Handles the creation and expiration of nominated delegated approver registrations. |

Command objects inherit from abstract class Command created as part of the com.pwcglobal.tls.delta.shared.common.commands package. Transactional logic for these components is contained with the execute method. This component will implement the following two commands.

| Component | Type | Custom | Description |
| --- | --- | --- | --- |
| ModifyDefaultGroupCommand | Command | Yes | The ModifyDefaultGroupCommand will update the default group assigned to a folder. |
| CreateDefaultGroupRegCommand | Command | Yes | The CreateDefaultGroupRegCommand will create registrations for all users currently in the default group assigned to a folder. |
| ModifyApproverAccessCommand | Command | Yes | The ModifyApproverAccessCommand will update the Registration Approver Access setting for Clients, Jobs and Projects as well as update the GUID of who modified the setting. |
| UpdateProductGroupRegCommand | Command | Yes | The UpdateProductGroupRegCommand will update the Registrations for Product Specific Access to match the members in the configured group. |
| SetUnsetProductSpecificAccessCommand | Command | Yes | Will Set or Unset the group access setting for a list of Jobs. |
| UpdateProductSpecificAccessExclusionCommand | Command | Yes | Adds and removes Clients from the Product Specific Access Exclusion List. |

## Queries

Note: Only the queries modified from the US design for the Global implementation have been inclided. Existing queries may need to be modified during construction to use the t\_ultimate\_head\_client\_code attribute to cater for Global’s multiple level Subclients.

| ID | Query | Referenced |
| --- | --- | --- |
| 1 | SELECT t\_ultimate\_head\_client\_code FROM tls\_client WHERE t\_client\_code = A | Used to get the Head Client of an object in getHeadClient method. |
| 2 | UPDATE tls\_client  SET t\_client\_default\_group = ‘<group\_name>’  WHERE r\_object\_id = ‘<object\_id>’ | Used in ModifyDefaultGroupCommand to update the default group of a Client. |
| 3 | UPDATE ‘<object\_type>’  SET t\_create\_reg = ‘<reg setting>’  SET t\_create\_reg\_user = ‘<user\_name>’  WHERE r\_object\_id = ‘<object\_id>’ | Used in ModifyRegApproverAccessCommand to update the Registration Approver Access setting. |
| 4 | SELECT r\_object\_id FROM tls\_product\_item WHERE t\_product\_code = ‘<Product Code>’ | Used in UpdateProductGroupRegCommand to retrieve the Product Item Id for a given Product Code. The Product Item Id is used to identify the Registration group that requires updating. |
| 5 | UPDATE tls\_job SET t\_product\_group\_access = ‘<access\_enabled>’ WHERE r\_object\_id = ‘<object\_id>’ | Used in SetUnsetProductSpecificAccessCommand to update a Job’s Product Specific Access setting. |
| 6 | SELECT prd.t\_product\_code AS product\_code, cli.r\_object\_id AS client\_id FROM tls\_client cli, tls\_product\_item prd, tls\_job job WHERE cli.r\_object\_id = job.t\_client\_id AND prd.r\_object\_id = job.t\_product\_id AND job.r\_object\_id = '<object\_id>' | Used in isAllowedProductSpecificAccess to get the Product and Client Codes of a Job to verify with configuration values. |
| 7 | SELECT t\_product\_group\_access FROM tls\_job WHERE r\_object\_id = ‘<object\_id>’ | Used in SetUnsetProductSpecificAccessCommand to get the current group access setting for a Job. |
| 8 | INSERT INTO dm\_dbo. tls\_product\_specific\_exclusion values(‘<Client Object Id>’, ‘<Excluded Date>’, ‘<Excluded By>’) | **addClientsForProductSpecificAccessExclusion**  Used to Insert new entries into the tls\_product\_specific\_exclusion registered table to exclude Clients from Product Specific Access. |
| 9 | DELETE FROM dm\_dbo.tls\_product\_specific\_exclusion WHERE t\_excluded\_client = ‘<Client Id>’ | **removeClientsFromProductSpecificAccessExclusion**  Used to Remove existing entries from the tls\_product\_specific\_exclusion registered table to cancel Clients from exclusion of Product Specific Access. |
| 10 | SELECT r\_excluded\_client\_id FROM tls\_product\_specific\_exclusion | **addClientsForProductSpecificAccessExclusion removeClientsFromProductSpecificAccessExclusion**  Used to get the currently excluded Clients. |
| 11 | SELECT 1 FROM tls\_client WHERE r\_object\_id = ‘<object\_id>’ | **addClientsForProductSpecificAccessExclusion**  Used to validate that the Client Object Id that is to be added to the tls\_product\_specific\_exclusion table is an existing Client in the DMS. |
| 12 | **ORACLE:**  SELECT r\_object\_id t\_product\_id, t\_product\_group\_code||t\_product\_code||' - '||t\_description t\_product\_name  FROM tls\_product\_item\_s  **SQL:**  SELECT r\_object\_id AS "t\_product\_id", CONCAT(t\_product\_group\_code, t\_product\_code, ' - ', t\_description) As "t\_product\_name"  FROM tls\_product\_item\_s | Created as a Database VIEW and registered as tls\_product\_name to be used by the RegQuery class. |
| 13 | SELECT t\_product\_name FROM tls\_product\_name WHERE t\_product\_id = ‘<object\_id>’ | Used in RegQuery to return a Product name which is a concatenation of the Product Group Code, Product Code and Product Description of a Product. The Product name will be used as the location for Product Specific Access Registrations. |
| 14 | SELECT 1 FROM tls\_prod\_access\_exclusion WHERE t\_excluded\_client\_id = '<t\_client\_id>' | Used to verify if a Client has been excluded from Product Specific Access. |
| 15 | SELECT r.t\_folder\_id as objectid,  'Head Client' as object\_type  FROM tls\_registration r  WHERE r.t\_user\_name = '[APPROVER\_USER\_GUID]'  AND r.t\_approver\_status = 'AUTO'  AND EXISTS  (  SELECT 1  FROM tls\_client c  WHERE c.t\_confidential = False  AND c.r\_object\_id = r.t\_folder\_id  )  UNION  SELECT r.t\_folder\_id as objectid,  'Subclient' as object\_type  FROM tls\_registration r  WHERE r.t\_user\_name = '[APPROVER\_USER\_GUID]'  AND r.t\_approver\_status = 'AUTO'  AND EXISTS  (  SELECT 1  FROM tls\_client c  WHERE c.t\_confidential = True  AND c.r\_object\_id = r.t\_folder\_id  )  UNION  SELECT r.t\_folder\_id as objectid,  'Job' as object\_type  FROM tls\_registration r  WHERE r.t\_user\_name = '[APPROVER\_USER\_GUID]'  AND r.t\_approver\_status = 'AUTO'  AND EXISTS  (  SELECT 1  FROM tls\_job j  WHERE j.t\_confidential = True  AND j.r\_object\_id = r.t\_folder\_id  )  UNION  SELECT r.t\_folder\_id as objectid,  'Project' as object\_type  FROM tls\_registration r  WHERE r.t\_user\_name = '[APPROVER\_USER\_GUID]'  AND r.t\_approver\_status = 'AUTO'  AND EXISTS  (  SELECT 1  FROM tls\_project p  WHERE p.t\_confidential = True  AND p.r\_object\_id = r.t\_folder\_id  ) | Used to get the folder type and path of all folders for which the specified user is an Auto Approver. |
| 16 | SELECT r\_object\_id from tls\_registration where t\_approver\_status ='DELEGATED' and t\_folder\_id= <object id> | isDelegatedApprover  Used to check if the user is a delegated approver for the folder |
| 17 | SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  fp.r\_folder\_path,  'Head Client' AS object\_type,  '1' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  tls\_user u,  dm\_dbo.tls\_folder\_path fp  WHERE t\_status LIKE 'ACTIVE%'  AND r.t\_folder\_id = fp.r\_object\_id  AND r.t\_user\_name = u.user\_name  AND <One of the following three additional search values.>  //If Client Code is not null and username is null  {  AND t\_folder\_id = '[HC\_OBJECT\_ID]'  }  //Username is not null but Client Code is null  {  AND t\_user\_name = '[SEARCH\_USER\_GUID]'  AND EXISTS  (  SELECT 1  FROM tls\_client c  WHERE c.t\_confidential = False  AND c.r\_object\_id = r.t\_folder\_id  )  }  //Head Client and Username are NOT NULL  {  AND t\_folder\_id = '[HC\_OBJECT\_ID]'  AND t\_user\_name = '[SEARCH\_USER\_GUID]'  } | The DQL to fetch Registration information for a Head Client when the logged in user is a Security Admin.  Note: Query displayed is only for the most common scenario, does not include additional filter conditions that can be selected by a user. |
| 18 | SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  fp.r\_folder\_path,  'Subclient' AS object\_type,  '2' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  tls\_user u,  dm\_dbo.tls\_folder\_path fp  WHERE  t\_status LIKE 'ACTIVE%'  AND r.t\_folder\_id = fp.r\_object\_id  AND r.t\_user\_name = u.user\_name  AND <one of the following three additional search criterias>  //IF head Client code is not Null  {  EXISTS  (  SELECT 1  FROM tls\_client cl  WHERE cl.t\_confidential = True  AND cl.t\_ultimate\_head\_client\_code = '[HC\_CLIENT\_CODE]'  AND r.t\_folder\_id = cl.r\_object\_id  )    }  //IF Username is not null  {  t\_user\_name = '[SEARCH\_USER\_GUID]'  AND EXISTS  (  SELECT 1  FROM tls\_client c  WHERE t\_confidential = True  AND c.r\_object\_id = r.t\_folder\_id  )  }  //IF Username and Client Code is not null  {  EXISTS  (  SELECT 1  FROM tls\_client cl  WHERE cl.t\_confidential = True  AND cl.t\_ultimate\_head\_client\_code = '[HC\_CLIENT\_CODE]'  AND r.t\_folder\_id = cl.r\_object\_id  )  AND t\_user\_name ='[SEARCH\_USER\_GUID]'  } | The DQL to fetch Registration information for a Sub Client when the logged in user is a Security Admin.  Note: Query displayed is only for the most common scenario, does not include additional filter conditions that can be selected by a user. |
| 19 | SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  fp.r\_folder\_path,  'Job' AS object\_type,  '3' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  tls\_user u,  dm\_dbo.tls\_folder\_path fp  WHERE t\_status LIKE 'ACTIVE%'  AND r.t\_folder\_id = fp.r\_object\_id  AND r.t\_user\_name = u.user\_name  AND <One of the following three additional search values.>  //IF head Client Code is not null  {  EXISTS  (  SELECT 1  FROM tls\_job jb  WHERE jb.t\_confidential = True  AND jb.t\_client\_id = '[HC\_OBJECT\_ID]'  AND r.t\_folder\_id = jb.r\_object\_id  )  }  //IF username and Client Code is not null  {  EXISTS  (  SELECT 1  FROM tls\_job jb  WHERE jb.t\_confidential = True  AND jb.t\_client\_id = '[HC\_OBJECT\_ID]'  AND r.t\_folder\_id = jb.r\_object\_id  )  //If username is not null  AND t\_user\_name = '[SEARCH\_USER\_GUID]'  } | The DQL to fetch Registration information for a Job under Head Client when the logged in user is a Security Admin.  Note: Query displayed is only for the most common scenario, does not include additional filter conditions that can be selected by a user. |
| 20 | SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  fp.r\_folder\_path,  'Project' AS object\_type,  '4' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  tls\_user u,  dm\_dbo.tls\_folder\_path fp  WHERE  t\_status LIKE 'ACTIVE%'  AND r.t\_folder\_id = fp.r\_object\_id  AND r.t\_user\_name = u.user\_name  AND <One of the following three additional search values.>    //IF head Client code is not null  {  EXISTS  (  SELECT 1  FROM tls\_project pr  WHERE pr.t\_confidential = True  AND pr.t\_client\_id = '[HC\_OBJECT\_ID]'  AND r.t\_folder\_id = pr.r\_object\_id  )  }  //IF head Client code and username is not null  {  WHERE EXISTS  (  SELECT 1  FROM tls\_project pr  WHERE pr.t\_confidential = True  AND pr.t\_client\_id = '[HC\_OBJECT\_ID]'  AND r.t\_folder\_id = pr.r\_object\_id  )  //if username not null  AND t\_user\_name = '[SEARCH\_USER\_GUID]'  } | The DQL to fetch Registration information for a Project under Head Client when the logged in user is a Security Admin.  Note: Query displayed is only for the most common scenario, does not include additional filter conditions that can be selected by a user. |
| 21 | SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  f.r\_folder\_path,  'Product' AS object\_type,  '5' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  dm\_dbo.tls\_product\_name\_view f,  tls\_user u  WHERE t\_status LIKE 'ACTIVE%'  AND r.t\_folder\_id = f.t\_product\_id  AND r.t\_user\_name = u.user\_name  AND <One of the following additional search values.>  //IF headclient code is NOT NULL  {  AND EXISTS  (  SELECT 1  FROM tls\_job j  WHERE t\_product\_group\_access = True  AND f.t\_product\_id = j.t\_product\_id  AND j.t\_client\_id = '[HC\_OBJECT\_ID]'  )  {  AND t\_user\_name =  [SEARCH\_USER\_GUID]'  }    } | The DQL to fetch Product specific Registration under a Head Client when the logged in user is a Security Admin.  Note: Query displayed is only for the most common scenario, does not include additional filter conditions that can be selected by a user. |
| 22 | SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  fp.r\_folder\_path,  'Job' AS object\_type,  '3' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  tls\_user u,  dm\_dbo.tls\_folder\_path fp  WHERE EXISTS  (  SELECT 1  FROM tls\_job jb  WHERE r.t\_folder\_id = jb.r\_object\_id  AND jb.t\_confidential = True  AND <One of the following additional three search values.>  //IF head Client code is not null and the user name is null  {  AND EXISTS  (  SELECT 1  FROM tls\_client cl  WHERE cl.t\_ultimate\_head\_client\_code = '[HC\_CLIENT\_CODE]'  AND jb.t\_client\_id = cl.r\_object\_id  )  }    //IF Head Client code and user name is not null  {  AND EXISTS  (  SELECT 1  FROM tls\_client cl  WHERE cl.t\_ultimate\_head\_client\_code = '[HC\_CLIENT\_CODE]'  AND jb.t\_client\_id = cl.r\_object\_id  )  //If username is not null  AND t\_user\_name = [SEARCH\_USER\_GUID]'  }  }  AND t\_status LIKE 'ACTIVE%'  AND r.t\_folder\_id = fp.r\_object\_id  AND r.t\_user\_name = u.user\_name | The DQL to fetch Job Registration under a Sub Client when the logged in user is a Security Admin.  Note: Query displayed is only for the most common scenario, does not include additional filter conditions that can be selected by a user. |
| 23 | SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  fp.r\_folder\_path,  'Project' AS object\_type,  '4' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  tls\_user u,  dm\_dbo.tls\_folder\_path fp  WHERE  EXISTS  (  SELECT 1  FROM tls\_project p  WHERE r.t\_folder\_id = p.r\_object\_id  AND p.t\_confidential = True  AND <One of the following additional three search values.>  //IF Client Code is not null  {  AND EXISTS  (  SELECT 1  FROM tls\_client cl  WHERE cl.t\_ultimate\_head\_client\_code = '[HC\_CLIENT\_CODE]'  AND p.t\_client\_id = cl.r\_object\_id  )  }    //IF username and Client code is  not NULL  {  AND EXISTS  (  SELECT 1  FROM tls\_client cl  WHERE cl.t\_ultimate\_head\_client\_code = '[HC\_CLIENT\_CODE]'  AND p.t\_client\_id = cl.r\_object\_id  )  //If username is not null  AND t\_user\_name = '[SEARCH\_USER\_GUID]'  }  )  AND t\_status LIKE 'ACTIVE%'  AND r.t\_folder\_id = fp.r\_object\_id  AND r.t\_user\_name = u.user\_name | The DQL to fetch Project Registration under a Sub Client when the logged in user is a Security Admin.  Note: Query displayed is only for the most common scenario, does not include additional filter conditions that can be selected by a user. |
| 24 | SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  f.r\_folder\_path,  'Product' AS object\_type,  '5' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  dm\_dbo.tls\_product\_name\_view f,  tls\_user u  WHERE t\_status LIKE 'ACTIVE%'  AND r.t\_folder\_id = f.t\_product\_id  AND r.t\_user\_name = u.user\_name  AND<One of the following additional search values.>  //IF Client Code is not null  {  EXISTS  (  SELECT 1  FROM tls\_job j  WHERE t\_product\_group\_access = True  AND f.t\_product\_id = j.t\_product\_id  AND EXISTS  (  SELECT 1  FROM tls\_client cl  WHERE cl.t\_ultimate\_head\_client\_code = '[HC\_CLIENT\_CODE]'  AND j.t\_client\_id = cl.r\_object\_id  )  )  //IF username is not null  {  AND t\_user\_name = '[SEARCH\_USER\_GUID]'  }  } | The DQL to fetch Product Specific Registration under a Sub Client when the logged in user is a Security Admin.  Note: Query displayed is only for the most common scenario, does not include additional filter conditions that can be selected by a user. |
| 25 | SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  fp.r\_folder\_path,  'Head Client' AS object\_type,  '1' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  tls\_user u,  dm\_dbo.tls\_folder\_path fp  WHERE  EXISTS  (  SELECT 1  FROM tls\_registration reg2,  tls\_client cl  WHERE reg2.t\_user\_name = '[APPROVER\_USER\_GUID]'  AND  ( reg2.t\_approver\_status != 'NONE'  OR cl.t\_create\_reg != '[CREATE\_REG\_APPROVER]'  )  AND r.t\_folder\_id = reg2.t\_folder\_id  AND cl.r\_object\_id = reg2.t\_folder\_id  )  AND <One of the following additional three search values.>    //IF Client Code and is not Null and username is null  {  AND t\_folder\_id = '[HC\_OBJECT\_ID]'  }  ///Manage user  //IF Client code is NULL and user name is not NULL  {  AND t\_user\_name = '[SEARCH\_USER\_GUID]'  AND EXISTS  ( SELECT 1  FROM tls\_client c  WHERE c.t\_confidential = False  AND c.r\_object\_id = r.t\_folder\_id  )  }  ///Manage hc user  //IF Client Code and user name are NOT NULL  {    AND t\_folder\_id = '[HC\_OBJECT\_ID]'  AND t\_user\_name = '[SEARCH\_USER\_GUID]'  }  AND t\_status LIKE 'ACTIVE%'  AND r.t\_folder\_id = fp.r\_object\_id  AND r.t\_user\_name = u.user\_name | The DQL to fetch Head Client Registration when the logged in user is an Approver.  Note: Query displayed is only for the most common scenario, does not include additional filter conditions that can be selected by a user. |
| 26 | SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  fp.r\_folder\_path,  'Subclient' AS object\_type,  '2' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  tls\_user u,  dm\_dbo.tls\_folder\_path fp  WHERE EXISTS  (  SELECT 1  FROM tls\_registration reg2,  tls\_client cl  WHERE reg2.t\_user\_name = '[APPROVER\_USER\_GUID]'  AND  (  reg2.t\_approver\_status != 'NONE'  OR cl.t\_create\_reg != '[CREATE\_REG\_APPROVER]'  )  AND r.t\_folder\_id = reg2.t\_folder\_id  AND cl.r\_object\_id = reg2.t\_folder\_id  )  AND <One of the following additional three search values.>  //IF Head Client Code not null and username is null  {  AND EXISTS  (  SELECT 1  FROM tls\_client c  WHERE r.t\_folder\_id = c.r\_object\_id  AND c.t\_ultimate\_head\_client\_code = '[HC\_CLIENT\_CODE]'  )    }  //IF username is not null and Client Code and Client name is null  {  AND t\_user\_name = '[SEARCH\_USER\_GUID]'  AND EXISTS  (  SELECT 1  FROM tls\_client c  WHERE t\_confidential = True  AND c.r\_object\_id = r.t\_folder\_id  )  }  //IF Username is not null and Client code is not null  {  AND EXISTS  (  SELECT 1  FROM tls\_client c  WHERE r.t\_folder\_id = c.r\_object\_id  AND t\_ultimate\_head\_client\_code = '[HC\_CLIENT\_CODE]'  )  AND t\_user\_name = '[SEARCH\_USER\_GUID]'    }  AND t\_status LIKE 'ACTIVE%'  AND r.t\_user\_name = u.user\_name  AND r.t\_folder\_id = fp.r\_object\_id | The DQL to fetch Sub Client Registration when the logged in user is an Approver.  Note: Query displayed is only for the most common scenario, does not include additional filter conditions that can be selected by a user. |
| 27 | SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  fp.r\_folder\_path,  'Job' AS object\_type,  '3' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  tls\_user u,  dm\_dbo.tls\_folder\_path fp  WHERE EXISTS  (  SELECT 1  FROM tls\_registration reg2,  tls\_job jb  WHERE reg2.t\_user\_name = '[APPROVER\_USER\_GUID]'  AND  (  reg2.t\_approver\_status != 'NONE'  OR jb.t\_create\_reg != '[CREATE\_REG\_APPROVER]'  )  AND r.t\_folder\_id = reg2.t\_folder\_id  AND jb.r\_object\_id = reg2.t\_folder\_id  )  AND <One of the following additional search values.>  //IF Head Client Code is not null  {  AND EXISTS  (  SELECT 1  FROM tls\_job j  WHERE r.t\_folder\_id = j.r\_object\_id  AND j.t\_client\_id = '[HC\_OBJECT\_ID]'  )  }  //IF username and Client code is not null  {  AND t\_user\_name = '[SEARCH\_USER\_GUID]'  AND EXISTS  (  SELECT 1  FROM tls\_job j  WHERE r.t\_folder\_id = j.r\_object\_id  AND j.t\_client\_id = '[HC\_OBJECT\_ID]'  )  }  AND t\_status LIKE 'ACTIVE%'  AND r.t\_user\_name = u.user\_name  AND r.t\_folder\_id = fp.r\_object\_id | The DQL to fetch Job Registration under Head Client when the logged in user is an Approver.  Note: Query displayed is only for the most common scenario, does not include additional filter conditions that can be selected by a user. |
| 28 | SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  fp.r\_folder\_path,  'Project' AS object\_type,  '4' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  tls\_user u,  dm\_dbo.tls\_folder\_path fp  WHERE EXISTS  (  SELECT 1  FROM tls\_registration reg2,  tls\_project pr  WHERE reg2.t\_user\_name = '[APPROVER\_USER\_GUID]'  AND  (  reg2.t\_approver\_status != 'NONE'  OR pr.t\_create\_reg != '[CREATE\_REG\_APPROVER]'  )  AND r.t\_folder\_id = reg2.t\_folder\_id  AND pr.r\_object\_id = reg2.t\_folder\_id  )  AND <One of the following additional search values.>  //IF Head Client Code is not null  {  AND EXISTS  (  SELECT 1  FROM tls\_project p  WHERE r.t\_folder\_id = p.r\_object\_id  AND t\_client\_id = '[HC\_OBJECT\_ID]'  )  }  //IF username and Head Client Code is not null  {  AND EXISTS  (  SELECT 1  FROM tls\_project p  WHERE r.t\_folder\_id = p.r\_object\_id  AND t\_client\_id = '[HC\_OBJECT\_ID]'  )  AND t\_user\_name = '[SEARCH\_USER\_GUID]'  }  AND t\_status LIKE 'ACTIVE%'  AND r.t\_user\_name = u.user\_name  AND r.t\_folder\_id = fp.r\_object\_id | The DQL to fetch Project Registration under Head Client when the logged in user is an Approver.  Note: Query displayed is only for the most common scenario, does not include additional filter conditions that can be selected by a user. |
| 29 | SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  fp.r\_folder\_path,  'Job' AS object\_type,  '3' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  tls\_user u,  dm\_dbo.tls\_folder\_path fp  WHERE EXISTS  (  SELECT 1  FROM tls\_registration reg2,  tls\_job jb  WHERE reg2.t\_user\_name = '[APPROVER\_USER\_GUID]'  AND  (  reg2.t\_approver\_status != 'NONE'  OR jb.t\_create\_reg != '[CREATE\_REG\_APPROVER]'  )  AND r.t\_folder\_id = reg2.t\_folder\_id  AND jb.r\_object\_id = reg2.t\_folder\_id  )  AND <One of the following additional search values.>  //IF head Client Code is not null  {  AND EXISTS  (  SELECT 1  FROM tls\_job j  WHERE r.t\_folder\_id = j.r\_object\_id  AND t\_confidential = true  AND EXISTS  (  SELECT 1  FROM tls\_client cl  WHERE t\_ultimate\_head\_client\_code = '[HC\_CLIENT\_CODE]'  AND j.t\_client\_id = cl.r\_object\_id  )  )  //If username is not null  {  AND EXISTS  (  SELECT 1  FROM tls\_job j  WHERE r.t\_folder\_id = j.r\_object\_id  AND t\_confidential = true  AND t\_user\_name = '[SEARCH\_USER\_GUID]'  )  }  //username and Client Code is not null  {  AND EXISTS  (  SELECT 1  FROM tls\_job j  WHERE r.t\_folder\_id = j.r\_object\_id  AND t\_confidential = true  AND EXISTS  (  SELECT 1  FROM tls\_client cl  WHERE t\_ultimate\_head\_client\_code = '[HC\_CLIENT\_CODE]'  AND j.t\_client\_id = cl.r\_object\_id  )  )  AND t\_user\_name = '[SEARCH\_USER\_GUID]'  }  AND t\_status LIKE 'ACTIVE%'  AND r.t\_user\_name = u.user\_name  AND r.t\_folder\_id = fp.r\_object\_id  } | The DQL to fetch Job Registration under Sub Client when the logged in user is an Approver.  Note: Query displayed is only for the most common scenario, does not include additional filter conditions that can be selected by a user. |
| 30 | SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  fp.r\_folder\_path,  'Project' AS object\_type,  '4' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  tls\_user u,  dm\_dbo.tls\_folder\_path fp  WHERE EXISTS  (  SELECT 1  FROM tls\_registration reg2,  tls\_project pr  WHERE reg2.t\_user\_name = '[APPROVER\_USER\_GUID]'  AND  (  reg2.t\_approver\_status != 'NONE'  OR pr.t\_create\_reg != '[CREATE\_REG\_APPROVER]'  )  AND r.t\_folder\_id = reg2.t\_folder\_id  AND pr.r\_object\_id = reg2.t\_folder\_id  )  AND <One of the following additional search values.>  //IF Client Code is not Null  {  AND EXISTS  (  SELECT 1  FROM tls\_project p  WHERE r.t\_folder\_id = p.r\_object\_id  AND EXISTS  (  SELECT 1  FROM tls\_client cl  WHERE t\_ultimate\_head\_client\_code = '[HC\_CLIENT\_CODE]'  AND p.t\_client\_id = cl.r\_object\_id  )  )  } | The DQL to fetch Project Registration under Sub Client when the logged in user is an Approver.  Note: Query displayed is only for the most common scenario, does not include additional filter conditions that can be selected by a user. |
| 31 | SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  fp.r\_folder\_path,  'Head Client' AS object\_type,  '1' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  tls\_user u,  dm\_dbo.tls\_folder\_path fp  WHERE t\_user\_name = '[SEARCH\_USER\_GUID]'  AND t\_status LIKE 'ACTIVE%'  AND r.t\_folder\_id = fp.r\_object\_id  AND r.t\_user\_name = u.user\_name  AND EXISTS  (  SELECT 1  FROM tls\_client c  WHERE c.t\_confidential = False  AND c.r\_object\_id = r.t\_folder\_id  )  UNION  SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  fp.r\_folder\_path,  'Subclient' AS object\_type,  '2' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  tls\_user u,  dm\_dbo.tls\_folder\_path fp  WHERE t\_user\_name = '[SEARCH\_USER\_GUID]'  AND t\_status LIKE 'ACTIVE%'  AND r.t\_folder\_id = fp.r\_object\_id  AND r.t\_user\_name = u.user\_name  AND EXISTS  (  SELECT 1  FROM tls\_client c  WHERE c.t\_confidential = True  AND c.r\_object\_id = r.t\_folder\_id  )  UNION  SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  fp.r\_folder\_path,  'Job' AS object\_type,  '3' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  tls\_user u,  dm\_dbo.tls\_folder\_path fp  WHERE t\_user\_name = '[SEARCH\_USER\_GUID]'  AND t\_status LIKE 'ACTIVE%'  AND r.t\_folder\_id = fp.r\_object\_id  AND r.t\_user\_name = u.user\_name  AND EXISTS  (  SELECT 1  FROM tls\_job j  WHERE j.t\_confidential = True  AND j.r\_object\_id = r.t\_folder\_id  )  UNION  SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  fp.r\_folder\_path,  'Project' AS object\_type,  '4' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  tls\_user u,  dm\_dbo.tls\_folder\_path fp  WHERE t\_user\_name = '[SEARCH\_USER\_GUID]'  AND t\_status LIKE 'ACTIVE%'  AND r.t\_folder\_id = fp.r\_object\_id  AND r.t\_user\_name = u.user\_name  AND EXISTS  (  SELECT 1  FROM tls\_project p  WHERE p.t\_confidential = True  AND p.r\_object\_id = r.t\_folder\_id  )  UNION  SELECT r\_object\_id,  t\_folder\_id,  t\_user\_name,  t\_approver,  t\_status,  t\_expiry\_date,  t\_approver\_status,  f.r\_folder\_path,  'Product' AS object\_type,  '5' AS type\_sort,  u.t\_full\_name  FROM tls\_registration r,  dm\_dbo.tls\_product\_name\_view f,  tls\_user u  WHERE t\_user\_name = '[SEARCH\_USER\_GUID]'  AND t\_status LIKE 'ACTIVE%'  AND r.t\_folder\_id = f.t\_product\_id  AND r.t\_user\_name = u.user\_name | The Query to populate Registration values on My Registration window.  Note: Query displayed is only for the most common scenario, does not include additional filter conditions that can be selected by a user. |

## External Dependencies

|  |  |  |  |
| --- | --- | --- | --- |
| This component | Component Id | Name | Description |
| Calls | CFG\_C04 | Security & ACLs |  |
| Calls | COM\_B69 | Security ACL Commands |  |
| Is called by | REG\_W02 | Add Registration | This component creates Registration submissions which are processed. |
| Is called by | REG\_W01 | My Registrations | This component displays Registration information |
| Is called by | REG\_W03 | Manage Registrations | This component manages Registration information. |
| Is called by | REG\_W06 | Nominate Delegated Approver Window | To set nominated delegated approver details. |
| Is called by | CAT\_W08 | Job Properties | This component is used to update Job information |
| Is called by | CAT\_B83, B84 | TLS Client Facade and Commands | This component contains the commands relevant for Clients. It calls this component when reassigning a pending Client. |
| Is called by | CAT\_B80, B81 | TLS Job Facade and Commands | This component contains the commands relevant for Jobs. It calls this component to update registrations when creating or moving a Job. |
| Is called by | PRO\_B71, B72 | TLS Project Facade and Commands | This component contains the commands relevant for Projects. It calls this component to update registrations when creating, copying, and rolling over a Project. |
| Is called by | PRO\_W01 | Project Properties | This component updates a Project |
| Is called by | CAT\_B04 | Import Client Commands | This component imports Clients into the DMS. |
| Is called by | CAT\_B07 | Import Job Commands | This component imports Jobs into the DMS. |
| Is called by | REG\_W05 | Add Delegated Approver | This components allows a batch add delegated approver. |
| Is called by | REG\_B04 | Manage Nominated Delegated Approvers Job | Called to add and remove Nominated Delegateed Approvers. |
| Is called by | INF\_W28 | Client Find | Called to retrieve the registration Approvers for a Client. |
| ~~Is called by~~ | ~~ADM\_U04~~ | ~~Synch Registration Groups~~ | ~~Called to synch the registrations between archive and active docbases.~~ |
| ~~Is called by~~ | ~~ARC\_C05~~ | ~~TLS Generic Archive Command~~ | ~~Called to apply the security rules when the folders are archived.~~ |
| Is called by | REG\_W09 | Set Registration Approver Access | Called to verify if a user has access to modify the “Athorized to Create Registrations” setting. |
| Is called by | REG\_W08 | Modify Client Default Access Group | Called to verify if a user has access to modify the default access groups as well as called to modify the default access group. |
| Is called by | CAT\_W07 | Client Properties | This component updates a Client. |
| Is called by | CAT\_W08 | Job Properties | This component updates a Job |
| Is called by | REG\_W10 | Set-Unset Product Specific Access | Called to verify and set/unset the Product Specific Access command for list of Jobs. |
| Is called by | REG\_W11 | Manage Default Group Exclusions | Called to update the Clients within the Product Specific Access Exclusion List. |

## TlsRegistration

This interface class extends the IDfService interface. All common functionality related to the manipulation of registrations should be implemented in this SBO.

**General Construction Hints:**

Archive functionality will be commented out instead of removed to reduce work when implemented in a later release.

Auto Approvers for Client are configurable. US code will need to be modified to make use of configuration.

Look at creating a method to get Client Approvers based on configuration of DMS to use either Client Partner/Manager or Tax Client Partner/Manager.

### Superinterfaces

The superinterface for this SBO is:

1. IDfService

The superinterface of IDfService can be viewed via the DFC v5.2 API Specification.

This implementation class extends the DfService class and implements the IDfService interface. This class contains all fields and methods pertaining to this custom SBO.

## Custom objects

The following objects (No super-type) will need to be added for this component

TLS\_REGISTRATION

*Custom object.*

1. T\_FOLDER\_ID
2. T\_REQUESTER (The first user who submitted the registration, SYSTEM if it is system generated such as a copy Job operation, setting the Auto Approvers for folder or a folder being unmark confidential, or SYSTEM-NONEXPIRY in the case of the creation of a non-expiring non-Auto Approver registration. The requestor for Nominated Delegated Approvers will be SYSTEM-NOMINATED, this is to ensure that the Nominated Delegated Approver registration will not over write existing Auto Approver registrations.)
3. T\_USER\_NAME (The user who has the registration)
4. T\_APPROVER (The user who updated or denied the registration or SYSTEM if it is system generated such as a lapsed registration, or SYSTEM-NONEXPIRY if it is a non-expiring non-Auto Approver registration. The approver for Nominated Delegated Approvers will be the user listed within the ‘t\_approver\_user\_name’ field of the TLS\_NOMINATED\_APPROVER table, this is to ensure that the Nominated Delegated Approver registration will not over write existing Auto Approver registrations.)
5. T\_STATUS (Valid Statuses: ACTIVE, EXPIRED-LEFT, EXPIRED-LAPSED, EXPIRED-DENIED)
6. T\_EXPIRY\_DATE (Valid Dates are NULLDATE for non-expiring or the date that the registration should expire)
7. T\_APPROVER\_STATUS (Valid Statuses: NONE, DELEGATED,DELEGATED-NOMINATED,AUTO)

NOTE: The best way to work out whether or not a user has non-expiring non-Auto Approver registrations to a folder is look at the t\_expiry\_date which will be set to ‘nulldate’ and the t\_approver\_status which will be set to ‘NONE’ columns.

It is possible that the t\_approver for a non-expiring registration may be SYSTEM.

TLS\_REGISTRATION\_LOG

*Custom object.*

1. T\_FOLDER\_ID (The object id of the folder the registration is for.)
2. T\_PERFORMER (The user who performed the action.)
3. T\_USER\_NAME (The user who the registration is for. The field will be blank if the record if for a folder being Marked/Unmarked Restricted)
4. T\_ACTION (The name of the action that took place. The action will have one of the following values: ADD, REMOVE, EXTEND, UPDATE, DELETE, MARK RESTRICTED, UNMARK RESTRICTED).
5. T\_DATE (The date the registration was created/modified.)
6. T\_ACTION\_DETAILS (Any additional information about the action carried out, such as that the registration was deleted due to a folder being unmarked as confidential.)

TLS\_NOMINATED\_APPROVER

*Custom object.*

1. T\_APPROVER\_USER\_NAME (The GUID of the user nominating a user as a delegated approver.)
2. T\_NOMINATED\_USER\_NAME (The GUID of the user who is a nominated delegated approver for the folders where the approver is an Auto Approver.)
3. T\_DATE\_SET (The date that the user was nominated as a delegated approver.)

**Construction Hint:** The TLS\_NOMINATED\_APPROVER will need to be updated to include the t\_date\_set field, as required by SCR#532.

TLS\_PRODUCT\_SPECIFIC\_EXCLUSION

*Custom registered table.*

1. T\_EXCLUDED\_CLIENT\_ID (The r\_object\_id of the Client submitted for exclusion).
2. T\_EXCLUDED\_DATE (The date that the Client was submitted for exclusion).
3. T\_EXCLUDED\_BY (The GUID of the user who submitted the Client for exclusion).

**Construction Hint:** An index on the t\_exclided\_client\_id may assist with performance for calling components that check if a client has been added for exclusion in this table.

## TlsRegistrationImpl.java

This class contains the logic and method to service Registrations.

This is used to get Registration information such as Approvers and searching for Registration information.

It is also the entry point to update and create Registrations.

##### Public Methods

| Name | Input | Output | Description |
| --- | --- | --- | --- |
| AddDelegatedApprover | IdfList, String | IDfLISt | This will add a user to a number of folders |
| addDelegatedApprover | IDfId, String | None | Adds a user as Delegated Approver to a folder. |
| addDelegatedApprovers | IDfId, IDfList | None | Adds a list of users as Delegated Approvers to a folder. |
| addNominatedApprovers | IDfList | None | This will add Nominated Delegateed Approvers for the currently logged in user. |
| addRegistration | IDfId, String, Integer | None | Creates a Registration to a folder for a user for a time period. |
| addRegsForUsers | IDfId, IDfList, Integer | None | Creates Registrations for a list of users to a folder for a time period. |
| addRegsToFolders | IDfList, String, Integer | None | Creates Registrations for a user to a list of folders for a time period. |
| addRegsToFoldersForUsers | IDfList, IDfList, Integer | None | Creates Registrations for a list of users to a list of folders for a time period. |
| addRegsToFoldersForUsers | IDfList, IDfList, Integer, Boolean | None | Creates Registrations for a list of users to a list of folders for a time period. The Boolean argument will determine whether registrations should be added to the Head Client as well. |
| batchresetRegistrations | String, String | None | This will apply security to Head Clients of the given letters. |
| canAccess | IDfId | Boolean | Returns whether or not the logged in user has permission to access a folder. |
| canModifyConfidential | IDfId | Boolean | Returns whether or not the logged in user can modify the Confidential status of a folder. |
| cleanupDeletedFolder | IDfId, String | None | This will clean up all registrations for a deleted Job, Project or deactivated Client or Sub Client. |
| copyRegistrations | IDfId, IDfId | None | This will copy Registrations from one folder to another folder. |
| copyRegistrationsX | IDfId, IDfList | None | This will copy Registrations from one folder to a list of folders. |
| deleteRegistrationsForFolder | IDfId | None | This will delete all Registrations to a folder. |
| extendRegistrations | IDfList, Integer | None | This will extend a list of Registrations by the given time period. |
| ~~getAllRegsForFolder~~ | ~~IDfId, String~~ | ~~IDfList~~ | ~~This will return a list of Registrations the logged in user can approve within the same Head Client as the folder passed in and for a given status.~~ |
| ~~getAllRegsForFolderQuery~~ | ~~IDfId, String,~~  ~~String~~ | ~~String~~ | ~~This will return the query used to retun a list of Registrations the logged in user can approve within the same Head Client as the folder passed in, for a given status and folder level to restrict to.~~ |
| ~~getAllRegsForFolderAndUser~~ | ~~IDfId, String, String~~  ~~String~~ | ~~IDfList~~ | ~~This will return a list of Registrations the logged in user can approve within the same Head Client as the folder passed in, for a user, for a given status and folder level to restrict to.~~ |
| ~~getAllRegsForFolderAndUserQuery~~ | ~~IDfId, String, String,~~  ~~String~~ | ~~String~~ | ~~This will a query that can be run to return a list of Registrations the logged in user can approve within the same Head Client as the folder passed in, for a user, for a given status and folder level to restrict to.~~ |
| ~~getAllRegsForUser~~ | ~~String, String~~ | ~~IDfList~~ | ~~This will return a list of Registrations the logged in user can approve for a user and for a given status.~~ |
| ~~getAllRegsIcanApproveForFolderAndUserQueryArchive~~ | ~~IDfId,~~  ~~String,~~  ~~String,~~  ~~String~~ | ~~String~~ | ~~This method will be used only by the archive docbase. For registration administrators it will return all results but for registration Approvers it will not return shadow folders.~~  ~~This method returns the folders the user is approver of, when a folder and user name is used as search criteria.~~ |
| ~~getAllRegsIcanApproveForFolderQueryArchive~~ | ~~IDfId,~~  ~~String,~~  ~~String,~~  ~~String~~ | ~~String~~ | ~~This method will be used only by the archive docbase. For registration administrators it will return all results but for registration Approvers it will not return shadow folders.~~  ~~This method returns the folders the user is approver of, when a folder name is used as search criteria.~~ |
| ~~getAllRegsIcanApproveForUserQueryArchive~~ | ~~String,~~  ~~String,~~  ~~String~~ | ~~String~~ | ~~This method will be used only by the archive docbase. For registration administrators it will return all results but for registration Approvers it will not return shadow folders.~~  ~~This method returns the folders the user is approver of, when a user name is used as search criteria.~~ |
| getApprovers | IDfId, String | IDfList | This will get all Auto Approvers and/or Delegated Approvers for a folder based around the filter condition. |
| getApproversQuery | IDfId, String,  String[] | ArrayList | This will get all Auto Approvers and/or Delegated Approvers for a folder based around the filter condition. It will also return additional data about the Approvers, from the tls\_users table, as described by the column headings input. It is used by Manage Approvers. |
| ~~getApproversAndRegistrationsQuery~~ | ~~IDfId,~~  ~~String[]~~ | ~~String~~ | ~~This will return a query that can be run to retrieve a list of Approvers and their registration details for the folder passed in.~~ |
| getApproversAndRegistrationsQuery | IDfId,  String[],  boolean | String | This will return a query that can be run to retrieve a list of Approvers and their registration details for the folder passed in. If the passed in Boolean value is true, the generated query will fetch only those approvers who are active users within DMS. |
| getApproverUserId | IDfId, String | IDfList | This returns the object id of an approver to a folder. |
| ~~getFoldersUserIsAutoApproverFor~~ | ~~None~~ | ~~IdfList~~ | ~~This will return a list of folder which the user is an autoapprover for~~ |
| ~~getFoldersUserIsAutoApproverForQuery~~ | ~~String~~ | ~~String~~ | ~~This will return a DQL query string of querying folder type and path of folders for which the specified user is Auto Approver.~~ |
| getConfidentialParent | IDfId | IDfId | This returns the id of first Confidential folder found starting at the folder passed in, and working its way up the tree. If no Confidential folders are found it will return the Head Client. |
| getHeadClient | IDfId | IDfId | This returns the id of the Head Client for the given folder. |
| ~~getMyRegsitrations~~ | ~~String~~ | ~~IDfList~~ | ~~This returns a List of the logged in users Registration based on the status.~~ |
| ~~getMyRegistrationsQueryArchive~~ | ~~String,~~  ~~String~~ | ~~String~~ | ~~This method will be used only by the archive docbase. For registration administrators it will return all results but for registration Approvers it will not return shadow folders.~~  ~~This method will get all the registrations for the logged in user in archive.~~ |
| getNominatedApprovers | None | IDfList | This will return the list of user names of the Nominated Delegateed Approvers for the currently logged in user. |
| getRegisteredGroup | IDfId | String | This returns the name of the Registration for the given folder id if one exists, otherwise it returns null. |
| getSubFoldersUserCanApprove | IDfId | IDfList | This returns a list of folders the logged in user is an approver for a given Client folder. |
| getSubFoldersUserCanApproveQuery | IDfld | String | This returns a query string for getting a list of folders the logged in user is an approver for a given Client folder. |
| isApprover | IDfId | Boolean | This returns whether or not the logged in user is an approver for the given folder. |
| isApproverWithinClient | IDfId | Boolean | This returns whether or not the logged in user is an approver for any folders within the given Client. |
| isAutoApprover | IDfId | Boolean | This returns whether or not the logged in user is an Auto Approver for the given folder. |
| isConfidential | IDfId | Boolean | This returns whether or not the given folder is Confidential. |
| isHeadClient | IDfId | Boolean | This returns whether or not the given folder is a Head Client. |
| markConfidential | IDfId | None | This will mark the current folder as Confidential. |
| markConfidential | IDfId, Boolean | None | This will mark the current folder as Confidential running asynchhronously or synchhronously depending on the Boolean passed in. |
| markConfidentialX | IDfList, Boolean | None | This will mark a list of folders as Confidential running asynchhronously or synchhronously depending on the Boolean passed in. |
| moveRegistrations | IDfId, IDfId | None | This will move registration from one folder to another as part of the Client/Job Move process. |
| reassignPendingClientRegistrations | IDfId, IDfId | None | This will move registration from one folder to another as part of a Pending Client reassignment. |
| removeAutoApprover | IDfId, String | None | This will remove the passed in user as an Auto Approver for the given folder. |
| removeAutoApprovers | IDfId, IDfList | None | This will remove a list of Auto Approvers for the given folder. |
| removeDelegatedApprover | IDfId, String | None | This will remove a Delegated Approver for the given folder. |
| removeDelegatedApprovers | IDfId, IDfList | None | This will remove a list of Delegated Approvers for the given folder. |
| removeNominatedApprovers | IDfList | None | This will remove the Nominated Delegateed Approvers for the currently logged in user. |
| removeRegistrations | IDfList | None | This will expire a list of Registrations, for the Registration Object Id’s passed in. |
| setAutoApprovers | IDfId | None | This will set the Auto Approvers for the given folder. |
| setAutoApprovers | IDfList, Boolean | None | This will set the Auto Approvers for a list of folders running asynchhronously or synchhronously depending on the Boolean passed in. |
| unmarkConfidential | IDfId | None | This will unmark a folder as Confidential. |
| unmarkConfidentialX | IDfList, Boolean | None | This will unmark a list of folders as Confidential running asynchhronously or synchhronously depending on the Boolean passed in. |
| ~~synchRegistration~~ | ~~IdfID,~~  ~~String,~~  ~~IDfID,~~  ~~String~~ | ~~None~~ | ~~This will synch the registrations in active and archive docbase~~ |
| isRegistrationAdmin | String | Boolean | Returns whether or not the given user is a Registration Admin user. |
| canModifyRegApproverAccess | IDfId, String | Boolean | Returns whether or not the given user can modify the “Authorized to create Registrations” setting of a folder. |
| canModifyDefaultGroup | IDfid, String | Boolean | Returns whether or not the given user can modify the default access group of a folder. |
| modifyClientDefaultGroup | IDfid, String | None | Updates the Default Group attribute on Clients |
| createClientDefaultGroupReg | IDfid, String | None | Creates Registrations for all users in the group assigned to a Client.  Note: Expected to only be called during Client creation. |
| addClientDefaultGroupReg | IDfid, String | None | Adds a Registration for a user to a Client for the configured period used by Client Default Group Registrations. |
| modifyRegApproverAccess | IDfid, String, String | None | Updates the Registration Approver Access setting and the GUID of the user who requested the modification for Clients, Jobs and Projects. |
| updateProductGroupReg | IDfList, String | None | Updates the Registrations for a configured Product to align with the members of a group. |
| setProductSpecificAccess | IDfList | None | Sets a Job to use Product Specific Access. |
| unsetProductSpecificAccess | IDfList | None | Unsets a Job to use Product Specific Access. |
| isAllowedProductSpecificAccess | IDfid | Boolean | Verifies Product Specific Access can be enabled for a Folder. |
| canSetUnsetProductSpecificAccess | IDfId, String | Boolean | Verifies a user has permission to Set/Unset Product Specific Access. |
| updateProductSpecificAccessExclusions | String, String, String | None | Updates the Product Specific Access Exclusion list with Client changes. |
| isDelegatedApprover | IDfId, String | boolean | Checks if the user has a Delegated Approver Registration to the folder. |
| getMyRegistrationQuery | String,  String,  String,  String,  String, | String | Method will generate query based on the search criteria and filter condition and will be used to populate My Registration window. |
| getManageRegistrationsQuery | IDfId,  String,  String,  String,  String,  String,  String,  String | String | The method based on the filter Criteria/ Search Criteria will will deciede which  Queries to join to generate the final Registration Search Result query for Manage Registration window. |

##### Private Methods

| Name | Input | Output | Description |
| --- | --- | --- | --- |
| applySecurityToClient | IDfId | None | This will apply security to a Head Client. |
| canAddDelegatedApprover | IDfId | Boolean | This will return whether or not the Delegated Approver limit has been reached for the given folder. |
| createResultObject | IDfId | IDfId | This will create the result object to be passed to the server. |
| createResultsObject | IDfId, IDfList | IDfId | This will create the results object to be passed to the server, taking in a list of user Guids and storing them within the ‘t\_result\_value’ attribute of the result object. |
| deleteServerObject | IDfId | None | This will delete a result object. |
| runServerMethod | String[], String. Boolean | None | This will run the server method. |
| setAutoApprovers | IDfId, Boolean | None | This will set the Auto Approvers for a folder running asynchhronously or synchhronously depending on the Boolean passed in. |

#### getManageRegistrationsQuery

This function will return a query string, based on the passed in parameters that can be executed to populate the Manage Registration search results window.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idClient | IDfId | IN | The Client id of the Head Client. |
| strSelectedUser | String | IN | The user whose Registration is searched for. |
| strStatus | String | IN | Status of the Registration. Active or Expired. |
| strRestrictedFolderName | String | IN | The name of the Restricted folder on which the search is made. |
| strRestrictedFolderType | String | IN | The search filter type used to search Restricted folder name. The values are ‘starts with’, ‘Contains’, ‘Ends with’. |
| strRestrictedFolderType | String | IN | The folder type of the folder being searched. |
| strFilterName | String | IN | The filter name on which the search results are filterd on. |
| strFilterValue | String | IN | The corresponding value for the filter name. |
| strDQL | String | OUT | The query string to populate Manage Registration search results window. |

##### Pseudocode

*Call TlsRegQuery.getManageRegistrationsQuerypassing all the parameters.*

#### getMyRegistrationQuery

the function will return Registration Query that can be used to populate ‘My Registration’ search results window.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strRestrictedFolderName | String | IN | The name of the Restricted folder on which the search is made. |
| strRestrictedFolderType | String | IN | The search filter type used to search Restricted folder name. The values are ‘starts with’, ‘Contains’, ‘Ends with’. |
| strRestrictedFolderType | String | IN | The folder type of the folder being searched. |
| strFilterName | String | IN | The filter name on which the search results are filterd on. |
| strFilterValue | String | IN | The corresponding value for the filter name. |
| strDQL | String | OUT | The query string to populate My Registration search results window. |

##### Pseudocode

*Call method TlsRegQuery.getMyRegistrationQuery passing all parameters.*

#### getFoldersUserIsAutoApproverFor

This function will return a list of folders which the user is an auto-approver for.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstFolders | List | OUT | The folders |

##### Pseudocode

*Call TlsRegQuery*

#### getFoldersUserIsAutoApproverForQuery

This function will return a query string for getting the folder type and path of folders for which the specified user is an auto-approver.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strUserName | String | IN | The user who is an auto-approver for the folders being queried. |
| strQuery | String | OUT | The query string for getting folder type and path of folders for which the specified user is an auto-approver. |

##### Pseudocode

*Call TlsRegQuery*

#### addDelegatedApprover

This function is the batch version of add Delegated Approver to a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstFolders | List | IN | The folders |
| strUserName | String | IN | The user being added. |

##### Pseudocode

*Loop through list and call existing add delegated approver (below)*

#### addDelegatedApprover

This function calls the server side component to add a user as a Delegated Approver to a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | String | IN | The folder. |
| strUserName | String | IN | The user being added. |

##### Pseudocode

*None Shown*

#### addDelegatedApprovers

This function calls the server side component to add a list of users as Delegated Approvers to a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | String | IN | The folder. |
| lstUserName | IDfList | IN | The users being added. |

##### Pseudocode

*None Shown*

#### addRegistration

This function calls the server side component to add a Registration for a user a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | String | IN | The folder. |
| strUserName | String | IN | The user being added. |
| intDuration | Integer | IN | The duration to create the Registration for. |

##### Pseudocode

*None Shown*

#### addRegsForUsers

This function calls the server side component to add Registrations for a list of users to a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | String | IN | The folder. |
| lstUserName | IDfList | IN | The users being added. |
| intDuration | Integer | IN | The duration to create the Registration for. |

##### Pseudocode

*None Shown*

#### addRegsToFolders

This function calls the server side component to add Registrations for a user to a list of folders.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstFolder | IDfList | IN | The folders to add the user to. |
| strUserName | String | IN | The user being added. |
| intDuration | Integer | IN | The duration to create the Registration for. |

##### Pseudocode

*None Shown*

#### addRegsToFoldersForUsers

This function calls the server side component to add Registrations for a list of users to ~~a~~ a list of folders.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstFolder | IDfList | IN | The folders to add the users to. |
| lstUserName | IDfList | IN | The users being added. |
| intDuration | Integer | IN | The duration to create the Registration for. |

##### Pseudocode

*None Shown*

#### addRegsToFoldersForUsers

This function calls the server side component to add Registrations for a list of users to a list of folders. If blnAddRegistrationToHeadClient is true it will also register users to the Head Client of Restricted folders, provided the logged in user is an registration approver at the Head Client level for the Restricted folder(s).

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstFolder | IDfList | IN | The folders to add the users to. |
| lstUserName | IDfList | IN | The users being added. |
| intDuration | Integer | IN | The duration to create the Registration for. |
| blnAddRegistrationToHeadClient | Boolean | IN | A Boolean value to determine whether registrations should be added to the Head Client as well. |

##### Pseudocode

*Make a list of Head Clients for which to add registrations to*

lstHeadClients = new List

IF blnAddRegistrationToHeadClient == true Then

For each folder in lstFolders Do

SET headClient = GET Head Client of folder, calling getHeadClient passing  
folderId

*Check if the Head Client is already in the list of headClients*

IF lstHeadClients does not contain headClient Then

*Check if user is a registration approver for the Head Client*

IF (CALL isApprover passing headClient)

Then Add headClient to lstHeadClients

ELSE write error to log

END IF

END FOR

END IF

Add lstHeadClients to lstFolder

Construct result object

Run method on server

**Construction Hint**

Currently, the method checks if the user is an AutoApprover for the Head Client and hence the registrants are not able to add new registrations. The design above checks for registrations Approvers and hence registrants will also be able to add registrations if t\_create\_reg is set to “Registrants and Approvers”.

#### batchresetRegistrations

This function will apply security to Head Clients of the given letters.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strClientLetters | String | IN | The letters of the Head Clients to reset. |
| strCabinetId | String | IN | The cabinet id that the Head Clients to reset are within. |

##### Pseudocode

*None Shown*

#### canAccess

This function will return whether or not the logged in user has permission to access a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolderId | IDfId | IN | The folder id. |
| blnCanAccess | Boolean | OUT | Whether or not the user can access the folder. |

##### Pseudocode

*None Shown*

#### canModifyConfidential

This function will return whether or not the logged in user has permission to modify the Confidential status of a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolderId | IDfId | IN | The folder id. |
| blnCanModifyConfidential | Boolean | OUT | Whether or not the user can modify the Confidential status of the folder. |

##### Pseudocode

*None Shown*

#### copyRegistrations

This function will call the server side component to copy Registrations from one folder to another.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idSourceFolder | IDfId | IN | The folder to copy Registrations from. |
| idTargetFolder | IDfId | IN | The folder to copy Registrations to. |

##### Pseudocode

*None Shown*

#### copyRegistrationsX

This function will call the server side component to copy Registrations from one folder to a list of folders.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idSourceFolder | IDfId | IN | The folder to copy Registrations from. |
| lstDestinationFolder | IDfList | IN | The folders to copy Registrations to. |

##### Pseudocode

*None Shown*

#### deleteRegistrationsForFolder

This function will call the server side component to delete all Registration to a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder to delete all Registrations for. |

##### Pseudocode

*None Shown*

#### extendRegistrations

This function will call the server side component to extend a list of Registrations for a period of time.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstRegistrations | IDfList | IN | The Registrations to extend. |
| intDuration | Integer | IN | The duration to extend the Registrations by. |

##### Pseudocode

*IF lstRegistration IS NULL THROW new ILLEGALARGUMENTEXCEPTION*

*IF lstRegistration.getCount < 1 THROW new ILLEGALARGUMENTEXCEPTION*

*IF lstRegistration.getElementType != IdfList.DF\_ID THROW new ILLEGALARGUMENTEXCEPTION*

*IDfId pobjResult = createResultObject(lstRegistration);*

*//set up parameters to call server method to perform registration update.*

*String[] strarrParams =*

*{*

*HYPHEN + ManageRegistrations.ARG\_ACTION,*

*ManageRegistrations.ACTION\_EXTEND\_REGISTRATION,*

*HYPHEN + ManageRegistrations.ARG\_RESULT\_OBJECT,*

*pobjResult.toString(), HYPHEN+*

*ManageRegistrations.ARG\_DURATION,*

*Integer.toString(intDuration)};*

*runServerMethod(strarrParams,MANAGE\_REGISTRATIONS\_METHOD\_NAME,* ***true****);*

#### ~~getAllRegsForFolder~~

~~This function will return a list of Registrations object ids that the logged in user can approve within the same Head Client as the folder passed in with a particular Registration status.~~

~~Valid statuses are ‘ALL’, ‘ACTIVE’, ‘EXPIRED’, ‘EXPIRED-LEFT’, ‘EXPIRED-DENIED’ and ‘EXPIRED-LAPSED’.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~idFolder~~ | ~~IDfId~~ | ~~IN~~ | ~~The folder to get all Registrations within the same Head Client the user can approve.~~ |
| ~~strStatus~~ | ~~String~~ | ~~IN~~ | ~~The status of the Registrations to retrieve.~~ |
| ~~lstRegistrations~~ | ~~IDfList~~ | ~~OUT~~ | ~~A list of Registration object ids the user can approve.~~ |

##### ~~Pseudocode~~

~~IF one of the arguments was null, throw an exception.~~

~~IF idFolder is not a Head Client, get the Head Client for this folder~~

~~IF strStatus == ALL~~

~~RETURN TlsRegQuery getAllRegsIcanApproveForFolder passing idFolder~~

~~ELSE IF strStatus is one of ‘ACTIVE’, ‘EXPIRED’, ‘EXPIRED-LEFT’, ‘EXPIRED-DENIED’ or ‘EXPIRED-LAPSED’~~

~~RETURN~~

~~RETURN TlsRegQuery getAllRegsIcanApproveForFolder passing idFolder, strStatus~~

~~ELSE throw an exception as this is an illegal status~~

~~END IF~~

#### ~~getAllRegsForFolderQuery~~

~~This function will return a query that can be run to return a list of Registrations object ids that the logged in user can approve within the same Head Client as the folder passed in with a particular Registration status. This query will also have ability to restrict the list of Registrations object ids to either Sub-client or Job or Project level folders.~~

~~Valid statuses are ‘ALL’, ‘ACTIVE’, ‘EXPIRED’, ‘EXPIRED-LEFT’, ‘EXPIRED-DENIED’ and ‘EXPIRED-LAPSED’.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~idFolder~~ | ~~IDfId~~ | ~~IN~~ | ~~The folder to get all Registrations within the same Head Client the user can approve.~~ |
| ~~strStatus~~ | ~~String~~ | ~~IN~~ | ~~The status of the Registrations to retrieve.~~ |
| ~~strRestrictFolder~~ | ~~String~~ | ~~IN~~ | ~~The folder levels to retrieve Registrations for.~~ |
| ~~strQuery~~ | ~~String~~ | ~~OUT~~ | ~~A query that can be run to return a list of Registration object ids the user can approve.~~ |

##### ~~Pseudocode~~

~~IF one of the arguments was null, throw an exception.~~

~~IF strStatus == ALL~~

~~RETURN TlsRegQuery getAllRegsIcanApproveForFolderQuery passing idFolder and strRestrictFolder.~~

~~ELSE IF strStatus is one of ‘ACTIVE’, ‘EXPIRED’, ‘EXPIRED-LEFT’, ‘EXPIRED-DENIED’ or ‘EXPIRED-LAPSED’~~

~~RETURN~~

~~RETURN TlsRegQuery getAllRegsIcanApproveForFolderQuery passing idFolder, strStatus and strRestrictFolder.~~

~~ELSE throw an exception as this is an illegal status~~

~~END IF~~

#### ~~getAllRegsForFolderAndUser~~

~~This function will return a list of Registrations object ids that the logged in user can approve within the same Head Client as the folder passed in, for a particular user, with a particular Registration status.~~

~~Valid statuses are ‘ALL’, ‘ACTIVE’, ‘EXPIRED’, ‘EXPIRED-LEFT’, ‘EXPIRED-DENIED’ and ‘EXPIRED-LAPSED’.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~idFolder~~ | ~~IDfId~~ | ~~IN~~ | ~~The folder to get all Registrations within the same Head Client the user can approve.~~ |
| ~~strUserName~~ | ~~String~~ | ~~IN~~ | ~~The Guid of the user to retrieve Registrations for.~~ |
| ~~strStatus~~ | ~~String~~ | ~~IN~~ | ~~The status of the Registrations to retrieve.~~ |
| ~~strRestrictFolder~~ | ~~String~~ | ~~IN~~ | ~~The folder levels to retrieve Registrations for.~~ |
| ~~lstRegistrations~~ | ~~IDfList~~ | ~~OUT~~ | ~~A list of Registration object ids the user can approve.~~ |

##### ~~Pseudocode~~

~~IF one of the arguments was null, throw an exception.~~

~~IF strStatus == ALL~~

~~RETURN TlsRegQuery getAllRegsIcanApproveForFolderAndUser passing idFolder, strUserName and strRestrictFolder.~~

~~ELSE IF strStatus is one of ‘ACTIVE’, ‘EXPIRED’, ‘EXPIRED-LEFT’, ‘EXPIRED-DENIED’ or ‘EXPIRED-LAPSED’~~

~~RETURN~~

~~RETURN TlsRegQuery getAllRegsIcanApproveForFolderAndUser passing idFolder, strUserName, strStatus and strRestrictFolder.~~

~~ELSE throw an exception as this is an illegal status~~

~~END IF~~

#### ~~getAllRegsForFolderAndUserQuery~~

~~This function will return a query that can be used to return a list of Registrations object ids that the logged in user can approve within the same Head Client as the folder passed in, for a particular user, with a particular Registration status.~~

~~Valid statuses are ‘ALL’, ‘ACTIVE’, ‘EXPIRED’, ‘EXPIRED-LEFT’, ‘EXPIRED-DENIED’ and ‘EXPIRED-LAPSED’.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~idFolder~~ | ~~IDfId~~ | ~~IN~~ | ~~The folder to get all Registrations within the same Head Client the user can approve.~~ |
| ~~strUserName~~ | ~~String~~ | ~~IN~~ | ~~The Guid of the user to retrieve Registrations for.~~ |
| ~~strStatus~~ | ~~String~~ | ~~IN~~ | ~~The status of the Registrations to retrieve.~~ |
| ~~strRestrictFolder~~ | ~~String~~ | ~~IN~~ | ~~The folder levels to retrieve Registrations for.~~ |
| ~~strQuey~~ | ~~String~~ | ~~OUT~~ | ~~A query that can be run to return a list of Registration object ids the user can approve.~~ |

##### ~~Pseudocode~~

~~IF one of the arguments was null, throw an exception.~~

~~IF strStatus == ALL~~

~~RETURN TlsRegQuery getAllRegsIcanApproveForFolderAndUserQuery passing idFolder, strUserName and strRestrictFolder.~~

~~ELSE IF strStatus is one of ‘ACTIVE’, ‘EXPIRED’, ‘EXPIRED-LEFT’, ‘EXPIRED-DENIED’ or ‘EXPIRED-LAPSED’~~

~~RETURN~~

~~RETURN TlsRegQuery getAllRegsIcanApproveForFolderAndUserQuery passing idFolder, strUserName, strStatus and strRestrictFolder.~~

~~ELSE throw an exception as this is an illegal status~~

~~END IF~~

#### getApprovers

This function will return a list of object ids of the users who are Auto and/or Delegated Approvers for a given folder.

Valid filter statuses are ‘ALL’, ‘AUTO’ and ‘DELEGATED’.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder to get the Approvers for. |
| strFilter | String | IN | Whether to get Auto Approvers, Delegated Approvers or both for the folder. |
| lstApprovers | IDfList | OUT | A list of object ids of user who are Approvers for the folder given the filter status. |

##### Pseudocode

*None Shown*

#### getApproversQuery

This function will return an ArrayList of the specified data of users who are Auto and/or Delegated Approvers for a given folder.

Valid filter statuses are ‘ALL’.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder to get the Approvers for. |
| strFilter | String | IN | Whether to get Auto Approvers, Delegated Approvers or both for the folder. |
| strarrHeadings | String[] | IN | An array of column headings to be returned |
| strQuery | String | OUT | A query that can be run to return the specified parameters for the users who are Approvers for the folder given the filter status. |

##### Pseudocode

IF strFilter is ALL Then

Return CALL TlsRegQuery.getAllApproversQuery(idFolder, strarrHeadings)

ELSE

Return CALL TlsRegQuery.getApproversByStatusQuery(idFolder, strFilter,   
 strarrHeadings)

END IF

#### getApproversAndRegistrationsQuery

This function will return the Query that can be executed to retrieve the specified data for users who are Auto and Delegated Approvers for a given folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idClient | IDfId | IN | The folder to get the Approvers and Registrations for. |
| strarrHeadings | String[] | IN | An array of column headings to query on. Valid headings are from the tls\_users and tls\_registrations table |
| strQuery | String | OUT | A query that can be run to return the specified parameters for the Approvers and their registrations. |

##### Pseudocode

Return CALL TlsRegQuery.getAllApproversAndRegistrationsQuery(idClient, strarrHeadings)

#### getApproversAndRegistrationsQuery

This overridden function will return the Query that can be executed to retrieve the specified data for users who are Auto and Delegated Approvers for a given folder.

If the Boolean value passed into the method is ‘True’ the query returned will fetch only those Registration Approvers who are active users within the DMS else the query returned will fetch both inactive and active users.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idClient | IDfId | IN | The folder to get the Approvers and Registrations for. |
| strarrHeadings | String[] | IN | An array of column headings to query on. Valid headings are from the tls\_users and tls\_registrations table |
| blnIsActiveUserOnly | Boolean | IN | Indicates if the method returns all Approver irrespective of whether they are active or inactive user within DMs. True indicates only active users and false will indiacate both actyive and inactive users with DMS |
| strQuery | String | OUT | A query that can be run to return the specified parameters for the Approvers and their registrations. |

##### Pseudocode

String strReturnQuery = TlsRegQuery.getAllApproversAndRegistrationsQuery(idClient, strarrHeadings)

IF blnIsActiveUserOnly == TRUE

strReturnQuery = + “AND user\_state ='0' ”

RETURN strReturnQuery

#### getApproverUserId

This function will return a list of object ids of the users who are Approvers for a Registration give the folder id and name of the user whose Registration the approver will be returned for.

Note: Currently only one user object id will be returned.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder to get the Registration is to. |
| strUserName | String | IN | The Guid of the user the Registration is for. |
| lstApproverUserId | IDfList | OUT | A list of object ids of the user who was the Approver of the Registration. |

##### Pseudocode

*None Shown*

#### getConfidentialParent

This function will return the object id of the first Confidential found, by starting at the folder passed in, and working its way up the tree. If no Confidential folder is found it will return the Head Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder to start looking for the Confidential folder at. |
| idConfidentialParent | IDfId | OUT | The object id of the first Confidential folder found, or the Head Client. |

##### Pseudocode

*None Shown*

#### getHeadClient

This function will return the object id of a folder’s Head Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder to locate the Head Client for. |
| idHeadClient | IDfId | OUT | The object id of the Head Client. |

##### Pseudocode

*None Shown*

**Construction Hint**

May need updating to use t\_ultimate\_head\_client\_code attribute and get correct top level Head Client.

#### ~~getMyRegistrations~~

~~This function will return a list of Registrations object ids that belong to the logged in user with a particular Registration status.~~

~~Valid statuses are ‘ALL’, ‘ACTIVE’, ‘EXPIRED’, ‘EXPIRED-LEFT’, ‘EXPIRED-DENIED’ and ‘EXPIRED-LAPSED’.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~strStatus~~ | ~~String~~ | ~~IN~~ | ~~The status of the Registrations to retrieve.~~ |
| ~~lstRegistrations~~ | ~~IDfList~~ | ~~OUT~~ | ~~A list of Registration object that belong to the user with the selected status.~~ |

##### ~~Pseudocode~~

~~IF one of the arguments was null, throw an exception.~~

~~IF idFolder is not a Head Client, get the Head Client for this folder~~

~~IF strStatus == ALL~~

~~RETURN TlsRegQuery getMyRegistrations~~

~~ELSE IF strStatus is one of ‘ACTIVE’, ‘EXPIRED’, ‘EXPIRED-LEFT’, ‘EXPIRED-DENIED’ or ‘EXPIRED-LAPSED’~~

~~RETURN~~

~~RETURN TlsRegQuery getMyRegistrationsByStatus passing strStatus~~

~~ELSE throw an exception as this is an illegal status~~

~~END IF~~

#### getRegisteredGroup

This function will return a name of the Registration group for the selected folder. If the group can not be found then null will be returned.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder. |
| strRegisteredGroup | String | OUT | The name of the Registration group for the folder. |

##### Pseudocode

*None Shown*

#### getReadOnlyGroup

This function will return the name of the Read Only group for the selected folder. If the group cannot be found then null will be returned.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder. |
| strReadOnlyGroup | String | OUT | The name of the Read Only group for the folder. |

##### Pseudocode

*Get the Read only group for the current folder*

grpReadOnlyGroup =

CALL getGroup passing TlsRegistration.READ\_ONLY\_GROUP\_PREFIX + idFolder.getId

IF grpReadOnlyGroup == null The

RETURN null

ELSE RETURN grpRegistrationGroup.getGroupName()

#### getSubFoldersUserCanApprove

This function will return a list of folder ids for all the folders within a given Client that the logged in user is an approver for.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idClient | IDfId | IN | The Client folder to get all folders the user is an Approver for. |
| lstFolders | IDfList | OUT | A list of folder ids the user is an approver for in within the Client folder. |

##### Pseudocode

*~~Get the logged in username~~*

*~~Call the Registration SBO method getManageRegistration() passing the Head Client code and the logged in user name.~~*

*~~The SBO method will return query that can be run to get the list of~~* ~~folders within a given Client that the logged in user is an approver for.~~

~~RETURN~~ *~~the list of r\_object\_id from the query execution resultset.~~*

#### getSubFoldersUserCanApproveQuery

This function will return a query string for getting a list of folder ids, folder path and folder type for all the folders within a given Client that the logged in user is an approver for.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idClient | IDfId | IN | The Client folder to get all folders for which the user is an Approver. |
| ~~strInStatement~~ | ~~String~~ | ~~IN~~ | ~~The column name used in the In-Clause statement.~~ |
| strQuery | String | OUT | The query string for getting a list of folder ids the user is an approver for in within the Client folder. |

##### Pseudocode

*RETURN CALL TLSQuery.getSubFoldersUserCanApproverQuery(idClient~~, true, strInStatement)~~*

##### Construction Hint:

The existing logic of generating a query that can be used to populate the folder information has been moved to the method TlsRegQuery.getSubFoldersUserCanApproveQuery.

#### isApprover

This function will return if the logged in user is an Auto or Delegated Approver for the given folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder. |
| blnIsApprover | Boolean | OUT | Whether or not the user is an Auto or Delegated Approver for the folder. |

##### Pseudocode

*None Shown*

#### ~~isApproverWithinClient~~

~~This function will return if the logged in user is an Auto or Delegated Approver for any folder within the given Client.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~idClient~~ | ~~IDfId~~ | ~~IN~~ | ~~The Client folder.~~ |
| ~~blnIsApproverWithinClient~~ | ~~Boolean~~ | ~~OUT~~ | ~~Whether or not the user is an Auto or Delegated Approver for a folder within the Client.~~ |

##### ~~Pseudocode~~

*~~None Shown~~*

#### isAutoApprover

This function will return if the logged in user is an Auto Approver for the given folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder. |
| blnIsAutoApprover | Boolean | OUT | Whether or not the user is an Auto Approver for the folder. |

##### Pseudocode

*None Shown*

#### isConfidential

This function will return whether or not the given folder’s t\_confidential attribute is set to true.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder. |
| blnIsConfidential | Boolean | OUT | Whether or not the folder is Confidential. |

##### Pseudocode

*None Shown*

#### isHeadClient

This function will return whether or not the given folder is a Head Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder to check. |
| blnIsHeadClient | Boolean | OUT | Whether or not the folder is a Head Client. |

##### Pseudocode

*None Shown*

#### markConfidential

This function will call the server side component to mark a folder as Confidential.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder to mark Confidential. |

##### Pseudocode

*None Shown*

#### markConfidential

This function will call the server side component to mark a folder as Confidential, running either asynchhronously or synchhronously.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder to mark Confidential. |
| blnAsycn | Boolean | IN | Whether to run the method asynchhronously or synchhronously. |

##### Pseudocode

*None Shown*

#### markConfidentialX

This function will call the server side component to mark a list of folders as Confidential, running either asynchhronously or synchhronously.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstFolders | IDfList | IN | A list of folders to mark Confidential. |
| blnAsycn | Boolean | IN | Whether to run the method asynchhronously or synchhronously. |

##### Pseudocode

*None Shown*

#### moveRegistrations

This function will call the server side component to move Registration from one folder to another after a Client, Job, or Project ove has taken place.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idOldConfidentialParent | IDfId | IN | The old Parent folder the moved folder was under. |
| idDestinationFolder | IDfId | IN | The new parent folder of the moved folder. |

##### Pseudocode

IF the source or destination folders are empty

Then throw IllegalArgumentException.

END IF

Call method createResultObject to return a result object from destination folder id.

Set up parameters to call server method to perform registration update.

Run the method on the server passing the parameters for the registrations.

#### reassignPendingClientRegistrations

This function will call the server side component to move Registration after a Pending Client has been reassigned.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idPendingFolder | IDfId | IN | The Pending Client being reassigned. |
| idDestinationFolder | IDfId | IN | The folder the Pending Client was reassigned to. |

##### Pseudocode

*None Shown*

#### reassignPendingJobRegistrations

This function will call the server side component to move Registration after a Pending Job has been reassigned.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idPendingFolder | IDfId | IN | The Pending Job being reassigned. |
| idDestinationFolder | IDfId | IN | The folder the Pending Job was reassigned to. |

##### Pseudocode

*None Shown*

#### removeAutoApprver

This function will call the server side component to remove a user’s Auto Approver Registration to a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder. |
| strUserName | String | IN | The Guid of the user to remove as an Auto Approver. |

##### Pseudocode

*None Shown*

#### removeAutoApprovers

This function will call the server side component to remove a list of user’s Auto Approver Registrations to a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder. |
| lstUserName | IDfList | IN | The list of users to remove as an Auto Approvers. |

##### Pseudocode

*None Shown*

#### removeDelegatedApprover

This function will call the server side component to remove a user’s Delegated Approver Registration to a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder. |
| strUserName | String | IN | The Guid of the user to remove as a Delegated Approver. |

##### Pseudocode

*None Shown*

#### removeDelegatedApprovers

This function will call the server side component to remove a list of user’s Delegated Approver Registrations to a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder. |
| lstUserName | IDfList | IN | The list of users to remove as Delegated Approvers. |

##### Pseudocode

*None Shown*

#### removeRegistrations

This function will call the server side component to expire a list of Registrations.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstRegistrations | IDfList | IN | The list of Registration ids to expire. |

##### Pseudocode

*None Shown*

#### setAutoApprovers

This function will call the private method setAutoApprovers to set the Auto Approvers for a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder to set the Auto Approvers for. |

##### Pseudocode

*None Shown*

#### setAutoApprovers

This function will call the server side component to set the Auto Approvers for a list of folders.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstFolder | IDfList | IN | The list of folders to set the Auto Approvers for. |
| blnAsycn | Boolean | IN | Whether to run the method asynchhronously or synchhronously. |

##### Pseudocode

*None Shown*

#### setAutoApprovers

This function will call the private method setAutoApprovers to set the Auto Approvers for a folder. Running either asynchronously or synchronously depending on the Boolean passed in.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder. |
| blnAsycn | Boolean | IN | Whether to run the method asynchronously or synchronously. |

##### Pseudocode

*None Shown*

#### unmarkConfidential

This function will call the server side component to unmark a folder as Confidential.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder to unmark as Confidential. |

##### Pseudocode

*None Shown*

#### unmarkConfidentialX

This function will call the server side component to unmark a list of folders as Confidential.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstFolders | IDfList | IN | A list of folders to mark Confidential. |
| blnAsycn | Boolean | IN | Whether to run the method asynchronously or synchronously. |

##### Pseudocode

Create an IDfList pobjResult by calling createResultObject, passing lstFolders

Create strarrParams, a list of parameters to pass to the server

Run the method on the server

IF blnAsycn was true Then

Delete the server object, calling deleteServerObject

END IF

##### Construction Hint

* + - 1. This method is very similar to markConfidentialX, only it calls the ACTION\_UNMARK\_CONFIDENTIAL action.

#### applySecurityToClient

This private function will call apply security to a Head Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idClientId | IDfId | IN | The Client to apply security to. |

##### Pseudocode

*None Shown*

#### canAddDelegatedApprover

This private function will check if the Delegated Approver limit has been reached and return true if it hasn’t a Delegated Approvers can be added to the folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder. |
| blnCanAddDelegatedApprover | Boolean | OUT | Whether or not more Delegated Approvers can be added to the folder. |

##### Pseudocode

*None Shown*

#### getNomintedApprovers

This function will return the list of guids of the Nominated Delegated Approvers for the currently logged on user. If the logged in user hasn’t nominated a delegated approver then an empty list will be returned.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstUserName | IDfList | OUT | The list of guids of the users who are nominated delegated approver for the currently logged on user. An empty list will be return if the user doesn’t have any Nominated Delegateed Approvers. |

##### Pseudocode

*Get the currently logged in user’s name.*

CALL getNominatedApprovers method within TlsRegQuery

#### addNominatedApprovers

This function will add the Nominated Delegateed Approvers for the logged in user. The actual Nominated Delegated Approver access to the folder will be granted by REG\_B04 – Manage Nominated Delegated Approver Job.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstUserName | IDfList | IN | The list of guids of the users who will be set as the nominated delegated approver. |

##### Pseudocode

*Get the currently logged in user’s name.*

CALL addNominatedApprover method within TlsRegQuery

#### removeNominatedApprovers

This function will remove the Nominated Delegateed Approvers for the logged in user. The actual Nominated Delegated Approver access to the folder will be granted by REG\_B04 – Manage Nominated Delegated Approver Job.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstUserName | IDfList | IN | The list of guids of the users who will be set as the nominated delegated approver. |

##### Pseudocode

*Get the currently logged in user’s name.*

CALL removeNominatedApprover method within TlsRegQuery

#### cleanupDeletedFolder

This function will call the server side component to clean up all registrations for a deleted Job or deactivated Client or Sub Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The folder to clean up registrations. |
| strRequestor | String | IN | The user requesting the operation. |

##### Pseudocode

*Create a new IDfList for objects retrieved from the folder*

IDfList lstObjects = new DfList();

*Append the folder ID to lstObjects*

lstObjects.appendId(idFolder);

*Call createResultObject passing in lstObjects*

IDfId pobjResult = createResultObject(lstObjects);

*Set up parameters to call server method to perform registration update.*

String[] strarrParams =

{

HYPHEN + ManageRegistrations.ARG\_ACTION,

ManageRegistrations. ACTION\_CLEAN\_UP\_DELETED\_FOLDER HYPHEN + ManageRegistrations.ARG\_RESULT\_OBJECT,

pobjResult.toString(),

HYPHEN + ManageRegistrations.ARG\_REQUESTOR,

strRequestor

};

runServerMethod(strarrParams, MANAGE\_REGISTRATIONS\_METHOD\_NAME, true);

#### synchRegistration

~~This method synchs the new registration changes in the active docbase with the archive docbase.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~srcFolder~~ | ~~IDfId~~ | ~~IN~~ | ~~Source folder ID~~ |
| ~~srcDocbase~~ | ~~String~~ | ~~IN~~ | ~~Source docbase name~~ |
| ~~dstFolder~~ | ~~IDfId~~ | ~~IN~~ | ~~Destination folder ID~~ |
| ~~dstDocbase~~ | ~~String~~ | ~~IN~~ | ~~Destination docbase name~~ |

##### ~~Pseudocode~~

* + - 1. ~~Get the registration users for the active folder.~~
      2. ~~Get the registration users for the archive folder.~~
      3. ~~For all users present in archive and not in active call removeRegistration and remove the user in the archive docbase.~~
      4. ~~For all users present in active and not in archive call addRegistration method in the archive docbase.~~

#### getMyRegistrationsQueryArchive

~~This function will return a query that can be run to return a list of the object ids of all the Registrations for the currently logged in user regardless of the Registration status.~~

~~Admin users will be able to retrive all the folders in line with the active docbase. Non- admin users will retrive only archived folders and not archive shadow folders.~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~strStatus~~ | ~~String~~ | ~~IN~~ | ~~The status of the Registrations to retrieve.~~ |
| ~~strWhereClause~~ | ~~String~~ | ~~IN~~ | ~~The where clause~~ |
| ~~strQuey~~ | ~~String~~ | ~~OUT~~ | ~~A query that can be run to return a list of Registration object ids the user can approve.~~ |

##### ~~Pseudocode~~

~~Call TlsRegQuery.getMyRegistrationsByStatusQueryArchive~~

#### getAllRegsIcanApproveForUserQueryArchive

~~This function will return a query that can be run to return a list of the object ids of all the Registrations that the currently logged in user is able to approve for the selected user regardless of the Registration status. The returned query can also be Restricted to look up registrations for a particular folder level.~~

~~Admin users will be able to retrieve all the folders in line with the active docbase. Non- admin users will retrieve only archived folders and not archive shadow folders.~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~strUser~~ | ~~String~~ | ~~IN~~ | ~~The user to get all Registrations.~~ |
| ~~strStatus~~ | ~~String~~ | ~~IN~~ | ~~The status of the Registrations to retrieve.~~ |
| ~~strWhereClause~~ | ~~String~~ | ~~IN~~ | ~~The where clause~~ |
| ~~strQuey~~ | ~~String~~ | ~~OUT~~ | ~~A query that can be run to return a list of Registration object ids the user can approve.~~ |

##### ~~Pseudocode~~

~~IF (strUser is NULL) OR (strStatus is null)~~

~~Throw exception.~~

~~END IF~~

~~return TlsRegQuery.getAllRegsIcanApproveForUserQueryArchive()~~

#### getAllRegsIcanApproveForFolderQueryArchive

~~This function will return a query that can be run to return a list of the object ids of all the Registrations that the currently logged in user is able to approve within the given Head Client or any Restricted folder regardless of the Registration status. The returned query can also be Restricted to look up registrations for a particular folder level.~~

~~Admin users will be able to retrieve all the folders in line with the active docbase. Non- admin users will retrieve only archived folders and not archive shadow folders.~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~idFolder~~ | ~~IDfId~~ | ~~IN~~ | ~~The folder to get all Registrations within the same Head Client the user can approve.~~ |
| ~~strStatus~~ | ~~String~~ | ~~IN~~ | ~~The status of the Registrations to retrieve.~~ |
| ~~strRestrictFolder~~ | ~~String~~ | ~~IN~~ | ~~The folder levels to retrieve Registrations for.~~ |
| ~~strWhereClause~~ | ~~String~~ | ~~IN~~ | ~~The where clause~~ |
| ~~strQuey~~ | ~~String~~ | ~~OUT~~ | ~~A query that can be run to return a list of Registration object ids the user can approve.~~ |

##### ~~Pseudocode~~

~~IF (idFolder is NULL) OR (strStatus is null) OR (strRestrictedFolder is null)~~

~~Throw exception.~~

~~END IF~~

~~return TlsRegQuery.getAllRegsIcanApproveForFolderQueryArchive()~~

#### getAllRegsIcanApproveForFolderAndUserQueryArchive

~~This function will return a query that can be run to return a list of the object ids of all the Registrations that the currently logged in user is able to approve within the given Head Client or any Restricted folder and the selected user regardless of the Registration status. The returned query can also be Restricted to look up registrations for a particular folder level.~~

~~Admin users will be able to retrieve all the folders in line with the active docbase. Non- admin users will retrieve only archived folders and not archive shadow folders.~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~idFolder~~ | ~~IDfId~~ | ~~IN~~ | ~~The folder to get all Registrations within the same Head Client the user can approve.~~ |
| ~~strUserName~~ | ~~String~~ | ~~IN~~ | ~~The Guid of the user to retrieve Registrations for.~~ |
| ~~strStatus~~ | ~~String~~ | ~~IN~~ | ~~The status of the Registrations to retrieve.~~ |
| ~~strRestrictFolder~~ | ~~String~~ | ~~IN~~ | ~~The folder levels to retrieve Registrations for.~~ |
| ~~strWhereClause~~ | ~~String~~ | ~~IN~~ | ~~The where clause~~ |
| ~~strQuey~~ | ~~String~~ | ~~OUT~~ | ~~A query that can be run to return a list of Registration object ids the user can approve.~~ |

##### ~~Pseudocode~~

~~IF (idFolder is NULL) OR (strUserName is null) OR (strStatus is null) OR (strRestrictedFolder is null)~~

~~Throw exception.~~

~~END IF~~

~~return. TlsRegQuery.getAllRegsIcanApproveForFolderAndUserQueryArchive()~~

#### isRegistrationAdmin

This function will return whether or not the given user is a Registration Admin user.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strUser | String | IN | The user to check permissions for. |
| blnIsAdmin | Boolean | OUT | Whether or not the user is a Reg Admin. |

##### Pseudocode

Check if user is in Reg Admin group.

##### Construction Hint

1. Extract from isApprover method. Methods that used to check if a user is a Registration Admin, will now call this method.

#### canModifyRegApprovalAccess

This function will return whether or not the given user has permission to modify the “Authorized to create Registrations” setting of a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolderId | IDfId | IN | The folder id. |
| strUser | String | IN | The user to check permissions for. |
| blnCanModify | Boolean | OUT | Whether or not the user can modify the “Authorized to create Registrations” setting of the folder. |

##### Pseudocode

RETURN CALL isAutoApprover()

**Construction Hint**

Ensure only returns Client Partner and Client Manager for Head Clients and not all Auto Approvers.

#### canModifyDefaultGroup

This function will return whether or not the given user has permission to modify the default access group of a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolderId | IDfId | IN | The folder id. |
| strUser | String | IN | The user to check permissions for. |
| blnCanModifyConfidential | Boolean | OUT | Whether or not the user can modify the default access group of the folder. |

##### Pseudocode

IF idFolderId object type is tls\_client THEN

blnApproverCan = get value of configuration item value of

‘/REG\_B01/ClientApproverCanModifyDefAccessGroup’

IF blnApproverCan

RETURN CALL isAutoApprover()

ELSE

RETURN CALL isRegistrationAdmin()

END IF

ELSE

RETURN False

END IF

#### modifyClientDefaultGroup

This function will update the Default Group attribute on Clients.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolderId | IDfid | IN | The folder to update default group for. |
| group | String | IN | The group to assign to the folder. |

##### Pseudocode

Calls the ModifyClientDefaultGroupCommand

#### createClientDefaultGroupReg

Creates Registrations for users in the default group for a Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolderId | IDfid | IN | The folder to update default group for. |
| group | String | IN | The group to assign to the folder. |

##### Pseudocode

Calls the CreateClientDefaultGroupRegCommand

#### addClientDefaultGroupReg

Calls the server method to create a Client Default Group Registration for a user.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolderId | IDfid | IN | The folder to update default group for. |
| userName | String | IN | The user to add a Registration for. |

##### Pseudocode

None Shown

#### modifyRegApproverAccess

This function will update the Registration Approver Access setting for a Client, Job or Project.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolderId | IDfid | IN | The folder to update setting for. |
| strRegAccess | String | IN | The Registration Approver Access setting. |
| strUser | String | IN | The user who requested the modification. |

##### Pseudocode

Calls the ModifyRegApproverAccessCommand

#### updateProductGroupReg

Creates Registrations for users in the group for a Product.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| productItemId | IDfid | IN | The Product Item Id to update group Registrations for. |
| group | String | IN | The group to assign to the folder. |

##### Pseudocode

Calls the UpdateProductGroupRegCommand

#### setProductSpecificAccess

Sets Product Specific Access for a Job.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idList | IDfList | IN | The id’s of the Jobs to set Product Specific Access |

##### Pseudocode

Calls the SetUnsetProductSpecificAccessCommand using “set” action.

#### unsetProductSpecificAccess

Unsets Product Specific Access for a Job.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idList | IDfList | IN | The id’s of the Jobs to unset Product Specific Access |

##### Pseudocode

Calls the SetUnsetProductSpecificAccessCommand using “unset” action.

#### isAllowedProductSpecificAccess

Verifies if a folder can have Product Specific Access enabled.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The id of the folder to check Restricted. |
| blnResult | Boolean | OUT | Whether can have Product Specific Access |

##### Pseudocode

blnPSAEnabled = Get value of ‘/Registration/ProductSpecificAccess/Enabled’

IF blnPSAEnabled = true THEN

colResults = EXECUTE Query 6 passing idFolder and

strProdCode = colResults.getValue(‘t\_product\_code’)

strClientCode = colResults.getValue(‘t\_client\_code’)

strClientId = colResults.getValue(‘client\_id’)

blnProdConfigured = check if strProdCode in

‘/Registration/ProductSpecificAccess/ProductGroups’

IF blnProdConfigured = true THEN

~~blnClientDisabled = check if strClientCode in~~

~~‘/Registration/ProductSpecificAccess/ExcludeClients’~~

blnClientDisabled = IF EXECUTE QUERY 12 Passing strClientId

RETURN blnClientDisabled

ELSE

RETURN false

END IF

ELSE

RETURN false

END IF

#### canSetUnsetProductSpecificAccess

Verifies if a user can set/unset Product Specific Acces.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strUserName | String | IN | The user id to verify. |
| idFolder | IDfId | IN | The folder to verify for. |
| blnResult | Boolean | OUT | Whether the user has access to set/unset. |

##### Pseudocode

IF idFolderId object type is tls\_job THEN

RETURN CALL isAutoApprover()

ELSE

RETURN False

END IF

#### updateProductSpecificAccessExclusions

Updates the Product Specific Access Exclusion list with requested Client changes.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strAddClients | String | IN | The Clients to add to the exclusion list. |
| strRemoveClients | String | IN | The Clients to remove from the exclusion list. |
| strRequester | String | IN | The GUID of the user requesting the changes. |

##### Pseudocode

Calls the UpdateProductSpecificAccessExclusionsCommand

## TlsRegQuery.java

This class contains the logic to execute queries and return the results to the TlsRegistrationImpl class.

**General Construction Hints**

Queries will need updating to allow for extra level of Clients by using t\_ultimate\_head\_client\_code.

Auto Approvers will need to allow for the configurable Client Auto Approvers and use the correct approver attributes.

#### getManageRegistrationsQuery

This function, based on the Registration search criteria and search result filter criterias will deciede which query to join to generate the final query string that can be used to populate Registration information for Manage Registration search results window.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idClient | IDfId | IN | The Client id of the Head Client on which the Manage Registration search is done. The Client id or the Selected user name must be entered. |
| strSelectedUser | String | IN | The uid of the user whose Registration information is searched for. The Client id or the Selected user name must be entered. |
| strStatus | String | IN | The Status of the Registration searched for (Active or Expired). |
| strRestrictedFolderName | String | IN | The name of the Restricted folder whose user Registration details are searched for. The value will be set to null if it is blank. If it is not balank the strRestrictedFolderFilterType should not be blank. |
| strRestrictedFolderFilterType | String | IN | Restricted folder name search criteria. Possible values are 'Contains', 'Begins with' and 'Ends with'. The value will be set to null if it is blank. |
| strRestrictedFolderType | String | IN | The folder type of the Restricted folder being searched for. The valid values are SUB\_CLIENT, JOB, PROJECT or RESTRICT.  The value will be set to null if it is blank. |
| strFilterName | String | IN | The name of the Search Result filter. The value will be set to null if it is blank. If filter name is not blank, the filter value should not be blank. |
| strFilterValue | String | IN | The corresponding Search Result filter value. The value will be set to null if it is blank. |
| strCurrentUser | String | IN | The logged in user name. The value will be set to null if it is blank. |
| strManageRegDQL | String | OUT | The final generated query that can be used to populate Manage Registration search Result based on the filter criteria and the search condition. |

##### Pseudocode:

Generate the below queries first:

**Query 1**: Query that can be run to fetch Registrations under a Head Client folder (strHeadClientRegQuery):

strHeadClientRegQuery = CALL getManageRegQueryForFolderType method passing all the argument but the folder type argument should be ‘HEAD\_CLIENT’, sort type as ‘1’ and argument blnIsForHeadClient as ‘True’ indicating that the registration is for Head Client.

**Query 2**: Query that can be run to fetch Registrations for a Job under Head Client folder (strJobRegUnderHeadClientQuery):

strJobRegUnderHeadClientQuery = CALL getManageRegQueryForFolderType method passing all the argument but the folder type argument should be ‘JOB’, sort type as ‘3’ and argument blnIsForHeadClient as ‘True’ indicating that the Registration is for a folder under Head Client.

**Query 3**: Query that can be run to fetch Registrations for a Project under Head Client folder (strProjectRegUnderHeadClientQuery):

strProjectRegUnderHeadClientQuery = CALL getManageRegQueryForFolderType method passing all the argument but the folder type argument should be ‘PROJECT’, sort type as ‘4’ and argument blnIsForHeadClient as ‘True’.

**Query 4**: Query that can be run to fetch Registrations for a Job under a Sub Client folder. (strSubClientRegQuery):

strSubClientRegQuery = CALL getManageRegQueryForFolderType method passing all the argument but the folder type argument should be ‘SUB\_CLIENT, sort type as ‘2’ and argument blnIsForHeadClient as ‘false’ indicating the registration is for Sub Client.

**Query 4**: Query that can be run to fetch Registrations for a Job under a Sub Client folder. (strJobRegUnderSubClientQuery):

strJobRegUnderSubClientQuery= CALL getManageRegQueryForFolderType method passing all the argument but the folder type argument should be ‘JOB’, sort type as ‘3’ and argument blnIsForHeadClient as ‘false’ indicating the registration is for a Job folder under Sub Client.

**Query 5**: Query that can be run to fetch Registrations for a Job under a Sub Client folder. (strProjectRegUnderSubClientQuery):

strProjectRegUnderSubClientQuery = CALL getManageRegQueryForFolderType method passing all the argument but the folder type argument should be ‘PROJECT’, sort type as ‘4’ and argument blnIsForHeadClient as ‘false’ indicating the registration is for a Project under Sub Client.

**Query 6**: Query that can be run to fetch Registrations for a Product under a Head Client folder. (strProductRegUnderHeadClientQuery):

strProductRegUnderHeadClientQuery= CALL getManageRegQueryForFolderType method passing all the argument but the folder type argument should be ‘PRODUCT’, sort type as ‘5’ and argument blnIsForHeadClient as ‘true’ indicating the registration is for a Product under Head Client.

**Query 7**: Query that can be run to fetch Registrations for a Product under a Sub Client folder. (strProductRegUnderSubClientQuery):

strProductRegUnderSubClientQuery= CALL getManageRegQueryForFolderType method passing all the argument but the folder type argument should be ‘PRODUCT’, sort type as ‘5’ and argument blnIsForHeadClient as ‘false’ indicating the registration is for a Product under Sub Client.

IF(strRestrictedFolderType != null || LEVEL.equals(strFilterName) )

IF (SUB\_CLIENT.equals(strRestrictedFolderType)|| SUB\_CLIENT.equals(strFilterValue)))

//if Restricted folder searched in Sub Client or search results are filtered on Sub Client

//then returns DQL to fetch Sub client Registrations.

Final Query (strManageRegDQL) = strSubClientRegQuery;

END IF

ELSE IF (JOB.equals(strRestrictedFolderType) || JOB.equals(strFilterValue))

//if Restricted folder searched in Job or search results are filtered on Job folder

//then return DQL to fetch Job Registrations at Head client and Sub Client level.

strManageRegDQL=strJobRegUnderHeadClientQueryUNION strJobRegUnderSubClientQuery

ENDIF

ELSEIF ( PROJECT.equals(strRestrictedFolderType) || PROJECT.equals(strFilterValue))

//if Restricted folder searched in Project or search results are filtered on Project folder

//then return DQL to fetch Project Registrations at Head client and Sub Client level.

strManageRegDQL = strProjectRegUnderHeadClientQueryUNION strProjectRegUnderSubClientQuery

ENDIF

ELSEIF (HEAD\_CLIENT.equals(strFilterValue))

//If search results are filtered on Head Client then

//return DQL to fetch Head Client Registration

strManageRegDQL = strHeadClientRegQuery

ENDIF

ELSE IF (RESTRICT.equals(strRestrictedFolderType))

//If the search is made only on 'Restricted' level folder

//Then return the Job Project registrations at head Client and Sub client level

//and Sub Client Registration.

strManageRegDQL =strJobRegUnderHeadClientQuery UNION strProjectRegUnderHeadClientQuery UNION

strSubClientRegQuery UNION strJobRegUnderSubClientQuery UNION strProjectRegUnderSubClientQuery

ENDIF

ELSEIF (PRODUCT.equals(strFilterValue) && blnIsSecurityAdmin)

//if the logged in user is a Security Admin and Search Results are

///filtered on Product folder then return dql to fetch

//Product Registration at Head Client and Sub Client Level

strManageRegDQL = strProductRegUnderHeadClientQuery UNION strProductRegUnderSubClientQuery

ENDIF

ENDIF

ELSEIF(FOLDER\_PATH.equals(strFilterName))

//If the search resluts are filtered based on folder path then get registration detail fromregistration table based on the

// object type and folder path

strManageRegDQL = CALL getRegistrationforFolderPathQuery());

ENDIF

ENDIF

ELSE

//If none of the above condition is true then the search is made on Head Client folder

//Hence return the union of all the above DQL

strManageRegDQL = strHeadClientRegQuery UNION

strJobRegUnderHeadClientQuery UNION strProjectRegUnderHeadClientQuery UNION

strSubClientRegQuery UNION strJobRegUnderSubClientQuery UNION

strProjectRegUnderSubClientQuery

IF(user is security admin)

//Union the DQL to fetch Product Registrations .

UNION strProductRegUnderHeadClientQuery UNION strProductRegUnderSubClientQuery

ENDIF

ENDIF

ENDIF

RETURN final Query strDQL.

##### Construction Hint:

Please refer to section [3.7.1.3](#_getManageRegQueryForFolderType) for implementaion logic for the method getManageRegQueryForFolderType.

#### getMyRegistrationsQuery

This function will return a query that can be executed to return a list of Active Registrations for the logged in user. The query will have the same results table with the same column names and filtering capabilities as is generated by the Manage Registration functionality above.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strRestrictedFolderName | String | IN | The name of the Restricted folder whose user Registration details are searched for. The value will be set to null if it is blank. If it is not blank the strRestrictedFolderFilterType should not be blank. |
| strRestrictedFolderType | String | IN | The folder type of the Restricted folder being searched for. The valid values are SUB\_CLIENT, JOB, PROJECT or RESTRICT.  The value will be set to null if it is blank. |
| strRestrictedFolderFilterType | String | IN | Restricted folder name search criteria. Possible values are 'Contains', 'Begins with' and 'Ends with'. The value will be set to null if it is blank. |
| strFilterName | String | IN | The name of the Search Result filter. The value will be set to null if it is blank. If filter name is not blank, the filter value should not be blank. |
| strFilterValue | String | IN | The corresponding Search Result filter value. The value will be set to null if it is blank. |
| strCurrentUser | String | IN | The Logged in user name. The value cannot be null |
| strManageRegDQL | String | OUT | The final query generated that can be used to populate active registration information for the logged in user. |

##### Pseudocode

The logic to generate the final My Registration will follow the same logic as the getManageRegistrationsQuery method above. The following are the few differences in the final query generated:

* The query will always return Active Registration.
* Get all Registrations information for the current user.
* The Head Client code and Head Client codes are always null.

The logic to generate the sub queries are as under:

**Query 1**: Query that can be run to fetch Registrations under a Head Client folder (strHeadClientRegQuery):

strHeadClientRegQuery = CALL getManageRegQueryForFolderType method passing all the argument but the folder type argument should be ‘HEAD\_CLIENT’, sort type as ‘1’ , argument blnIsForHeadClient as ‘True’ indicating that the registration is for Head Client, status of the Registration should be passed as ACTIVE, the indicator for security admin should be passed as True so that Product Specific Registration (if user has) can be returned and finally the uid for selected user and the current user should be same.

**Query 2**: Query that can be run to fetch Registrations for a Job under Head Client folder (strJobRegUnderHeadClientQuery):

strJobRegUnderHeadClientQuery = CALL getManageRegQueryForFolderType method passing all the argument but the folder type argument should be ‘JOB’, sort type as ‘3’, argument blnIsForHeadClient as ‘True’ indicating that the Registration is for a folder under Head Client, status of the Registration should be passed as ACTIVE, the indicator for security admin should be passed as True so that Product Specific Registration (if user has) can be returned and finally the uid for selected user and the current user should be same.

**Query 3**: Query that can be run to fetch Registrations for a Project under Head Client folder (strProjectRegUnderHeadClientQuery):

strProjectRegUnderHeadClientQuery = CALL getManageRegQueryForFolderType method passing all the argument but the folder type argument should be ‘PROJECT’, sort type as ‘4’, argument blnIsForHeadClient as ‘True’, status of the Registration should be passed as ACTIVE, the indicator for security admin should be passed as True so that Product Specific Registration (if user has) can be returned and finally the uid for selected user and the current user should be same.

**Query 4**: Query that can be run to fetch Registrations for a Job under a Sub Client folder. (strSubClientRegQuery):

strSubClientRegQuery = CALL getManageRegQueryForFolderType method passing all the argument but the folder type argument should be ‘SUB\_CLIENT, sort type as ‘2’, argument blnIsForHeadClient as ‘false’ indicating the registration is for Sub Client, status of the Registration should be passed as ACTIVE, the indicator for security admin should be passed as True so that Product Specific Registration (if user has) can be returned and finally the uid for selected user and the current user should be same .

**Query 4**: Query that can be run to fetch Registrations for a Job under a Sub Client folder. (strJobRegUnderSubClientQuery):

strJobRegUnderSubClientQuery= CALL getManageRegQueryForFolderType method passing all the argument but the folder type argument should be ‘JOB’, sort type as ‘3’, argument blnIsForHeadClient as ‘false’ indicating the registration is for a Job folder under Sub Client, status of the Registration should be passed as ACTIVE, the indicator for security admin should be passed as True so that Product Specific Registration (if user has) can be returned and finally the uid for selected user and the current user should be same.

**Query 5**: Query that can be run to fetch Registrations for a Job under a Sub Client folder. (strProjectRegUnderSubClientQuery):

strProjectRegUnderSubClientQuery = CALL getManageRegQueryForFolderType method passing all the argument but the folder type argument should be ‘PROJECT’, sort type as ‘4’, argument blnIsForHeadClient as ‘false’ indicating the registration is for a Project under Sub Client, status of the Registration should be passed as ACTIVE, the indicator for security admin should be passed as True so that Product Specific Registration (if user has) can be returned and finally the uid for selected user and the current user should be same .

**Query 6**: Query that can be run to fetch Registrations for a Product under a Head Client folder. (strProductRegUnderHeadClientQuery):

strProductRegUnderHeadClientQuery= CALL getManageRegQueryForFolderType method passing all the argument but the folder type argument should be ‘PRODUCT’, sort type as ‘5’, argument blnIsForHeadClient as ‘true’ indicating the registration is for a Product under Head Client, status of the Registration should be passed as ACTIVE, the indicator for security admin should be passed as True so that Product Specific Registration (if user has) can be returned and finally the uid for selected user and the current user should be same.

**Query 7**: Query that can be run to fetch Registrations for a Product under a Sub Client folder. (strProductRegUnderSubClientQuery):

strProductRegUnderSubClientQuery= CALL getManageRegQueryForFolderType method passing all the argument but the folder type argument should be ‘PRODUCT’, sort type as ‘5’, argument blnIsForHeadClient as ‘false’ indicating the registration is for a Product under Sub Client, status of the Registration should be passed as ACTIVE, the indicator for security admin should be passed as True so that Product Specific Registration (if user has) can be returned and finally the uid for selected user and the current user should be same.

The rest of the logic to generate the My Registration query will follow the same logic as the getManageRegistrations method above.

#### getManageRegQueryForFolderType

The method will be called by both My Registration and Manage Registration search functionality. The method, based on the supplied Search Criteria, Filter conditions and the folder type will generate query that can be used to retrieve Registration information for a user on a folder type with in the DMS.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idClient | IDfId | IN | The Client id of the Head Client on which the Manage Registration search is done. The Client id or the Selected user name must be entered. |
| strClientCode | String | IN | The Client code of the Head Client. The value will be set to null if it is blank. |
| strSelectedUser | String | IN | The uid of the user whose Registration information is searched for. The Client id or the Selected user name must be entered. |
| strStatus | String | IN | The Status of the Registration searched for (Active or Expired). The value can be null. |
| strRestrictedFolderName | String | IN | The name of the Restricted folder whose user Registration details are searched for. The value will be set to null if it is blank. If it is not blank the strRestrictedFolderFilterType should not be blank. |
| strRestrictedFolderFilterType | String | IN | Restricted folder name search criteria. Possible values are 'Contains', 'Begins with' and 'Ends with'. The value will be set to null if it is blank. |
| strRestrictedFolderType | String | IN | The folder type of the Restricted folder being searched for. The valid values are SUB\_CLIENT, JOB, PROJECT or RESTICT. The value can be null |
| strFilterName | String | IN | The name of the Search Result filter. The value will be set to null if it is blank. If filter name is not blank, the filter value should not be blank. |
| strFilterValue | String | IN | The corresponding Search Result filter value. The value will be set to null if it is blank. |
| strCurrentUser | String | IN | The logged in user name. The value will be set to null if it is blank. |
| blnIsSecurityAdmin | Boolean | IN | Indicator to say if the logged in user is a Security Admin within DMS. The default value will be false. |
| strFolderType | String | IN | The folder type for which the Registration search query is required. Valid values are HEAD\_CLIENT, SUB\_CLIENT, JOB, and PROJECT AND PRODUCTS. The value cannot be null. |
| strSortType | String | IN | The sort order value. The value cannot be null |
| blnIsForHeadClient | String | IN | The value if TRUE indicates that the Registration information is required for folders under Head Client and a value FALSE will indicate that the Registration information is for folders under Sub Client. |
| strManageRegDQL | String | OUT | The final generated query that can be used to populate Registration information for a user on a selected folder type. |

##### Pseudocode

StringBuffer strManageRegDQL // initialise the final Query String

//Get the Query headers and append to the final query

IF strFolderType ==PRODUCT

Get the Product specific query header String by calling the method getManageProductSpecificHeaderQuery and passing the sort type and folder type.

Append the string to the final query strManageRegDQL.

ELSE

// if the folder type is not Product

CALL method getManageRegHeaderQuery by passing the folder type and the sort type and get the query string.

Append the generated string to the final query strManageRegDQL.

END IF

//Get query part for Registration status

CALL method getManageStatusQuery to get Query string for Registration status and append to the final query.

IF blnIsForHeadClient == TRUE

//Registration search is for Folder under the Head Client

CALL method getManageRegFolderUserQueryForHeadClient and get the query string.

Append the query string to the final query.

ELSE

//Registration search is for folder under Sub Client

CALL method getManageRegFolderUserQueryForSubclient and get the query string.

Append the query string to the final query.

END IF

IF blnIsSecurityAdmin == FALSE

//get the Approver Registration query part

CALL getManageRegforApproversQuery and get the query string.

Append the Query string to the final query.

END IF

IF strRestrictedFolder != NULL ll AND strFolderType != HEAD\_CLIENT AND strFolderType != PRODUCT

//Get the query part for restricted level folder Registration search

CALL getManageRegAdditionalFilterQuery to get query string to get Registration information on Restricted folder.

Append the query string to the final query.

END IF

//Check for additional filter and get query string to filter the Registration result based on filters

IF strFilterName != NULL AND strFilterName == Security\_Role OR strFilterName == Actioned\_by OR strFilterName == Expiery\_Date

CALL getManageRegAdditionalFilterQuery to get the query string for additional filter criterias to be added to the Registration search.

END IF

RETURN final Query strDQL.

##### Construction Hit:

By changing the imput parameter strObjectType and blnIsForHeadClient the method can generate queries to get Registration information for Job, Project and Product folders under both Head Client and Sub Client.

#### getRegistrationforFolderPathQuery

The Method is called when the Registration search results are sorted based on the folder path. This will return a query that can be executed to get Registration information for a user based on the folder path.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strFolderPath | String | IN | The folder path selected in the search results window. |
| strStatus | String | IN | The status of the Registration searched. Can be Active or Inactive. |
| strUserName | String | IN | The searched users uid |
| blnIsSecurityAdmin | Boolean | IN | Indicator if the user is a Security Admin |
| strDQL | String | OUT | The query part for status Registration |

##### Pseudocode

Get the r\_object\_id and r\_object\_type from dm\_folder by passing folder path.

IF r\_object\_type == TLS\_CLIENT

IF the object is a Client Folder

strObjectType =Sub\_Client

strTypeSort = 2

ELSE IF the object is Head Client

strObjectType =Head\_Client

strTypeSort = 1

CALL getManageRegProductSpecificHeader passing strObjectType and strTypeSort and get the header query string

ELSE IF r\_object\_type == tls\_job

strObjectType =Job

strTypeSort = 3

CALL getManageRegProductSpecificHeader passing strObjectType and strTypeSort and get the header query string

ELSE IF r\_object\_type == tls\_project

strObjectType =Project

strTypeSort = 4

CALL getManageRegProductSpecificHeader passing strObjectType and strTypeSort and get the header query string

ELSE IF r\_object\_type == tls\_folder

strObjectType =Product

strTypeSort = 5

CALL getManageRegProductSpecificHeader passing strObjectType and strTypeSort and get the header query string

END IF

Append the generated header portion of the Query to the final query.

//Get query part for Registration status

CALL method getManageStatusQuery to get Query string for Registration status and append to the final query.

IF blbIsSecurityAdmin == False

//Append the query for Approvers Registration

CALL getManageRegforApproversQuery and get the query string.

Append the Query string to the final query.

ENDIF

RETURN final Query strDQL.

#### getManageRegProductSpecificHeaderQuery

This function, for a Product Specific Registration search will return the final Registration query’s header portion.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strFolderType | String | IN | The type of the folder whose Registration search is made. |
| strSortType | String | IN | The sort type value. |
| strDQL | String | OUT | The header portion of the final Registratioon Query. |

##### Pseudocode

The method will return the header part as under:

Return “SELECT r\_object\_id, t\_folder\_id, t\_user\_name, t\_approver, t\_status, t\_expiry\_date,

t\_approver\_status, f.r\_folder\_path, +“ strFolderType ”AS object\_type,

'strSortType' AS type\_sort, u.t\_full\_name

FROM tls\_registration r, dm\_dbo.tls\_product\_name\_view f,

WHERE r.t\_folder\_id = f.t\_product\_id

AND r.t\_user\_name = u.user\_name”

#### getManageRegHeaderQuery

This function, for a Head Client, Sub Client, Job and Project Registration search will return the final Registration query’s header portion.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strFolderType | String | IN | The type of the folder whose Registration search is made. |
| strSortType | String | IN | The sort type value. |
| strDQL | String | OUT | The header portion of the final Registratioon Query. |

##### Pseudocode

The method will return the header part as under:

Return “SELECT r\_object\_id, t\_folder\_id, t\_user\_name, t\_approver, t\_status, t\_expiry\_date,

t\_approver\_status, f.r\_folder\_path, +“ strFolderType ”AS object\_type,

'strSortType' AS type\_sort, u.t\_full\_name

FROM tls\_registration r, tls\_user u, dm\_dbo.tls\_folder\_path fp

WHERE r.t\_folder\_id = fp.r\_object\_id

AND r.t\_user\_name = u.user\_name”

#### getManageRegStatusQuery

The Method will return Query part to fetch Active or Expired Registration.. If ALL is passed in then the method will not append anything to the query string.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strStatus | String | IN | The status of the Registration searched. Can be Active or Expired |
| strDQL | String | OUT | The query part for status Registration |

##### Pseudocode

IF strStatus == ACTIVE

RETURN “AND t\_status=’strSatus’

ENDIF

ELSE IF

RETURN “AND t\_status LIKE ‘strStatus%’ ”

ENDIF

#### getManageRegFolderUserQueryForHeadClient

The method will return Query part that can be used to populate the Registration for a user on Job, Project and Product folders under a Head Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idClient | IDfid | IN | The Head Clients Client id |
| strSelectedUser | String | IN | The user whose Registration information is searched for under a Head Client folder. |
| strFolderType | String | IN | The folder type can be Head Client, Sub Client, Job, Project and Product. |
| strDQL | String | OUT | The query part for folder user query Registration |

##### Pseudocode

None shown

#### getManageRegFolderUserQueryForSubClient

The method will look for Sub Client folder under the given Head Client and will return query part that can be executed to get Registration information for a user on the Job, Project and Product folder under the Sub Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strClientCode | IDfid | IN | The Head Clients Client code. The method will use the Client code to check if the Head Client has a Sub Client folder under it. |
| strSelectedUser | String | IN | The user whose Registration information is searched for under a Head Client folder. |
| strFolderType | String | IN | The folder type can be Head Client, Sub Client, Job, Project and Product. |
| strDQL | String | OUT | The query part that can be used to get Registration information a user is having on a Job, Project and Product folder under a Sub Client. |

##### Pseudocode

None shown

#### getManageRegforApproversQuery

The method will only be called when the logged in user is a Registration Approver with in DMS. The method will return query part that can be used to retrieve Registrations that the current user is an Approver for.Te method will be called only when the logged in user is not a Security Admin.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strCurrentUser | String | IN | The currently logged in user who is a Registration Approver within DMS. |
| strFolderType | String | IN | The folder type can be Head Client, Sub Client, Job, Project and Product. |
| strDQL | String | OUT | The query part that can be used to get Approver Registration information the logged in user is having on a Head Client, Sub Client, Job or Project folder. |

##### Pseudocode

None shown

#### getManageRegRestrictedFolderQuery

The Method will return query part that can be used to apply Restricted folder name search filter criteria. The method will not be called if the folder is a Head Client or a Product.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strRestrictedFolderName | String | IN | The name of the Restricted folder being searched. |
| strRestrictedFolderFilterType | String | IN | The indicator to show if the search is to be made with ‘Begins with’, ‘Contains’,’Starts with’or ‘Ends with’ the given folder name. |
| strFolderType | String | IN | The Restricted folder type being searched. All the folder types except Products and Head Client can be a valid argument value. |
| strDQL | String | OUT | The query part that can be used to get Approver Registration information the logged in user is having on a Head Client, Sub Client, Job or Project folder. |

##### Pseudocode

None shown

##### Construction Hint:

The method will search on either the 'title' or 'object\_name' field, depending on the value within the configuration item '/REG\_W01/RestrictedSearchAttribute'.

#### getManageRegAdditionalFilterQuery

The method will be called when the Registration search results are filtered on Security Role, Actioned By or Expiery date filters. The method will return query part so that the Registration information can be filterd based on above mentioned filter criteria.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strFilterName | String | IN | The name of the filter being used to filter the Registration information. |
| strFilterValue | String | IN | The corresponding value of the selected filter. |
| strDQL | String | OUT | The query part that can be used to get Registration information based on the filter criteria. |

##### Pseudocode

None shown

#### getFoldersUserIsAutoApproverFor

This function will return folders which the logged on user is an Auto Approver for

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstRegistrations | IDfList | OUT | A list of the object ids of all Registrations the logged in user can approve. |

##### Pseudocode

StritrUserName = msessCurrent.getLoginUserName();

String strQuery = "";

if(isRegistrationAdmin(strUserName)) THEN

strQuery = "SELECT r\_object\_id as objectid FROM tls\_registration”

ELSE{

strQuery = "SELECT r\_object\_id as objectid

FROM tls\_registration reg1

WHERE exists (SELECT 1 FROM tls\_registration reg2

WHERE reg2.t\_user\_name = '" + strUserName +

AND reg2.t\_approver\_status != '" TlsRegistration.APPROVER\_STATUS\_AUTO

AND reg1.t\_folder\_id = reg2.t\_folder\_id)

END IF

return strQuery);

#### getFoldersUserIsAutoApproverForQuery

This function will return a query string of querying folder type and path of folders which the specified user is an auto-approver for.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strUserName | String | IN | The user who is an auto-approver for the folders being queried. |
| strDQL | String | IN | Query to get folder type and path of folders which the specified user is an auto-approver for. |

##### Pseudocode

Retrun query string Number 15 in Section [3.2](#_Queries) for the user specified in strUserName.

#### canAccess

This function will return whether or not the user can access a given folder. This method assumes that the folder passed in is a Head Client or Confidential folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolderId | IDfId | IN | Object id of the folder to check if the user has access to it. |
| blnCanAccess | Boolean | OUT | Whether or not the user has access to the folder. |

##### Pseudocode

*None Shown*

#### canAddDelegatedApprovers

This function will return whether or not more Delegated Approvers can be added to the given folder. This checks the current number of Delegated Approvers for the folder, and compares with the value of the configuration item “/Registration/DelegatedApproverLimit”.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolderId | IDfId | IN | Object id of the folder to check if more Delegated Approvers can be added. |
| blnCanAddDelegatedApprovers | Boolean | OUT | Whether or not more Delegated Approvers can be added to the folder. |

##### Pseudocode

*None Shown*

#### getAllApproversQuery

This function will return a list of the ids for all the users who are either Auto or Delegated Approvers for a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolderId | IDfId | IN | Object id of the folder to return all Approvers for. |
| strarrHeadings | String[] | IN | Headings of columns required |
| alstApprovers | ArrayList | OUT | An array list containing the specified information of all Auto or Delegated Approvers for a folder. |

##### Pseudocode

Ensure r\_object\_id has been passed in as a strarrHeading at index 0. If not, add / move it.

*Setup the query to select distint r\_object\_id, plus any additional headings*

SET strFirstLine = “”;

For each string in strarrHeadings Do

IF string = r\_object\_id Then

strFirstLine = ‘distinct (r\_object\_id) , ’ + strFirstLine

ELSE strFirstLine += string

END IF

END FOR

SET strQuery = SELECT ‘<strFirstLine>’

FROM tls\_user usertable

WHERE exists ( SELECT 1

FROM tls\_registration reg

WHERE reg.t\_user\_name = usertable.user\_name

AND t\_folder\_id = ‘<idFolder>’

AND t\_approver\_status != APPROVER\_STATUS\_NONE);

RETURN strQuery

#### getAllApproversAndRegistrationsQuery

This function will return a query which may be used to get the users who are Auto and Delegated Approvers for a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | Object id of the folder to return all Approvers for. |
| strarrHeadings | String[] | IN | Names of tls\_user and tls\_registration attributes to be included in the query. |
| strQuery | String | OUT | A string specifying the DQL to execute to get all Approvers and their registrations. |

##### Pseudocode

*Setup the query to select distinct r\_object\_id, plus any additional headings*

SET strFirstLine = ‘distinct (r\_object\_id) '

For each string in strarrHeadings Do

IF string != r\_object\_id Then

strFirstLine += string

END IF

END FOR

SET strQuery = SELECT ‘<strFirstLine>’

FROM tls\_user usertable, tls\_registration reg

WHERE reg.t\_user\_name = usertable.user\_name

AND reg.t\_folder\_id = '<idFolderID>'

AND reg.t\_approver\_status != 'NONE'

ORDER BY usertable.t\_full\_name;

RETURN strQuery

#### getAllRegsIcanApprove

This function will return a list of the object ids of all the Registrations that the currently logged in user is able to approve.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstRegistrations | IDfList | OUT | A list of the object ids of all Registrations the logged in user can approve. |

##### Pseudocode

*None Shown*

#### getAllRegsIcanApproveForFolder

This function will return a list of the object ids of all the Registrations that the currently logged in user is able to approve within the given Head Client folder regardless of the Registration status.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolderId | IDfId | IN | The object id of the Head Client folder to return all Registrations for. |
| lstRegistrations | IDfList | OUT | A list of the object ids of all Registrations the logged in user can approve given the search criteria. |

##### Pseudocode

*None Shown*

#### ~~getAllRegsIcanApproveForUserQuery~~

~~This function will return a query that can be run to return a list of the object ids of all the Registrations that the currently logged in user is able to approve for the selected user regardless of the Registration status. The returned query can also be Restricted to look up registrations for a particular folder level.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~strRegistrationUserName~~ | ~~String~~ | ~~IN~~ | ~~The passed in user name~~ |
| ~~strStatus~~ | ~~String~~ | ~~IN~~ | ~~Status of the passed in user.~~ |
| ~~strWhereClause~~ | ~~String~~ | ~~IN~~ | ~~Where Clause.~~ |
| ~~strQuery~~ | ~~String~~ | ~~OUT~~ | ~~A query that can be run to return a list of the object ids of all Registrations the logged in user can approve given the search criteria.~~ |

##### ~~Pseudocode~~

~~strUserName = session.getLoginUserName()~~

~~IF isRegistrationAdmin(strUserName) THEN~~

~~strQuery = "SELECT DISTINCT " +~~ *~~SELECT\_COLUMNS~~* ~~+~~

~~"FROM " +~~ *~~FROM\_TABLES~~* ~~+ ", tls\_client c " +~~

~~"WHERE t\_user\_name = '" + strRegistrationUserName + "'" +~~

~~strStatusQualifier +~~

~~getWhereClause(strWhereClause) +~~

*~~“AND R.T\_FOLDER\_ID = F.R\_OBJECT\_ID AND R.T\_USER\_NAME = U.USER\_NAME AND F.I\_POSITION = -1”~~* ~~+~~

~~"AND ((r\_object\_type='tls\_client' AND~~

~~r.t\_folder\_id=c.r\_object\_id and f.r\_object\_id=c.r\_object\_id) " +~~

~~"OR (NOT(f.r\_object\_type='tls\_client') AND~~

~~r.t\_folder\_id=f.r\_object\_id AND t\_head\_client\_code is~~

~~nullstring))" ;~~

~~ELSE~~

~~strQuery = "SELECT DISTINCT " +~~ *~~SELECT\_COLUMNS~~* ~~+~~

~~"FROM " +~~ *~~FROM\_TABLES~~* ~~+ ", tls\_client c " +~~

~~"WHERE exists (SELECT 1 FROM tls\_registration reg2 " +~~

~~"WHERE reg2.t\_user\_name = '" + strUserName +~~

~~"' AND (reg2.t\_approver\_status != '" +~~

~~TlsRegistration.~~*~~APPROVER\_STATUS\_NONE~~* ~~+~~

~~"' OR c.t\_create\_reg!='APPROVER') AND r.t\_folder\_id = reg2.t\_folder\_id) " + "AND t\_user\_name = '" + strRegistrationUserName + "'" + strStatusQualifier + getWhereClause(strWhereClause) +~~ *~~“AND R.T\_FOLDER\_ID = F.R\_OBJECT\_ID AND R.T\_USER\_NAME = U.USER\_NAME AND F.I\_POSITION = -1”~~* ~~+ " AND ((r\_object\_type='tls\_client' AND~~

~~r.t\_folder\_id=c.r\_object\_id and f.r\_object\_id=c.r\_object\_id) " +~~

~~"OR (NOT(f.r\_object\_type='tls\_client') AND~~

~~r.t\_folder\_id=f.r\_object\_id AND t\_head\_client\_code is~~

~~nullstring))" ;~~

~~END IF~~

~~Return strQuery~~

##### ~~Construction Hint~~

~~May require changes to Registration Admin section of method to cater for Product Specific Access.~~

#### getSubFoldersUserCanApproveQuery

This function will return a list of ~~object ids~~ Folder ids, Folder path and the Folder type for all the folders that the logged in user is an Approver for within a given Client folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idClient | IDfId | IN | The object id of the Client folder. |
| strDQL | String | OUT | The query string that can be used to retrieve folder information the logged in user is an Approver for under a Client. |

##### Pseudocode

STEP 1: Get the Client code by calling method getClientCode and passing idClient.

STEP 2: Check if the logged in user is an approver or a Security admin within DMS.

STEP 3: Call method GenerateSubFolderUserCanApproveQuery by passing the Client code, Client Id, The logged in user name folder type as Head Client , an indicator if the user is a Security Admin and an indicator that its for Folders under Head Client. The method will check if the logged in user is an approver on the Head Client, if he is an Approver the method will return query that can be used to retrieve Head Client folder information.

STEP 3: Call method GenerateSubFolderUserCanApproveQuery by passing the Client code, Client Id, The logged in user name, folder type as Head Client, an indicator if the user is a Security Admin and an indicator that it’s for folders under Head Client. The method will check if the logged in user is an approver on the Head Client, if he is an Approver the method will return query that can be used to retrieve Head Client folder information.

STEP 4: Call method GenerateSubFolderUserCanApproveQuery by passing the Client code, Client Id, The logged in user name, folder type as Job, an indicator if the user is a Security Admin and an indicator that it’s for folders under Head Client. The method will check if the logged in user is an approver on Job folders under Head Client, if he is an Approver the method will return query that can be used to retrieve Job folder information.

STEP 5: Call method GenerateSubFolderUserCanApproveQuery by passing the Client code, Client Id, The logged in user name, folder type as Project, an indicator if the user is a Security Admin and an indicator that it’s for folders under Head Client. The method will check if the logged in user is an approver on Projects folder under Head Client, if he is an Approver the method will return query that can be used to retrieve Project folder information.

STEP 6: Call method GenerateSubFolderUserCanApproveQuery by passing the Client code, Client Id, The logged in user name, folder type as Sub Client, an indicator if the user is a Security Admin and an indicator that it’s for folders under Sub Client. The method will check if the logged in user is an approver on Sub Client folder, if he is an Approver the method will return query that can be used to retrieve Sub Client folder information.

STEP 7: Call method GenerateSubFolderUserCanApproveQuery by passing the Client code, Client Id, The logged in user name, folder type as Job, an indicator if the user is a Security Admin and an indicator that it’s for Job folders under Sub Client. The method will check if the logged in user is an approver on Sub Client folder, if he is an Approver the method will return query that can be used to retrieve Job folder information.

STEP 8: Call method GenerateSubFolderUserCanApproveQuery by passing the Client code, Client Id, The logged in user name, folder type as Job, an indicator if the user is a Security Admin and an indicator that it’s for Job folders under Sub Client. The method will check if the logged in user is an approver on Sub Client folder, if he is an Approver the method will return query that can be used to retrieve Job folder information.

RETURN the union of query generated in Step 3, Step 4, Step 5, Step 6, Step 7 and Step 8

~~SET strUserName = username of logged in user~~

~~SET strClientObjectId = ID of idClient~~

~~IF (isRegistrationAdmin(strUserName)) Then~~

~~strQuery =~~

~~(Select r\_object\_id as objectid~~

~~FROM tls\_client~~

~~WHERE t\_confidential = True~~

~~AND folder(id(<'strClientObjectId'>),descend))~~

~~UNION~~

~~(Select t\_product \_id as objectid~~

~~FROM tls\_job~~

~~WHERE t\_product\_group\_access = True~~

~~AND folder(id(<'strClientObjectId'>),descend))~~

~~UNION~~

~~(Select r\_object\_id as objectid~~

~~FROM tls\_Job~~

~~WHERE t\_confidential = True~~

~~AND folder(id(<'strClientObjectId'>),descend))~~

~~UNION~~

~~(Select r\_object\_id as objectid~~

~~FROM tls\_Project~~

~~WHERE t\_confidential = True~~

~~AND folder(id(<'strClientObjectId'>),descend))~~

~~UNION~~

~~(select r\_object\_id as objectid from tls\_client~~

~~WHERE (t\_confidential = True OR t\_head\_client\_code = ' ')~~

~~AND r\_object\_id = <'strClientObjectId '>)~~

~~ELSE~~

~~strQuery =~~

~~(select t\_folder\_id as objectid~~

~~FROM tls\_registration reg~~

~~WHERE t\_approver\_status != <'TlsRegistration.APPROVER\_STATUS\_NONE '>~~

~~AND t\_user\_name = <'strUserName'>~~

~~AND exists (select 1~~

~~FROM tls\_client jb~~

~~WHERE reg.t\_folder\_id = jb.r\_object\_id AND folder(id(<'strClientObjectId'>),descend))~~

~~UNION~~

~~(select t\_folder\_id as objectid~~

~~FROM tls\_registration reg~~

~~WHERE t\_approver\_status != <'TlsRegistration.APPROVER\_STATUS\_NONE'>~~

~~AND t\_user\_name = <'strUserName '>~~

~~AND exists (select 1~~

~~FROM tls\_Job jb~~

~~WHERE reg.t\_folder\_id = jb.r\_object\_id AND folder(id(<'strClientObjectId'>),descend)))~~

~~UNION~~

~~(select t\_folder\_id as objectid )~~

~~FROM tls\_registration reg~~

~~WHERE t\_approver\_status != <'TlsRegistration.APPROVER\_STATUS\_NONE'>~~

~~AND t\_user\_name = <'strUserName '>~~

~~AND exists (select 1~~

~~FROM tls\_Project jb~~

~~WHERE reg.t\_folder\_id = jb.r\_object\_id AND folder(id(<'strClientObjectId'>),descend)))~~

~~UNION~~

~~(select t\_folder\_id as objectid~~

~~FROM tls\_registration reg~~

~~WHERE t\_approver\_status != <'TlsRegistration.APPROVER\_STATUS\_NONE'>~~

~~AND t\_user\_name = <'strUserName'>~~

~~AND t\_folder\_id = <'strClientObjectId '>~~

~~UNION~~

~~t\_user\_name = <'strUserName'>~~

~~AND exists (select 1~~

~~FROM tls\_client jb~~

~~WHERE reg.t\_folder\_id = jb.r\_object\_id AND~~

~~( jb.t\_create\_reg ==nullstring or jb.t\_create\_reg = 'ALL' )~~

~~AND folder(id(<'strClientObjectId'>),descend))~~

~~UNION~~

~~(select t\_folder\_id as objectid~~

~~FROM tls\_registration reg~~

~~WHERE t\_approver\_status !=~~

~~AND t\_user\_name = <'strUserName '>~~

~~AND exists (select 1~~

~~FROM tls\_Job jb~~

~~WHERE reg.t\_folder\_id = jb.r\_object\_id AND jb.t\_create\_reg = 'ALL' AND folder(id(<'strClientObjectId'>),descend)))~~

~~UNION~~

~~(select t\_folder\_id as objectid )~~

~~FROM tls\_registration reg~~

~~WHERE t\_approver\_status !=~~

~~AND t\_user\_name = <'strUserName '>~~

~~AND exists (select 1~~

~~FROM tls\_Project jb~~

~~WHERE reg.t\_folder\_id = jb.r\_object\_id AND jb.t\_create\_reg = 'ALL' AND folder(id(<'strClientObjectId'>),descend)))~~

~~UNION~~

~~(select t\_folder\_id as objectid~~

~~FROM tls\_registration reg~~

~~WHERE t\_approver\_status !=~~

~~AND t\_user\_name = <'strUserName'>~~

~~AND t\_folder\_id = <'strClientObjectId '>)~~

~~END IF~~

~~RETURN strQuery~~

##### ~~Construction Hint~~

* + - 1. ~~Copy method for getSubFoldersUserCanApprove, unioning the subqueries.~~
      2. ~~This method is used by following RegsIcanApprove methods.~~
      3. ~~If the blnUsedInInStatement = true, the subqueries should be connected using “[strInStatement] IN (“ instead of UNION, because unioning subqueries does not work in the In-Clasue.~~

#### generateSubFoldersUsersCanApproveQuery

This function will will return a Query string that can be used to retrieve folder information for the folder on which the user is an Approver for.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idClient | IDfId | IN | The Head clients Client id |
| strClientCode | String | IN | The Head Client’s Client code. It is used to Sub Clients for the Head Client. |
| strUserName | String | IN | The logged in users uid. |
| strFolderType | String | IN | Folder type of the folder. |
| strTypeSort | String | IN | Sort type corresponding to the folder type. |
| blnIsForHeadClient | Boolean | IN | Indicator to show if the folder is inder a Head Client or a Sub Client. |
| blnIsSecurityAdmin | Boolean | IN | Indicator if the user is a Security Admin or a normal user. |
| strDQL | String | OUT | The generated quey can be used to retrieve folder information about all the folders the user is an approver for under the Head Client. |

##### Pseudocode

StringBuffer strDQL = new StringBuffer();

//Append the header part of the Query

IF blnIsForHeadClient ==TRUE

//The folder is under Head Client so get query portion to generate query part to fetch //Registration for folders under Head Client.

CALL method getManageRegForUserQueryForHeadClient by passing the Client id, the logged in users uid and the folder type.

Append the returned Query to the final query

ELSE IF blnIsForHeadClient ==FALSE

//The folder is under Sub Client so get query portion to generate query part to fetch //Registration for folders under Sub Client.

CALL method getManageRegForUserQueryForSubClient by passing the Client id, the logged in users uid and the folder type.

Append the returned query to the final query.

END IF

IF blnIsSecurityAdmin == FALSE

//The user is not a Security Admin, then get query portion for getting Approvers //Registration

CALL method getManageRegforApproversQuery by passing logged in user name and folder type.

Append the returned query to the final query.

END IF

#### ~~getAllRegsIcanApproveForFolderQuery~~

~~This function will return a query that can be run to return a list of the object ids of all the Registrations that the currently logged in user is able to approve within the given Head Client folder regardless of the Registration status. The returned query can also be Restricted to look up registrations for a particular folder level.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~idFolderId~~ | ~~IDfId~~ | ~~IN~~ | ~~The object id of the Head Client folder to return all Registrations for.~~ |
| ~~strRestrictFolder~~ | ~~String~~ | ~~IN~~ | ~~The folder level to retrieve Registrations for.~~ |
| ~~strQuery~~ | ~~String~~ | ~~OUT~~ | ~~A query that can be run to return a list of the object ids of all Registrations the logged in user can approve given the search criteria.~~ |

##### ~~Pseudocode~~

~~SET strSubFoldersQuery = getSubFoldersUserCanApproveQuery(idFolder)~~

~~IF isRegistrationAdmin(strUserName) Then~~

~~strQuery = “ SELECT r\_object\_id as objectid~~

~~FROM tls\_registration~~

~~WHERE t\_folder\_id in <’ strSubFoldersQuery ‘>;”~~

~~ELSE~~

~~strQuery = “ SELECT r\_object\_id as objectid~~

~~FROM tls\_registration reg1~~

~~WHERE exists (SELECT 1 FROM tls\_registration reg2~~

~~WHERE reg2.t\_user\_name = <'strUserName'>~~

~~AND reg2.t\_approver\_status != None~~

~~AND reg1.t\_folder\_id = reg2.t\_folder\_id)~~

~~AND t\_folder\_id in <’ strSubFoldersQuery ‘> ;”~~

~~RETURN strQuery~~

##### ~~Construction Hint~~

1. ~~Copy these queries from the current getAllRegsIcanApproveForFolder method, replacing the ‘AND t\_folder\_id = folder id’ clause with ‘AND t\_folder\_id in folder list’~~
2. ~~In strSubFoldersQuery we need to make join with getSubClientsUserCanApproveQuery, getJobsUserCanApproveQuery, getProjectsUserCanApproveQuery and getHeadClientsUserCanApproveQuery depending on the passed in Restricted folder level.~~

#### getAllRegsIcanApproveForFolder

This function will return a list of the object ids of all the Registrations that the currently logged in user is able to approve within a given Head Client folder for the given Registration status. The returned list can also be limited to regirstrations for a particular folder level.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolderId | IDfId | IN | The object id of the Head Client folder to return all Registrations for. |
| strStatus | String | IN | The status of the Registrations to return. |
| strRestrictFolder | String | IN | The folder level to retrieve Registrations for. |
| lstRegistrations | IDfList | OUT | A list of the object ids of all Registrations the logged in user can approve given the search criteria. |

##### Pseudocode

*None Shown*

#### ~~getAllRegsIcanApproveForFolderQuery~~

~~This function will return a query that can be run to return a list of the object ids of all the Registrations that the currently logged in user is able to approve within a given Head Client folder for the given Registration status. The returned query can also be Restricted to look up regirstrations for a particular folder level.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~idFolderId~~ | ~~IDfId~~ | ~~IN~~ | ~~The object id of the Head Client folder to return all Registrations for.~~ |
| ~~strStatus~~ | ~~String~~ | ~~IN~~ | ~~The status of the Registrations to return.~~ |
| ~~strRestrictFolder~~ | ~~String~~ | ~~IN~~ | ~~The folder level to retrieve Registrations for.~~ |
| ~~strQuery~~ | ~~String~~ | ~~OUT~~ | ~~A query that can be run to return a object ids of all Registrations the logged in user can approve given the search criteria.~~ |

##### ~~Pseudocode~~

~~SET strSubFoldersQuery = getSubFoldersUserCanApproveQuery(idFolder)~~

~~IF isRegistrationAdmin(strUserName) Then~~

~~strQuery = " SELECT r\_object\_id as objectid~~

~~FROM tls\_registration~~

~~WHERE t\_folder\_id in <‘ strSubFoldersQuery ‘>~~

~~AND t\_status like '" <’strStatus’>"%'"~~

~~ELSE~~

~~strQuery = “ SELECT r\_object\_id as objectid~~

~~FROM tls\_registration reg1~~

~~WHERE exists (SELECT 1 FROM tls\_registration reg2~~

~~WHERE reg2.t\_user\_name = <'strUserName'>~~

~~AND reg2.t\_approver\_status != None~~

~~AND reg1.t\_folder\_id = reg2.t\_folder\_id)~~

~~AND t\_folder\_id in <’ strSubFoldersQuery ‘>~~

~~AND t\_status like <'strStatus'> %";~~

~~RETURN strQuery~~

##### ~~Construction Hint~~

1. ~~Copy these queries from the current getAllRegsIcanApproveForFolder method, replacing the ‘AND t\_folder\_id = folder id’ clause with ‘AND t\_folder\_id in folder list’~~
2. ~~In strSubFoldersQuery we need to make join with getSubClientsUserCanApproveQuery, getJobsUserCanApproveQuery, getProjectsUserCanApproveQuery and getHeadClientsUserCanApproveQuery depending on the passed in Restricted folder level.~~
3. ~~getAllRegsIcanApproverForFolderAndUser~~

~~This function will return a list of the object ids of all the Registrations that the currently logged in user is able to approve within a given Head Client folder, for a given user regardless of the Registration status.~~

~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~idFolderId~~ | ~~IDfId~~ | ~~IN~~ | ~~The object id of the Head Client folder to return all Registrations for.~~ |
| ~~strUserName~~ | ~~String~~ | ~~IN~~ | ~~The user name to return Registrations for.~~ |
| ~~strRestrictFolder~~ | ~~String~~ | ~~IN~~ | ~~The folder level to retrieve Registrations for.~~ |
| ~~lstRegistrations~~ | ~~IDfList~~ | ~~OUT~~ | ~~A list of the object ids of all Registrations the logged in user can approve given the search criteria.~~ |

##### ~~Pseudocode~~

*~~None Shown~~*

#### ~~getAllRegsIcanApproverForFolderAndUserQuery~~

~~This function will return a query that can be run to find a list of the object ids of all the Registrations that the currently logged in user is able to approve within a given Head Client folder, for a given user regardless of the Registration status.~~

~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~idFolderId~~ | ~~IDfId~~ | ~~IN~~ | ~~The object id of the Head Client folder to return all Registrations for.~~ |
| ~~strUserName~~ | ~~String~~ | ~~IN~~ | ~~The user name to return Registrations for.~~ |
| ~~strRestrictFolder~~ | ~~String~~ | ~~IN~~ | ~~The folder level to retrieve Registrations for.~~ |
| ~~strQuery~~ | ~~String~~ | ~~OUT~~ | ~~A list of the object ids of all Registrations the logged in user can approve given the search criteria.~~ |

##### ~~Pseudocode~~

~~SET strSubFoldersQuery = getSubFoldersUserCanApproveQuery(idFolder)~~

~~IF isRegistrationAdmin(strUserName) Then~~

~~strQuery = " SELECT r\_object\_id as objectid~~

~~FROM tls\_registration~~

~~WHERE t\_folder\_id in <’ strSubFoldersQuery >~~

~~AND t\_user\_name = <'strRegistrationUserName'> “~~

~~ELSE~~

~~strQuery = “ SELECT r\_object\_id as objectid~~

~~FROM tls\_registration reg1~~

~~WHERE exists (SELECT 1 FROM tls\_registration reg2~~

~~WHERE reg2.t\_user\_name = <'strUserName'>~~

~~AND reg2.t\_approver\_status != NONE~~

~~AND reg1.t\_folder\_id = reg2.t\_folder\_id)~~

~~AND t\_folder\_id in <’ strSubFoldersQuery ‘>~~

~~AND t\_user\_name = <'strRegistrationUserName'>;"~~

~~RETURN strQuery~~

##### ~~Construction Hint~~

1. ~~Copy these queries from the current getAllRegsIcanApproveForFolder method, replacing the ‘AND t\_folder\_id = folder id’ clause with ‘AND t\_folder\_id in folder list’~~
2. ~~In strSubFoldersQuery we need to make join with getSubClientsUserCanApproveQuery, getJobsUserCanApproveQuery, getProjectsUserCanApproveQuery and getHeadClientsUserCanApproveQuery depending on passed in Restricted folder level.~~
3. ~~getAllRegsIcanApproverForFolderAndUser~~

~~This function will return a list of the object ids of all the Registrations that the currently logged in user is able to approve within a given Head Client folder, for a given user, for a given Registration status.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |

##### ~~Pseudocode~~

*~~None Shown~~*

#### ~~getAllRegsIcanApproverForFolderAndUserQuery~~

~~This function will return a query that can be run to find a list of the object ids of all the Registrations that the currently logged in user is able to approve within a given Head Client folder, for a given user regardless of the Registration status.~~

~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~idFolderId~~ | ~~IDfId~~ | ~~IN~~ | ~~The object id of the Head Client folder to return all Registrations for.~~ |
| ~~strUserName~~ | ~~String~~ | ~~IN~~ | ~~The user name to return Registrations for.~~ |
| ~~strStatus~~ | ~~String~~ | ~~IN~~ | ~~The status of the Registrations to return.~~ |
| ~~strRestrictFolder~~ | ~~String~~ | ~~IN~~ | ~~The folder level to retrieve Registrations for.~~ |
| ~~strQuery~~ | ~~String~~ | ~~OUT~~ | ~~A query that can be run to return a list of the object ids of all Registrations the logged in user can approve given the search criteria.~~ |

##### ~~Pseudocode~~

~~SET strSubFoldersQuery = getSubFoldersUserCanApproveQuery(idFolder)~~

~~IF isRegistrationAdmin(strUserName) Then~~

~~strQuery = " SELECT r\_object\_id as objectid~~

~~FROM tls\_registration~~

~~WHERE t\_folder\_id in <’ strSubFoldersQuery ‘>~~

~~AND t\_user\_name = <'strRegistrationUserName'>~~

~~AND t\_status like <'strStatus’>% “~~

~~ELSE~~

~~strQuery = “ SELECT r\_object\_id as objectid~~

~~FROM tls\_registration reg1~~

~~WHERE exists (SELECT 1 FROM tls\_registration reg2~~

~~WHERE reg2.t\_user\_name = <'strUserName'>~~

~~AND reg2.t\_approver\_status != NONE~~

~~AND reg1.t\_folder\_id = reg2.t\_folder\_id)~~

~~AND t\_folder\_id in <’ strSubFoldersQuery ‘>~~

~~AND t\_user\_name = <'strRegistrationUserName'>;"~~

~~AND t\_status like <'strStatus’>% “~~

~~RETURN strQuery~~

##### ~~Construction Hint~~

1. ~~Copy these queries from the current getAllRegsIcanApproveForFolder method, replacing the ‘AND t\_folder\_id = folder id’ clause with ‘AND t\_folder\_id in folder list’~~
2. ~~In strSubFoldersQuery we need to make join with getSubClientsUserCanApproveQuery, getJobsUserCanApproveQuery, getProjectsUserCanApproveQuery and getHeadClientsUserCanApproveQuery depending on the passed in Restricted folder level.~~
3. ~~getAllRegsIcanApproverForUser~~

~~This function will return a list of the object ids of all the Registrations that the currently logged in user is able to approve for a given user regardless of the Registration status.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~strUserName~~ | ~~String~~ | ~~IN~~ | ~~The user name to return Registrations for.~~ |
| ~~lstRegistrations~~ | ~~IDfList~~ | ~~OUT~~ | ~~A list of the object ids of all Registrations the logged in user can approve given the search criteria.~~ |

##### ~~Pseudocode~~

*~~None Shown~~*

#### ~~getAllRegsIcanApproverForUser~~

~~This function will return a list of the object ids of all the Registrations that the currently logged in user is able to approve for a given user, for a given Registration status.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~strUserName~~ | ~~String~~ | ~~IN~~ | ~~The user name to return Registrations for.~~ |
| ~~strStatus~~ | ~~String~~ | ~~IN~~ | ~~The status of the Registrations to return.~~ |
| ~~lstRegistrations~~ | ~~IDfList~~ | ~~OUT~~ | ~~A list of the object ids of all Registrations the logged in user can approve given the search criteria.~~ |

##### ~~Pseudocode~~

*~~None Shown~~*

#### getApproversAndRegistrationsByStatusQuery

This function will return a query that can be run to return the specified attributes for the users who are either the Auto Approvers or the Delegated Approvers (this includes both the standard and nominated delegated Approvers) and their registrations for the folder passed in, depending on the approver status passed in.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolderId | IDfId | IN | The folder to return the Approvers for. |
| strApproverStatus | String | IN | The status of the Approvers to return details about. Can be ‘AUTO’ or ‘DELEGATED’ |
| strarrHeadings | String[] | IN | The columns that must be returned by this query. Valid columns are from the tls\_users and tls\_registration table. |
| strQuery | String | OUT | The query that may be executed to retrieve the Approvers and their registrations. |

##### Pseudocode

*Setup the query to select distint r\_object\_id, plus any additional headings*

SET strQuery = “SELECT ”;

For each string in strarrHeadings Do

IF string = r\_object\_id Then

strQuery = ‘distinct (r\_object\_id) , ’ + strQuery

ELSE strQuery += string

END IF

END FOR

*Check whether Auto or Delegated approver registrations are being retrieved.*

IF strApproverStatus = AUTO Then

strQuery += "FROM tls\_user usertable, tls\_registration reg1

WHERE usertable.user\_name = reg1.t\_user\_name

AND exists (SELECT 1 FROM tls\_registration reg WHERE reg.t\_user\_name = usertable.user\_name AND t\_folder\_id = 'idFolderId' AND t\_approver\_status = 'AUTO')”

ELSE IF strApproverStatus = DELEGATED Then

strQuery += “FROM tls\_user usertable, tls\_registration reg1

WHERE usertable.user\_name = reg1.t\_user\_name

AND exists (SELECT 1 FROM tls\_registration reg WHERE reg.t\_user\_name = usertable.user\_name AND t\_folder\_id = 'idFolderId' AND (t\_approver\_status = 'DELEGATED' OR t\_approver\_status = ‘DELGATED-NOMINATED’))”

ELSE

THROW an error saying “Invalid approver status passed in.”

END IF

Return strQuery

#### getApproverUserName

This function will return the list of names for the users who approved a Registration. Currently only ever one name will be returned.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolderId | IDfId | IN | The object id of the folder the Registration is for. |
| strUserName | String | IN | The name of the user who the Registration is for. |
| lstApproverUserName | IDfList | OUT | A list containing the object id of the users who approved the Registration. |

##### Pseudocode

*None Shown*

#### getClientsForLetter

This function will return a list of object ids for all the Client folders that begin with a certain letter.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| chLetter | Character | IN | The letter that all Client folders returned begin with. |
| strCabinetId | String | IN | The id of the cabinet to get all the Clients from. |
| lstClients | IDfList | OUT | A list containing the object id of the Clients. |

##### Pseudocode

*None Shown*

#### getMyRegisitrations

This function will return a list of the object ids of all the Registrations for the currently logged in user regardless of the Registration status.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstRegistrations | IDfList | OUT | A list of the object ids of all the user’s Registrations. |

##### Pseudocode

*None Shown*

#### ~~getMyRegistrationsByStatus~~

~~This function will return a list of the object ids of all the Registrations for the currently logged in user that have a given Registration status.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~strStatus~~ | ~~String~~ | ~~IN~~ | ~~The status of the Registrations to be returned.~~ |
| ~~lstRegistrations~~ | ~~IDfList~~ | ~~OUT~~ | ~~A list of the object ids of all the user’s Registrations for the given status.~~ |

##### ~~Pseudocode~~

*~~None Shown~~*

#### ~~getMyRegistrationsByStatusQuery~~

~~This function will return a query that can be run to return a list of the object ids of all the Registrations for the currently logged in user that have a given Registration status.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~strStatus~~ | ~~String~~ | ~~IN~~ | ~~The status of the Registrations to be returned.~~ |
| ~~lstRegistrations~~ | ~~IDfList~~ | ~~OUT~~ | ~~A query that can be run to return a list of the object ids of all the user’s Registrations for the given status.~~ |

##### ~~Pseudocode~~

~~SET strUsername = Get current user~~

~~strQuery = " SELECT r\_object\_id as objectid~~

~~FROM tls\_registration~~

~~WHERE t\_user\_name = < ‘strUsername’ >~~

~~AND t\_status like <’strStatus’>%";~~

~~Return strQuery~~

##### ~~Construction Hint~~

1. ~~Copy these queries from the current getMyRegisitrations method.~~

#### getSubFoldersUserCanApprove

This function will return a list of object ids for all the folders that the logged in user is an Approver for within a given Client folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idClient | IDfId | IN | The object id of the Client folder. |
| lstFolders | IDfList | OUT | A list of the object ids for the folders that user is an Approver for within the Client. |

##### Pseudocode

*Aivin Update*

#### isApprover

This function will return whether or not the currently logged in user is an Approver (Auto or Delegated) for the given folder. If a Client, Job or Project has ‘All’ in its t\_create\_reg variable then the logged in user is also an approver for the object.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The object id of the folder. |
| blnIsApprover | Boolean | OUT | Whether or not the user is an Approver for the folder. |

##### Pseudocode

*None Shown*

#### isAutoApprover

This function will return whether or not the currently looked in user is an Auto Approver for the given folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The object id of the folder. |
| blnIsAutoApprover | Boolean | OUT | Whether or not the user is an Auto Approver for the folder. |

##### Pseudocode

*None Shown*

#### addLists

This private function will add two lists together.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstA | IDfList | IN | The first list to add together. |
| lstB | IDfList | IN | The second list to add together. |
| concatList | IDfList | OUT | The two lists added together. |

##### Pseudocode

*None Shown*

#### executeQueryCount

This private function will execute a query which contains ‘count(\*) as counter’ within the select statement, and return the results for the counter as an integer.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strQuery | String | IN | The query to execute. |
| intCount | Integer | OUT | The results of the counter within the query. |

##### Pseudocode

*None Shown*

#### executeQueryObjectId

This private function will execute a query which contains ‘r\_object\_id as objectid’ within the select statement and return the results for objectid in a list.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strQuery | String | IN | The query to execute. |
| lstResults | IDfList | OUT | The results of the query within an list. |

##### Pseudocode

*None Shown*

#### isRegistrationAdmin

This private function will return whether or not a given user is a Registration Administrator.

This uses the constant from the TlsRegistration.java class to find out what the Registration Admin group is.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strUserName | String | IN | The name of the user to check if they are Registration Admin. |
| blnIsRegistrationAdmin | Boolean | OUT | Returns whether or not the user is a Registration Admin. |

##### Pseudocode

*None Shown*

#### getNominatedApprovers

This function will execute a query to return list of guids of the nominated delegated approver for the currently logged on user. If the logged in user hasn’t nominated a delegated approver then an empty list will be returned.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strCurrentUser | Strng | IN | The user to retrieve the nominated delegated approver for. |
| lstUserName | IDfList | OUT | The list of guids of the users who are nominated delegated approver for the currently logged on user. An empty list will be return if the user doesn’t have any nominated delegated Approvers. |

##### Pseudocode

*Execute a query to locate the nominated delegated Approvers for the passed in user.*

EXECUTE query “SELECT t\_nominated\_user\_name FROM tls\_nominated\_approver WHERE t\_approver\_user\_name = strCurrentUser”

RETURN the result

#### addNominatedApprover

This function will execute a query to create the nominated delegated Approvers for the passed in user. The actual Nominated Delegated Approver access to the folder will be granted by REG\_B04 – Manage Nominated Delegated Approver Job.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strCurrentUser | String | IN | The user to set the nominated delegated approver for. |
| lstUserName | IDfList | IN | The list of guids of the user who will be set as the nominated delegated approver. |

##### Pseudocode

*Check if the current user already has a record within the tls\_nominated\_approver table.*

Get strUserName from the list and

EXECUTE query “CREATE tls\_nominated\_approver OBJECT

SET t\_approver\_user\_name = strCurrentUser ,

SET t\_nominated\_user\_name = strUserName,

SET t\_nominated\_date = DATE(TODAY) ”

for all user in the list

#### removeNominatedApprover

This function will execute a query to remove the nominated delegated Approvers for the passed in user. The actual Nominated Delegated Approver access to the folder will be revoked by REG\_B04 – Manage Nominated Delegated Approver Job.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strCurrentUser | String | IN | The user to set the nominated delegated approver for. |
| lstUserName | IDfList | IN | The list of guids of the user who will be removed as the nominated delegated approver. |

##### Pseudocode

*Check if the current user already has a record within the tls\_nominated\_approver table.*

Get strUserName from the list and

EXECUTE query “DELETE tls\_nominated\_approver object

WHERE t\_approver\_user\_name = strCurrentUser

AND t\_nominated\_user\_name = strUserName”

for all user in the list

#### getMyRegistrationsByStatusQueryArchive

~~This function will be called to get the registrations of the user filtered by status.This function will return a query that can be run to return a list of the object ids of all the Registrations for the currently logged in user that have a given Registration status.~~

~~Admin users will be able to retrive all the foldrs in line with the active docbase. Non- admin users will retrive only archived folders and not archive shadow folders.~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~strStatus~~ | ~~String~~ | ~~IN~~ | ~~Status of the passed in user.~~ |
| ~~strWhereClause~~ | ~~String~~ | ~~IN~~ | ~~Where Clause.~~ |
| ~~strQuery~~ | ~~String~~ | ~~OUT~~ | ~~A query that can be run to return a list of the object ids of all Registrations the logged in user can approve given the search criteria.~~ |

##### ~~Pseudocode~~

~~strUserName = session.getLoginUserName()~~

~~IF isRegistrationAdmin(strUserName) THEN~~

~~Call getMyRegistrationsByStatusQuery~~

~~ELSE~~

~~This part will use the same logic of getMyRegistrationsByStatusQuery for non-admin users. In addition we will join with the tls\_archive table with the following conditions:~~

~~i) t\_archive\_id of the tls\_archive table will be joined with dm\_folder’s r\_object\_id ii) check whether t\_is\_archived is true in the tls\_archive table.~~

~~END IF~~

~~Return strQuery~~

#### getAllRegsIcanApproveForUserQueryArchive

~~This function will be called to get the registrations the user can approve for the selected user.This function will return a list of the object ids of all the Registrations that the currently logged in user is able to approve within the given Head Client folder regardless of the Registration status.~~

~~Admin users will be able to retrive all the foldrs in line with the active docbase. Non- admin users will retrive only archived folders and not archive shadow folders.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~strRegistrationUserName~~ | ~~String~~ | ~~IN~~ | ~~The passed in user name~~ |
| ~~strStatus~~ | ~~String~~ | ~~IN~~ | ~~Status of the passed in user.~~ |
| ~~strWhereClause~~ | ~~String~~ | ~~IN~~ | ~~Where Clause.~~ |
| ~~strQuery~~ | ~~String~~ | ~~OUT~~ | ~~A query that can be run to return a list of the object ids of all Registrations the logged in user can approve given the search criteria.~~ |

##### ~~Pseudocode~~

~~strUserName = session.getLoginUserName()~~

~~IF isRegistrationAdmin(strUserName) THEN~~

~~Call getAllRegsIcanApproveForUserQuery~~

~~ELSE~~

~~This part will use the same logic of getMyRegistrationsByStatusQuery for non-admin users. In addition we will join with the tls\_archive table with the following conditions:~~

~~i) t\_archive\_id of the tls\_archive table will be joined with dm\_folder’s r\_object\_id ii) check whether t\_is\_archived is true in the tls\_archive table.~~

~~END IF~~

~~Return strQuery~~

#### getAllRegsIcanApproveForFolderQueryArchive

~~This function will be called to get the registrations the user can approve for the selected folder. This function will return a query that can be run to return a list of the object ids of all the Registrations that the currently logged in user is able to approve within the given Head Client folder regardless of the Registration status. The returned query can also be Restricted to look up registrations for a particular folder level.~~

~~Admin users will be able to retrive all the folders in line with the active docbase. Non- admin users will retrive only archived folders and not archived shadow folders.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~idFolder~~ | ~~IDfId~~ | ~~IN~~ | ~~The passed in folder ID~~ |
| ~~strStatus~~ | ~~String~~ | ~~IN~~ | ~~Status of the passed in user.~~ |
| ~~strRestrictedFolder~~ | ~~String~~ | ~~IN~~ | ~~Whether the folder is Restricted~~ |
| ~~strWhereClause~~ | ~~String~~ | ~~IN~~ | ~~Where Clause.~~ |
| ~~strQuery~~ | ~~String~~ | ~~OUT~~ | ~~A query that can be run to return a list of the object ids of all Registrations the logged in user can approve given the search criteria.~~ |

##### ~~Pseudocode~~

~~IF (strRestrictFolder!=null) THEN~~

~~IF strRestrictFolder.equalsIgnoreCase(CLIENT\_FOLDER) THEN~~

~~blnIsFolderClient = true;~~

~~ELSE IF strRestrictFolder.equalsIgnoreCase(JOB\_FOLDER) THEN~~

~~blnIsFolderJob = true;~~

~~ELSE IF strRestrictFolder.equalsIgnoreCase(PROJECT\_FOLDER) THEN~~

~~blnIsFolderProject = true;~~

~~ELSE IF strRestrictFolder.equalsIgnoreCase(RESTRICTED\_FOLDER)~~

~~blnIsFolderRestrict = true;~~

~~END IF~~

~~END IF~~

~~IF AdminUser THEN~~

~~Call getAllRegsIcanApproveForFolderQuery~~

~~ELSE IF (blnIsFolderClient) THEN~~

~~This part will use the same logic of getMyRegistrationsByStatusQuery for non-admin users. In addition we will join with the tls\_archive table with the following conditions:~~

~~i) t\_archive\_id of the tls\_archive table will be joined with dm\_folder’s r\_object\_id ii) check whether t\_is\_archived is true in the tls\_archive table.~~

~~END IF~~

#### getAllRegsIcanApproveForFolderAndUserQueryArchive

~~This function will be called to get the registrations the user can approve for the selected folder and the selected user. This function will return a list of the object ids of all the Registrations that the currently logged in user is able to approve within a given Head Client folder for the given Registration status and for the selected user. The returned list can also be limited to regirstrations for a particular folder level.~~

~~Admin users will be able to retrieve all the folders in line with the active docbase. Non- admin users will retrieve only archived folders and not archived shadow folders.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~IdFolder~~ | ~~IDfId~~ | ~~IN~~ | ~~The passed in folder ID~~ |
| ~~StrStatus~~ | ~~String~~ | ~~IN~~ | ~~Status of the passed in user.~~ |
| ~~strRestrictedFolder~~ | ~~String~~ | ~~IN~~ | ~~Whether the folder is Restricted~~ |
| ~~strWhereClause~~ | ~~String~~ | ~~IN~~ | ~~Where Clause.~~ |
| ~~StrRegistrationUserName~~ | ~~String~~ | ~~IN~~ | ~~User name~~ |
| ~~StrQuery~~ | ~~String~~ | ~~OUT~~ | ~~A query that can be run to return a list of the object ids of all Registrations the logged in user can approve given the search criteria.~~ |

##### ~~Pseudocode~~

~~IF (strRestrictFolder!=null) THEN~~

~~IF (strRestrictFolder.equalsIgnoreCase(CLIENT\_FOLDER)) THEN~~

~~blnIsFolderClient = true;~~

~~ELSE IF (strRestrictFolder.equalsIgnoreCase(JOB\_FOLDER)) THEN~~

~~blnIsFolderJob = true;~~

~~ELSE IF (strRestrictFolder.equalsIgnoreCase(PROJECT\_FOLDER)) THEN~~

~~blnIsFolderProject = true;~~

~~ELSE IF (strRestrictFolder.equalsIgnoreCase(RESTRICTED\_FOLDER)) THEN~~

~~blnIsFolderRestrict = true;~~

~~END IF~~

~~IF (blnIsAdmin) THEN~~

~~Call getAllRegsIcanApproveForFolderAndUserQuery~~

~~ELSE IF (blnIsFolderClient) THEN~~

~~This part will use the same logic of getMyRegistrationsByStatusQuery for non-admin users. In addition we will join with the tls\_archive table with the following conditions:~~

~~i) t\_archive\_id of the tls\_archive table will be joined with dm\_folder’s r\_object\_id ii) check whether t\_is\_archived is true in the tls\_archive table.~~

~~END IF~~

## tls\_regb01\_manage\_Registrations

### ManageRegistrations.java

This server method controls the management of Registrations and Approvers

#### Main

This method is the entry point of the server method and is responsible for validating arguments and setting the return value in the result object. The accessor names that should be included in the ACL should be passed in the t\_param1 attribute of the method result object.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| Object | ID | IN | Contains the ID of the folder to which a Registrations need to be applied. |
| Result\_object | ID | IN | The ID of the result object that should be used to pass object names and return the success value to the Client. |
| Modification\_type | String | IN | Identifies the type of Registration modification that needs to be applied. e.g.  ADD\_REGISTRATION.  REMOVE\_REGISTRATION  REMOVE\_APPROVER  EXTEND\_REGISTRATION  ADD\_DELEGATED\_APPROVER  SET\_AUTO\_APPROVERS |
| Requestor\_name | String | IN | The user requesting the operation |

##### Pseudocode

Check mandatory arguments

CALL process PASSING arguments

#### Process

This method performs the processing of the server method and calls different methods of the RegistrationsLib class.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strObjectID | ID | IN | Contains the ID of the folder to apply security. |
| strRequester | String | IN | The Requestor for the action |
| strUserName | String | IN | The user who the action is being performed (e.g., the user who is being registered) |
| strAction | String | IN | The Action |
| strTimePeriod | String | IN(OPTIONAL) | The time period for the Registration or extension. |
| intResult | Integer | OUT | Identifies whether an ACL was successfully applied to this object.  0 = Success  1 = Unknown failure  *2 = Object selected is not allowed to have folder security applied.*  *3 = Secured type change not allowed.* |

##### Pseudocode

*None Shown*

### RegistrationLib.java

This is an interface class.

### RegistrationLibImpl.java

This is a server side class which implements RegistrationLib and calls the AddRegistration.java, ConfidentialLIb.java, RegistrationApproverLib.java or RemoveRegistration.java class to carry out the required operations.

##### Public Methods

| Name | Input | Output | Description |
| --- | --- | --- | --- |
| addDelegatedApprover | IDfPersistenObject, String, String | None | This calls AddRegistration.java to add a Delegated Approver. |
| addNominatedApproverRegistrations | IDfPersistenObject, String, String | None | This calls AddRegistration.java to create a Nominated Delegated Approver registration. |
| addRegistration | IDfPersistenObject, String, String, Integer | None | This calls AddRegistration.java to create a Registration. |
| cleanUpRegistrations | IDfPersistenObject, String | None | This calls RemoveRegistration.java to delete a Registration Object. |
| copyRegistrations | IDfPersistenObject, String, String | None | This calls AddRegistration to copy Registrations. |
| createConfidentialGroup | IDfId | None | This calls ConfidentialLib.java to create the Registration Group for a folder. |
| deleteRegistrationsForFolder | IDfPersistenObject, String | None | This calls RemoveRegistration.java to delete all Registrations for a folder. |
| deleteNomiantedApproverRegsitration | IDfId | Nona | This calls RemoveRegistration to delete a Nominated Delegated Approver registration. |
| extendRegistrations | IDfPersistenObject, String, Integer | None | This calls AddRegistration.java to extend a Registration. |
| markConfidential | IDfPersistenObject, String | None | This calls ConfidentialLib.java to mark a folder Confidential. |
| moveRegistrations | IDfPersistenObject, String, String | None | This calls AddRegistration.java to move Registrations after a Job Move has taken place. |
| reassignClientRegistrations | IDfId, IDfId, String | None | This calls AddRegistration.java to move Registrations after a Pending Client Reassignment has taken place. |
| removeApprover | IDfPersistenObject, String, String | None | This calls RemoveRegistration.java to remove an Auto Approver from a folder. |
| removeDelegatedApprover | IDfPersistenObject, String, String | None | This calls RemoveRegistration.java to remove a Delegated Approver from a folder. |
| removeRegistration | IDfPersistenObject, String | None | This calls RemoveRegistration.java to expire a Registration. |
| removeRegistration | IDfPersistenObject, String, String | None | This calls RemoveRegistration.java to expire a Registration. |
| removeRegistrationExpired | IDfPersistenObject, String | None | This calls RemoveRegistration.java to expire a Registration. |
| removeRegistrationLeft | IDfPersistenObject, String | None | This calls RemoveRegistration.java to expire a Registration. |
| setAutoApprovers | IDfPersistenObject | None | This calls RegistrationApproverLib.java to set the Auto Approvers for a folder. |
| unmarkConfidential | IDfPersistenObject, String | None | This calls ConfidentialLib.java to unmark a folder as Confidential. |
| cleanupDeletedFolder | IDfId | None | This calls RemoveRegistration.java to clean up all registrations of a deleted Job, Project or deactivated Client/Sub Client. |
| addProductRegistration | String, String | None | This calls AddRegistration.java to add a Registration for Product Specific Access. |
| removeProductRegistration | String, String | None | This calls RemoveRegistration to delete a Registration used for Product Specific Access. |
| addClientDefaultGroupReg | IDfId, String | None | This calls the AddRegistration.java to add a Client Default Group Registration. |

#### addDelegatedApprover

This calls AddRegistration.java to add a Delegated Approver.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to make the user a Delegated Approver to. |
| strUserName | String | IN | The Guid of the user to make a Delegated Approver. |
| strRequestor | String | IN | The user requesting the operation. |

##### Pseudocode

*None Shown*

#### addNominatedApproverRegistration

This calls AddRegistration.java to add a Nominated Delegated Approver.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to make the user a Nominated Delegated Approver to. |
| strUserName | String | IN | The Guid of the user to make a Nominated Delegated Approver. |
| strApprover | String | IN | The user that the Nominated Delegated Approver is being created for. |

##### Pseudocode

*None Shown*

#### deleteNominatedApproverRegistration

This calls RemoveRegistration.java to delete a Nominated Delegated Approver registration object.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfId | IN | Nominated Delegated Approver registration object that will be deleted. |

##### Pseudocode

*None Shown*

#### addRegistration

This calls AddRegistration.java to create a normal Registration to.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to create the Registration to. |
| strUserName | String | IN | The Guid of the user to create a Registration for. |
| strRequestor | String | IN | The user requesting the operation. |
| intDuration | Integer | IN | The duration for the Registration. |

##### Pseudocode

*None Shown*

#### cleanUpRegistrations

This calls RemoveRegistration.java to delete a Registration object.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Registration object that will be deleted. |
| strRequestor | String | IN | The user requesting the operation. |

##### Pseudocode

*None Shown*

#### copyRegistrations

This calls AddRegistration.java to copy Registrations from one folder to another.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to copy Registration to. |
| strSource | String | IN | Folder to copy Registrations from. |
| strRequestor | String | IN | The user requesting the operation. |

##### Pseudocode

*None Shown*

**Construction Hint**

Do not copy Product Specific Registrations. Should not require any changes, as Product Specific Registrations will be against the Product Item Id, and not the Job Id.

#### createConfidentialGroup

This calls ConfidentialLib.java to create the Registration group for a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | Folder to create the Registration group to. |

##### Pseudocode

*None Shown*

#### deleteRegistrationsForFolder

This calls RemoveRegistration.java to delete all Registration to a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to delete all Registrations to. |
| strRequestor | String | IN | The user requesting the operation. |

##### Pseudocode

*None Shown*

#### extendRegistrations

This calls AddRegistration.java to extend the expiry date of a Registration.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjRegistration | IDfPersistenObject | IN | Registration to extend. |
| strRequestor | String | IN | The user requesting the operation. |
| intDuration | Integer | IN | The duration to extend the Registration for. |

##### Pseudocode

*None Shown*

#### markConfidential

This calls ConfidentialLib.java to mark a folder as Confidential.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | The folder to mark Confidential. |
| strRequestor | String | IN | The user requesting the operation. |

##### Pseudocode

*None Shown*

#### moveRegistrations

This calls AddRegistration.java to move Registrations after a Job has been moved.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder that was moved. |
| strSource | String | IN | The Parent folder that the Job was move from. |
| strRequestor | String | IN | The user requesting the operation. |

##### Pseudocode

*None Shown*

**Construction Hint**

Do not move to/from Product Specific Access as whole group will change on the ACL. Should not require any changes, as Product Specific Registrations will be against the Product Item Id, and not the Job Id.

#### reassignClientRegistrations

This calls AddRegistration.java to move Registrations after a Pending Client has been reassigned.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idSource | IDfId | IN | The object id of the Pending Client that was reassigned. |
| idTarget | IDfId | IN | The object id of the Client it was reassigned to. |
| strRequestor | String | IN | The user requesting the operation. |

##### Pseudocode

*None Shown*

#### removeApprover

This calls RemoveRegistration.java to remove an Auto Approver from a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to remove the user as an Auto Approver. |
| strUserName | String | IN | User being removed as an Auto Approver. |
| strRequestor | String | IN | The user requesting the operation. |

##### Pseudocode

*None Shown*

#### removeDelegatedApprover

This calls RemoveRegistration.java to remove a Delegated Approver from a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to remove the user as a Delegated Approver. |
| strUserName | String | IN | User being removed as a Delegated Approver. |
| strRequestor | String | IN | The user requesting the operation. |

##### Pseudocode

*None Shown*

#### removeRegistration

This calls RemoveRegistration.java to expire a Registration.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjRegistration | IDfPersistenObject | IN | The Registration to expire. |
| strRequestor | String | IN | The user requesting the operation. |

##### Pseudocode

*None Shown*

#### removeRegistration

This calls RemoveRegistration.java to expire a Registration to a folder for a user.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to expire the Registration to. |
| strUserName | String | IN | User whose Registration is being expired. |
| strRequestor | String | IN | The user requesting the operation. |

##### Pseudocode

*None Shown*

#### removeRegistrationExpired

This calls RemoveRegistration.java to expire a Registration due to the expiry date being reached.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Registration object to expire. |
| strRequestor | String | IN | The user requesting the operation. |

##### Pseudocode

*None Shown*

#### removeRegistrationLeft

This calls RemoveRegistration.java to expire a Registration due to the user leaving the firm.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Registration object to expire. |
| strRequestor | String | IN | The user requesting the operation. |

##### Pseudocode

*None Shown*

#### setAutoApprovers

This calls RegistrationApproverLib.java to set the Auto Approvers for a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to set the Auto Approvers to. |

##### Pseudocode

*None Shown*

#### unmarkConfidential

This calls ConfidentialLib.java to unmark a folder as Confidential.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to unmark as Confidential. |
| strRequestor | String | IN | The user requesting the operation. |

##### Pseudocode

*None Shown*

#### cleanupDeletedFolder

This calls cleanupDeletedFolder from RemoveRegistration.java to clean up all registrations of a deleted Job, Project or a deactivated Client\Sub Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | Deleted or Deactivated folder id to clean up all registrations |

##### Pseudocode

*Call cleanupDeletedFolder in Remove Registration*

RemoveRegistration.cleanupDeletedFolder(idFolder);

#### addClientDefaultGroupReg

This calls addClientDefaultGroupReg from AddRegistration.java to create a Registration for the user at the Client level.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The Client to create the Registration for. |
| strUserName | String | IN | The Guid of the user to create the Registration for. |

##### Pseudocode

*None Shown*

#### addProductRegistration

This calls addProductRegistration from AddRegistration.java to create a non expiring Registration for the user in the Product Registration group.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strProductItemId | String | IN | The Product Item Id to create the Registration for. |
| strUserName | String | IN | The Guid of the user to create the Registration for. |

##### Pseudocode

*None Shown*

#### removeProductRegistration

This calls removeProductRegistration from RemoveRegistration.java to remove the non expiring Registration for the user in the Product Registration group.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strProductItemId | String | IN | The Product Item Id to remove the Registration from. |
| strUserName | String | IN | The Guid of the user to remove the Registration for. |

##### Pseudocode

*None Shown*

### AddRegistration.java

This is a server side class which is responsible for adding Registrations.

#### addAutoApproverRegistration

This creates a system generated Registration that will make the user an Auto Approver for the given folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to create the Auto Approver Registration to. |
| strUserName | String | IN | The Guid of the user to make an Auto Approver. |
| strRequestor | String | IN | The user requesting the Registration be created. |

##### Pseudocode

*None Shown*

#### addDelegatedApproverRegistration

This function will create a Registration for a user that will make them a Delegated Approver to a folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to create the Delegated Approver Registration to. |
| strUserName | String | IN | The Guid of the user to make a Delegated Approver. |
| strRequestor | String | IN | The user requesting the Registration be created. |

##### Pseudocode

*None Shown*

#### addNominatedApproverRegistration

This will create a nominated delegated approver registration for the given user to the given folder.

This method will be called by the REG\_B04 – Manage Nominated Approvers component.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to create the Nominated Delegated Approver Registration to. |
| strUserName | String | IN | The Guid of the user to give the Nominated Delegated Approver registration to. |
| strApprover | String | IN | The Guid of the user who the Nominated Delegated Approver registration is being created for. |

##### Pseudocode

CALL the register method PASSING sobjTarget, strUserName, “SYSTEM-NOMINATED”, “ACTIVE”, strApprover, “DELEGATED-NOMINATED”, “nulldate”, “Nominated delegated approver access granted.”

#### addNonExpiryRegistration

This function will create a Registration for a user that does not expire and does not allow them to Approve Registrations for other users. This Registration is given to the Auto Approvers of non-Confidential folders below a Confidential folder so they wills till be able to view access the non-Confidential folder.

The requestor passed to the register method will be SYSTEM-NONEXPIRY, this is done to ensure that that if the user has an existing Approver Registration to the folder it won’t be over written.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to create a Non Expiring Registration to. |
| strUserName | String | IN | The Guid of the user to being given a Non Expiring Registrations. |

##### Pseudocode

*None Shown*

#### addRegistration

This function will create a normal expiring Registration for a user.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to create the Registration to. |
| strUserName | String | IN | The Guid of the user to create a Registration for. |
| strRequestor | String | IN | The user requesting the Registration be created. |
| intDuration | Integer | IN | The number of days the Registration will last for. |

##### Pseudocode

*None Shown*

#### addSystemRegistration

This function will create a system generated Registration for a user. This will over write any existing Registration they had to the folder.

It also checks whether the user should be given a Registration that will expire within the Default time period, or whether they should be given a non expiring Registration.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to create the Registration to. |
| strUserName | String | IN | The Guid of the user to create a Registration for. |
| strReason | String | IN | The message that will be logged when the Registration is created. |

##### Pseudocode

*None Shown*

#### addUnmarkRegistration

This function will create Registrations for users for the Default time period after a folder has been unmarked as Confidential. This will not over write the user’s Auto or Delegated Approver access to the folder that the Registration is being created for.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to create the Registration to. |
| strUserName | String | IN | The Guid of the user to create a Registration for. |
| strRequestor | String | IN | The user requesting the Registration be created. |
| strReason | String | IN | The message that will be logged when the Registration is created. |

##### Pseudocode

*None Shown*

#### copyRegistrations

This function will copy Registrations from one folder to another, copying the Registrations such that those on the target folder will match the source folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idTargetFolder | IDfId | IN | The id of the folder the Registrations will be copied to. |
| idSourceFolder | IDfId | IN | The id of the folder the Registrations will be copied from. |
| strRequestor | String | IN | The user requesting the Registrations be created. |

##### Pseudocode

*None Shown*

#### copyRegistrations

This function will copy Registrations from one folder to another. If the Boolean is set to true then any Auto or Delegated Approver Registrations copied from the source folder will be created as expiring Registrations. Copied Registrations will not remove a user’s existing Auto or Delegated Approver access to the target folder. If the Boolean is passed in as false then Delegated Approver Registrations will not be re-set and will be copied. Auto Approver registrations will be set to the default time period so that additional Auto Approvers will not be created for the folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idTargetFolder | IDfId | IN | The id of the folder the Registrations will be copied to. |
| idSourceFolder | IDfId | IN | The id of the folder the Registrations will be copied from. |
| strRequestor | String | IN | The user requesting the Registrations be created. |
| blnResetRegistrations | Boolean | IN | If this is true any Auto Approver or Delegated Approver Registrations on the source folder will be copied across as expiry Registrations. |

##### Pseudocode

*Get registrations from source folder.*

*Create registrations to the target folder.*

LOOP through the registrations

*If the registration is an Auto Approver registration OR we are resetting registrations and the registration is not expiring then give the user default access.*

IF the registrations is an Auto Approver registration OR (blnResetRegistrations = true AND the registration has no expiry date) THEN

*Create a registration to the target folder for the default time period*

ELSE

*Create a registration to the target folder that matches the current registration from the source folder*

END IF

END LOOP

**Construction Hint**

Do not copy Product Specific Registrations. Should not require any changes, as Product Specific Registrations will be against the Product Item Id, and not the Job Id.

#### extendRegistration

This function will extend an existing Registration, setting the new expiry date to today plus the new duration.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to extend the Registration to. |
| strUserName | String | IN | The Guid of the user’s Registration being extend. |
| strRequestor | String | IN | The user requesting the Registration be extended. |
| intDuration | Integer | IN | The number of days to extend the Registration for. |

##### Pseudocode

*None Shown*

#### moveRegistrations

This function moves Registrations after the Client, Job, or Project move has taken place, following the rules outlined in section .

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | The Job object that has been moved. |
| sobjParent | IDfPersistenObject | IN | The Parent object that Job has been moved to. |
| strRequestor | String | IN | The user who moved the Job. |

##### Pseudocode

IF sobjTarget is not a Job, Project or Client Then Throw error

idNewHeadClient, idOldHeadClient = Get the IDs of the new and old Head Client

If target object is tls\_Project

Call method moveRegistrationsForProject(targetObject, existingParent object, requestor)

End If

If target object is tls\_client

Call method moveRegistrationsForClient(target Head Client, moved Client, requester)

End If

IF Job is confidential Then

idNewConfidentialParent, idOldConfidentialParent = Get Restricted parent of objects

END IF

IF oldHeadClient != newHeadClient Then

CALL setAutoApproversForJobAndProjects passing idTargetJob,   
 idOldConfidentialParent, strRequestor

ELSE

*Jobs in same Head Client. Check if Job is now under a confidential sub-sclient, and if   
 so, and Job is not confidential, create registrations for all the Job and Project Auto   
 Approvers*

IF Job is not confidential and idNewConfidentialParent is confidential THEN

CALL registerUnconfidentialAutoApprovers passing sobjConfidentialParent

END IF

END IF

*Checking if registrations group must be copied*

IF Job is confidential Then

IF oldHeadClient == newHeadClient && oldConfidentialParent is not confidential &&   
targetParent is not confidential Then

Do nothing – Job has moved from Unrestricted sub-client to another   
 Unrestricted Subclient in the same Head Client.

ELSE

*The Confidential Job Group needs to be removed from the old Parent*

*Get the group name of the Confidential Job group*

strJobGroup = CALL getRegisteredGroup(idTargetJob)

*Get the group name of the Old Confidential Parent group*

strOldGroup = CALL getRegisteredGroupp(idOldConfidentialParent)

*Get the parent group*

IDfGroup grpOldParent = CALL getGroup(strOldGroup)

*Remove confidential group from the old parent*

CALL grpOldParent.removeGroup(strJobGroup)

*The Confidential group must be removed from the ReadOnly group of the parent*

SET idOldParentFolder = Retrieve ID of old parent folder

SET grpOldParentReadOnly = CALL getReadOnlyGroup(idOldParentFolder)

CALL grpOldParentReadOnly.removeGroup(strJobGroup)

*Update the new parent’s ReadOnly group with the confidential group*

SET idParentFolderId = Retrieve ID of new parent folder

CALL updateReadOnlyGroup passing idParentFolderId and strJobGroup

END IF

ELSE IF oldHeadClient != newHeadClient && idNewConfidentialParent is not confidential

*Copy the registrations from the old Head Client or Confidential Sub Client to the new folder.*

CALL moveCopyRegistrations

END IF

*Apply security*

CALL applySecurity from SvSecurityLib

**Construction Hint**

Do not move to/from Product Specific Access as whole group will change on the ACL. Should not require any changes, as Product Specific Registrations will be against the Product Item Id, and not the Job Id.

#### moveRegistrationsForClient

This function will move registrations when a Client is moved.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | The Client object that has been moved. |
| sobjParent | IDfPersistenObject | IN | The Parent object that Client has been moved from. |
| strRequestor | String | IN | The user who moved the Client. |

##### Development Hints

IF moved Client is NOT confidential Then

setAutoApproversForClientJobsAndProjects passing moved Client ID, old parent ID, true to set Auto Approvers, and requester GUID

END IF

IF moved Client has a Restricted group Then

Remove the Restricted group from the read-only group of the old parent

END IF

IF moved Client has a read-only group Then

Remove the read-only from the read-only group of the old parent

END IF

*Call method applySecurity passing moved Client and true to ensure that Restricted and read-only groups of contained Jobs and Projects are properly added to the parent groups.*

objSecurity.applySecurity(folDMSClient,true);

#### reassignClientRegistrations

This function will reassign Registrations after a Pending Client has been reassigned. Delegated approver of the Pending Client will be reassigned as Delegated Approver to the Existing Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idSource | IDfId | IN | The id of the old Client that was reassigned. |
| idTarget | IDfId | IN | The id of the new Client that the Pending Client was reassigned to. |
| strRequestor | String | IN | The user that carried out the reassignment. |

##### Pseudocode

*None Shown*

##### Construction Hint

1. Change the flag passed into copyRegistrations from true to false.

#### addDefaultReassignments

This function will add Registrations for a list of users to the target for the default duration, after a Pending Client/Job has been reassigned.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstUsers | IDfList | IN | List of users to create Registrations for. |
| idTarget | IDfId | IN | The id of the folder to create the Registration to. |
| strRequestor | String | IN | The user who reassigned the Pending Client/Job. |

##### Pseudocode

*None Shown*

#### addRegistration

This private function will work out the date when the Registration will expire and call the register method to create the actually Registration.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistenObject | IN | Folder to create the Registration to. |
| strUserName | String | IN | The Guid of the user to create a Registration for. |
| strRequestor | String | IN | The user requesting the Registration be created. |
| strApproverStatus | String | IN | The approver status that user will receive when creating the Registration. |
| intDuration | Integer | IN | The number of days the Registration will last for. |

##### Pseudocode

*None Shown*

#### extendRegistrationObject

This private function calculates the date that the Registration will expire, setting it to today’s date plus the duration. Once this has been done it will also update the existing Registration.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| objRegistration | IDfPersistenObject | IN | Registration object that will be extend. |
| strRequestor | String | IN | The user requesting the Registration be extended. |
| intDuration | Integer | IN | The number of days the Registration will last for. |

##### Pseudocode

*None Shown*

#### getConfidentialJobParentGroup

This private function will return the name of either the Confidential Sub Client group, or the Head Client group if the Job is not within a Confidential Sub Client above. This is used by moveRegistrations.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjJob | IDfSysObject | IN | The Job object to get the parent group from. |
| strConfidentialJobParentGroup | String | OUT | The name of Confidential Sub Client or Head Client group above the Job. |

##### Pseudocode

*None Shown*

#### isConfidentialJobParentGroup

This private function will return whether the first Client folder above a Confidential Job is Confidential.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjJob | IDfSysObject | IN | The Job object to check the if Client is Confidential. |
| isParentConfidential | Boolean | OUT | Whether or not Client above the Job is Confidential. |

##### Pseudocode

*None Shown*

#### moveConfidentialProjectGroupsForJob

This private function move of all of the Confidential Project groups from the old Parent folder of the Job, to the new Parent folder after a non-Confidential Job has been moved.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idJob | IDfId | IN | The id of the Job being moved. |
| idOldConfidentialParent | IDfId | IN | The id of the old Parent folder that the Confidential Project groups will be removed from. |
| idNewConfidentialParent | IDfId | IN | The id of the new Parent folder that the Confidential Project groups will be added to. |

##### Pseudocode

*None Shown*

#### moveCopyRegistrations

This private function will copy Registrations from one folder to another after a Job move has been carried out. Any Auto Approver or Delegated Approver Registration from the source folder will be copied as expiring Registrations to the target folder.

Note: This will not remove a user’s Auto Approver or Delegated Approver Registrations from the target folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idTargetFolder | IDfId | IN | The id of the folder to copy the Registrations from. |
| idSourceFolder | IDfId | IN | The id of the folder to copy Registrations to. |
| strRequestor | String | IN | The user who carried out the Job move. |

##### Pseudocode

*None Shown*

#### register

This private function will create a Registration for a user, adding an entry to tls\_registration table and adding the user to correct Registration group.

If a Registration for a user already exists then it will check if the existing Registration can be updated.

Registration will be updated if the user requesting the Registration is ‘SYSTEM’, or if a Non Expiring Registration is being set to a Delegated Approver, or it is a normal expiring Registration.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistentObject | IN | The folder object to create the Registration to. |
| strUserName | String | IN | The Guid of the user the Registration is being created for. |
| strRequestor | String | IN | The user who requested the Registration. |
| strRegistrationStatus | String | IN | The ‘t\_status’ value of the Registration. |
| strApprover | String | IN | The user who created the Registration. |
| strApproverStatus | String | IN | The ‘t\_approver\_status’ value of the Registrations. |
| dteEndRegDate | IDfTime | IN | The date that the Registration will be expired. If it can’t expired ‘nulldate’ will be passed in. |
| strReason | String | IN | The reason the Registration was being created. |

##### Pseudocode

*None Shown*

**Construction Hint**

Overload method to allow for both REG and PRD groups when adding a Registration to cater for Product Specific Access.

#### setAutoApproversForClientJobsAndProjects

This private function will set the Auto Approvers for all of the Jobs and Projects beneath a Client.

This is called by reassignClientRegistrations to ensure that all the Auto Approvers are correctly setup.

This method is also called after moving a Client to set all Auto Approvers.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idClient | IDfId | IN | The id of the Client folder to set the Auto Approvers for all the Jobs and Projects beneath. |
| blnSetAutoApprovers | Boolen | IN | Flag that indicates where Auto Approvers should be set for the Client. This flag should be set to TRUE for moved Clients. |
| idOldParent | IDfId | IN | ID of the old parent i.e. Head Client. Auto Approvers of the moved Client will be removed from the old Head Client if this argument is provided. |
| strRequestor | String | IN | The user who reassigned the Pending Client. |

##### Pseudocode

IF blnSetAutoApprovers THEN

Set Auto Approvers for the moved Client by calling method setAutoApprovers of the RegistrationLib.

IF ID of the old parent is provided THEN

Set Auto Approvers for the moved Client by calling method setAutoApprovers of the RegistrationLib.

Remove Auto Approvers of the moved Client from the old Head Client by calling the method RemoveApprover of the RegistrationLib

END IF

END IF

*Identify all Jobs for the passed-in Client idClient*

*Call method setAutoApproversForJobAndProjects for each Job of the Client.*

#### setAutoApproversForJobAndProjects

This private function will set the Auto Approvers for the passed in Job and all the Projects beneath it. It will also remove the Auto Approver from the old Parent folder.

If idOldParent folder is passed in as null, then it will simply set the Auto Approvers for the Job and its Projects and it will not remove them from the old Parent.

This function is used by reassignClientRegsitrations, reassignJobRegistrations and moveRegistrations.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idTargetJob | IDfId | IN | The id of the Job to set the Auto Approvers for and the Projects below it. |
| idOldParent | IDfId | IN | The id of the old Parent to remove the Auto Approvers being set from.  If this is null it will not only set the Auto Approvers. |
| strRequestor | String | IN | The user who reassigned/moved the Clients/Jobs. |

##### Pseudocode

*None Shown*

#### validateApproverStatus

This private function returns whether or not the status of the Registration is a valid Approver Status.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strApproverStatus | String | IN | The string to check if the value entered is a valid Approver Status. |
| validApproverStatus | Boolean | OUT | This will return whether or not the Approver Status entered is valid. |

##### Pseudocode

*None Shown*

#### addProductRegistration

This private function will create a non expiring Registration for the user in the Product Registration group.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strProductItemId | String | IN | The Product Item Id to create the Registration for. |
| strUserName | String | IN | The Guid of the user to create the Registration for. |

##### Pseudocode

*None Shown*

#### addClientDefaultGroupReg

This private function will create a Registration for the user to the Client for the configured period of time.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The Client object id. |
| strUserName | String | IN | The Guid of the user to create the Registration for. |

##### Pseudocode

GET registrationTime from config ‘/REG\_B01/DefaultGroupRegDuration’

CALL TlsRegistrations.addRegistration using idFolder, strUserName and registrationTime.

END LOOP

### RemoveRegistration.java

This is a server side class which is responsible for removing and expiring Registrations.

#### deleteRegistration

This function will check whether the registration can be deleted, if it is not a non expiring Registration then it will call the deleteRegistrationObject function.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistentObject | IN | The folder to delete the Registration for. |
| strUserName | String | IN | The name of the user to delete the Registration for. |
| strRequestor | String | IN | The user requesting the Registration be deleted. |
| strReason | String | IN | The reason the Registration will be deleted. |

##### Pseudocode

*None Shown*

#### deleteRegistraionsForFolder

This function will delete all Registration for a given folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| objFolder | IDfId | IN | The id of the folder to delete all Registration for. |
| strRequestor | String | IN | The user requesting all Registrations be deleted for the folder. |

##### Pseudocode

*None Shown*

#### deleteNominatedApproverRegistration

This function will delete a nominated delegated approver registration.

This method will be called by the REG\_B04 – Manage Nominated Approvers component.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idRegistration | IDfId | IN | The id of the nominated delegated approver registration to delete. |

##### Pseudocode

*Retrieve the registration object with an object id of idRegistration*

CALL deleteRegistrationObject PASSING the registration object retrieved, “SYSTEM-NOMINATED”, “Removing nominated approver access.”

#### deleteRegistrationObject

This function will delete a Registration object, and remove the user from the Registration group.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| objRegsitration | IDfPersistentObject | IN | The Registration to be deleted. |
| strRequestor | String | IN | The user requesting that the Registration be deleted. |
| strReason | String | IN | The reason the Registration is being deleted. |

##### Pseudocode

*None Shown*

#### deleteUnconfidentialAutoApprover

This function will delete an Unconfidential Auto Approver’s Registration. It will check if there user is an Auto Approver for any other Unconfidential folders below the one they are being deleted from, before removing them.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistentObject | IN | The folder to delete the Registration for. |
| strUserName | String | IN | The name of the user to delete the Registration for. |
| strRequestor | String | IN | The user requesting the Registration be deleted. |

##### Pseudocode

*None Shown*

#### removeAutoApprover

This function will remove an Auto Approver from the folder passed in if it is Confidential and the Head Client if necessary. Before removing the user as an Auto Approver it will carry out a series of checks to see if the user should be removed.

If the folder is Confidential it will check if the user is still an Auto Approver for that folder, if they aren’t they will lose their Auto Approver access to the Confidential folder.

If the folder is not Confidential it will check if there are any Confidential folders above it that the user needs to lose their Non Expiring Registration to. This will only be removed if the user is not an Auto Approver on Unconfidential folders below the Confidential one. (If they are being removed from a Confidential Sub Client, and they are an Auto Approver on an Unconfidential Project under a Confidential Job within the same Client they will still lose their Non Expiring Registration to the Sub Client, but they will still have it to the Confidential Job.)

It will also check if the Auto Approver needs to have their Auto Approver access removed from the Head Client as it is possible they may be an Auto Approver for more than one folder underneath the Head Client.

If after the checks have been carried out the Auto Approver is going to be removed, they will get a Registration to the folder for the Default time period, or a Non Expiring Registration if they are still an Auto Approver for an Unconfidential folder under the Confidential folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistentObject | IN | The folder to remove the user as an Auto Approver. |
| strUserName | String | IN | The name of the user to be removed as an Auto Approver. |
| strRequestor | String | IN | The user requesting the Auto Approver status be removed. |

##### Pseudocode

*None Shown*

#### removeDelegatedApprover

This function will remove a Delegated Approver from a folder, and give them a System Registration for either the Default period of time, or a Non Expiring Registration.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistentObject | IN | The folder to remove the user as a Delegated Approver. |
| strUserName | String | IN | The name of the user to be removed as a Delegated Approver. |
| strRequestor | String | IN | The user requesting the Delegated Approver status be removed. |

##### Pseudocode

*None Shown*

#### removeRegistration

This function will work out whether or not a Registration can be expired. Only Registrations an expiry date can be expired.

If the Registration is expired, it will be given a status of ‘EXPIRED-DENIED’.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistentObject | IN | The Registration object to be expired. |
| strRequestor | String | IN | The user requesting that the Registration be expired. |

##### Pseudocode

*None Shown*

#### removeRegistration

This function calls the unregister method to see if the Registration can be expired.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistentObject | IN | The folder object that the Registration will be expired for. |
| strUserName | String | IN | The user who’s Registration will be expired |
| strRequestor | String | IN | The user requesting that the Registration be expired. |

##### Pseudocode

*None Shown*

#### removeRegistrationExpired

This function expires a Registration due to the expiry date having lapsed.

The Registration will be given a status of ‘EXPIRED-LAPSED’.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistentObject | IN | The Registration object to be expired. |
| strRequestor | String | IN | The user expiring the Registration. |

##### Pseudocode

*None Shown*

#### removeRegistrationLeft

This function expires a Registrations due to user leaving the firm.

The expired Registration will be given a status of ‘EXPIRED-LEFT’.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistentObject | IN | The Registration object to be expired. |
| strRequestor | String | IN | The user expiring the Registration. |

##### Pseudocode

*None Shown*

#### expireRegistrationObject

This private function will expire a user’s Registration by updating the tls\_registration table and removing the user from Registration group.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistentObject | IN | The Registration object to be expired. |
| strRequestor | String | IN | The user expiring the Registration. |
| strReason | String | IN | The reason that the Registration is being expired. |

##### Pseudocode

*None Shown*

#### unregister

This private function will check whether a Registration is not a non expiring Registration before calling the expireRegistrationObject to expire it.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistentObject | IN | The folder object the Registration will be expired for. |
| strUserName | String | IN | The user who’s Registration is being expired. |
| strRequestor | String | IN | The user expiring the Registration. |
| strReason | String | IN | The reason that the Registration is being expired. |

##### Pseudocode

*None Shown*

#### cleanupDeletedFolder

This private function will check if the folder is a Job, Project, Client \ Sub Client. It will then check if the folder is confidential by calling isConfidential method from the SBO (TlsRegistration), if the folder is confidential it will call deleteRegistrationsForFolder from RemoveRegistration.java and delete all registrations for that folder. If the folder is not confidential it will also call deleteRegistrationsForFolder and check for folders above by calling a method on the SBO called getConfidentialParent. This function will return the Head Client of the folder passed in if no confidential folders are found. isHeadClient will then be called from the SBO to check if the result is a Head Client. removeAutoApprover method will be called after to remove all Auto Approver registrations. If the folder passed in is a Job or a Project this method will be called before the Job or Project is deleted. If it is a Client folder to be passed in then this method is called after the Client is deactivated.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | Deleted or Deactivated folder id to clean up all registrations |

##### Pseudocode

*Initialise and retrieve current session*

msessCurrent = SessionManager.getSession(mstrDocbaseName)

*Get the folder object from the folder id passed in*

pobjFolder as IDfPersistentObject = (IDfPersistentObject) msessCurrent.getObject(idFolder)

*Check if folder is a Job, Project or Client\Sub Client*

*Get the folder type*

strFolderType = pobjFolder.getString(“r\_object\_type”);

*Check if the folder is a Client*

IF (strFolderType.equals(“tls\_client”))

*Retrieve the Client Manager, Client Partner for the Client folder*

strManager = pobjFolder.getString(“t\_client\_manager”)

strPartner = pobjFolder.getString(“t\_client\_partner”)

End if

*Check if the folder is a Job*

IF (strFolderType.equals(“tls\_Job”))

*Retrieve the Job Partner and the Job Manager for the Job* strPartner = pobjFolder.getString(“t\_Job\_partner”) strManager = pobjFolder.getString(“t\_Job\_manager”);

*Clear the Manager and Manager for the Job object as the Auto Approver registrations will not be cleared at the Head Client if they are still set at the Job. Used dmadmin as it does not accept null string and dmadmin is not a user to be used for partner/manager.*

pobjFolder.setString(“t\_Job\_partner”, “dmadmin”);

pobjFolder.setString(“t\_Job\_manager”, “dmadmin”);

*Save the folder object*

pobjFolder.save();

End IF

*Check if the folder is a Project*

IF (strFolderType.equals(“tls\_Project”))

*Retrieve the Project Partner and Project Manager for the Project*

strPartner = pobjFolder.getString(“t\_Project\_partner”);

strManager = pobjFolder.getString(“t\_Project\_manager”);

*Clear the Manager and Manager for the Project object as the Auto Approver registrations will not be cleared at the Head Client if they are still set at the Project. Used dmadmin as it does not accept null string and dmadmin is not a user to be used for partner/manager.*

pobjFolder.setString(“t\_ Project\_partner”, “dmadmin”);

pobjFolder.setString(“t\_ Project\_manager”, “dmadmin”);

*Save the folder object*

pobjFolder.save();

End IF

*IF the folder is a Job or Project*

IF (strFolderType.equals(“tls\_Job”) || strFolderType.equals(“tls\_Project”))

*Get the parent group*

idParent = TlsRegistration.getConfidentialParent(idFolder)

*Get the parent type*

strParentType = pobjParent.getString(“r\_object\_type”)

IF (strParentType.equals(“tls\_Project”) || strParentType.equals(“tls\_Job”))

*Call deleteRegistrationsForFolder to delete all registrations for the folder*

deleteRegistrationsForFolder(idParent, strRequestor)

ELSE

*If it is not a Head Client*

IF (TlsRegistration.isHeadClient(idParent) = FALSE)

*Get the folder object from the parent id passed in*

pobjParent as IDfPersistentObject = (IDfPersistentObject) msessCurrent.getObject(idParent)

*Remove Auto Approvers for folder*

removeAutoApprover(pobjParent, strPartner, strRequestor)

removeAutoApprover(pobjParent, strManager, strRequestor)

END IF

END IF

*Get the Head Client*

idHeadClient = TlsRegistration.getHeadClient(idFolder)

*Get the folder object from the Head Client id passed in*

pobjHeadClient as IDfPersistentObject = (IDfPersistentObject) msessCurrent.getObject(idHeadClient)

*Remove Auto Approvers for folder*

removeAutoApprover(pobjParent, strPartner, strRequestor)

removeAutoApprover(pobjParent, strManager, strRequestor)

ELSE IF (strFolderType.equals(“tls\_client”))

*If it is not a Head Client*

IF (TlsRegistration.isHeadClient(idFolder) = FALSE)

*Check if it is confidential*

IF (TlsRegistration.isConfidential(idFolder) = TRUE)

*Call deleteRegistrationsForFolder to delete all registrations for the folder*

deleteRegistrationsForFolder(idFolder, strRequestor)

END IF

*Get Head Client*

idHeadClient= TlsRegistration.getHeadClient(idFolder)

*Get the folder object from the Head Client id passed in*

pobjHeadClient as IDfPersistentObject = (IDfPersistentObject) msessCurrent.getObject(idHeadClient)

*Remove Auto Approvers from Head Client*

removeAutoApprover(pobjHeadClient, strManager, strRequestor)

removeAutoApprover(pobjHeadClient, strClientPartner, strRequestor)

ELSE

*Call deleteRegistrationsForFolder to delete all registrations for the folder*

deleteRegistrationsForFolder(idFolders, strRequestor)

END IF

END IF

#### removeProductRegistration

This private function will remove a non expiring Registration for the user in the Product Registration group.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strProductItemId | String | IN | The Product Item Id to remove the Registration from. |
| strUserName | String | IN | The Guid of the user to remove the Registration for. |

##### Pseudocode

*None Shown*

### RegistrationApproverLib.java

This is a server side class which is responsible for creating Registrations for Auto Approvers.

#### getUnconfidentialAutoApprovers

This function will return a list of all users who are Auto Approvers for Unconfidential folders below the given Client or Job folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idTarget | IDfId | IN | The folder to get all Unconfidential Auto Approvers below. |
| lstUnconfidentialAutoApprovers | IDfList | OUT | A list of all the users who are Unconfidential Auto Approvers below the given folder. No user name will appear more than once in the list. |

##### Pseudocode

*None Shown*

**Construction Hints**

Abstract method to check Client Auto Approvers and use configuration to ensure correct Auto Approvers are used.

#### getUnconfidentialAutoApprovers

This function will return a list of all users who are Auto Approvers for Unconfidential folders below the given Client or Job folder.

If a user name is passed in then it will check if the user is Partner and/or Manager for any Unconfidential folders, returning the user’s name within the list if they are, or leaving the list blank if they aren’t.

If the user name is null then it will return all Unconfidential Auto Approvers.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idTarget | IDfId | IN | The folder to get all Unconfidential Auto Approvers below. |
| strUserName | String | IN | The user name to check if they are an Unconfidential Auto Approver below the folder. If null it will return all Auto Approvers below the folder. |
| lstUnconfidentialAutoApprovers | IDfList | OUT | A list of all the users who are Unconfidential Auto Approvers below the given folder. No user name will appear more than once in the list. |

##### Pseudocode

*None Shown*

#### getUnconfidentialJobsForSubClient

This function will return a list of all object ids for all Unconfidential Job folders below a Sub Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idSubClient | IDfId | IN | The Sub Client to get all Unconfidential Job folders for. |
| lstJobObjectId | IDfList | OUT | A list containing the object id of all Unconfidential Jobs below the Sub Client. |

##### Pseudocode

*None Shown*

#### isStillAutoApprover

This function will return whether or not the given user is an Auto Approver for the given folder.

This called when checking if an Auto Approver should be removed.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strUserName | String | IN | The user to check if they are still an Auto Approver. |
| sobjTarget | IDfSysObject | IN | The folder object to check if they are still an Auto Approver. |
| isStillAutoApprover | Boolean | OUT | Whether or not the user is an Auto Approver for the given folder. |

##### Pseudocode

*None Shown*

#### isStillSubFolderAutoApprover

This function will check if the user is an Auto Approver for any folders below the given folder.

This is called when checking if a user should be removed as an Auto Approver from a Head Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistentObject | IN | The folder object to check if they are an Auto Approver for any folders below. |
| strUserName | String | IN | The user to check if they are still an Auto Approver for any sub folders. |
| isStillAutoApprover | Boolean | OUT | Whether or not the user is an Auto Approver for any sub folders within the given folder. |

##### Pseudocode

lstFolders = CALL getUnconfidentialAutoApprovers passing sobjTarget and strUserName.

IF lstFolders is empty Return False

ELSE Return True

#### isStillUnconfidentialSubFolderAutoApprover

This function will check if a given user is an Auto Approver on any Unconfidential folders below the given folder.

This method will not check any Unconfidential folders below Confidential folders for the given folder. Eg: It will check the if the user is an Auto Approver for any Unconfidential Jobs and its Unconfidential Projects for a Sub Client is passed in. But it will not check if the user who is only an Auto Approver for an Unconfidential Project within a Confidential Job for the Sub Client.

This is called by deleteUnconfidentialAutoApprover.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistentObject | IN | The folder object to check if they are an Auto Approver for any Unconfidential folders below. |
| strUserName | String | IN | The user to check if they are still an Auto Approver for any Unconfidential sub folders. |
| isStillUnconfidentialAutoApprover | Boolean | OUT | Whether or not the user is an Auto Approver for any Unconfidential sub folders within the given folder. |

##### Pseudocode

*None Shown*

#### registerUnconfidentialAutoApprover

This function will create non expiring Registrations for all Auto Approvers of Unconfidential folders below a given folder.

This method will not create Registrations for any Auto Approvers of Unconfidential folders below Confidential folders for the given folder. E.g.: It will create Registrations for all the Auto Approver for any Unconfidential Jobs and its Unconfidential Projects for a Sub Client is passed in. But it will not create Registrations for users who are only Auto Approvers for an Unconfidential Project within a Confidential Job for a Sub Client. These people will have Registrations to the Confidential Job.

It will also return a list of users who were given non expiring Registrations to the folder.

This is called by markConfidential and moveRegistrations.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfSysObject | IN | The folder to create non expiring Registration for all Unconfidential Auto Approvers below it. |
| lstUserToRegister | IDfList | OUT | A list of all users who had non expiring Registrations created. |

##### Pseudocode

*None Shown*

#### setAutoApprovers

This function will setup the correct Registrations for Auto Approvers a folder so they will be able to access the folder, and to allow them to create Registrations to the Head Client and the folder if it is Confidential.

This works out what type of folder is the Auto Approvers are being set for and then calls the private method to set the Auto Approvers for the given folder type.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjT | IDfPersistentObject | IN | The folder object to set the Auto Approvers for. |

##### Pseudocode

*None Shown*

#### addAutoApprover

This private function will create an Auto Approver Registration for the user to the given folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfSysObject | IN | The folder object to create the Registration to. |
| strUserName | String | IN | The Guid of the user to create the Registration for. |

##### Pseudocode

*None Shown*

#### addNonExpiryRegistration

This private function will create a non expiring Registration for the user to the given folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfSysObject | IN | The folder object to create the Registration to. |
| strUserName | String | IN | The Guid of the user to create the Registration for. |

##### Pseudocode

*None Shown*

#### isSubclientAutoApprover

This private function will return whether or not the given user is an Auto Approver for the a Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strUserName | String | IN | The Guid of the user to check. |
| strClientId | String | IN | The object id of the Client folder. |
| blnIsClientApprover | Boolean | OUT | Whether or not the user is an Auto Approver for the Client. |

##### Pseudocode

RETURN CALL getUnconfidentialAutoApprovers from RegistrationApprovers passing strClientId and strUserName

**Construction Hints**

Abstract method to check Client Auto Approvers and use configuration to ensure correct Auto Approvers are used.

#### blnIsConfidentialApprover

This private function will return whether or not the user is an Auto Approver for the given Confidential folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjT | IDfPersistentObject | IN | The folder object for the Confidential folder. |
| strUserName | String | IN | The Guid of the user to check. |
| isConfidentialApprover | Boolean | OUT | Whether or not the user is an Auto Approver for the Confidential folder. |

##### Pseudocode

*None Shown*

#### blnIsHeadClientApprover

This private function will return whether or not the use is an Auto Approver for a Head Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistentObject | IN | The folder object for the Head Client. |
| strUserName | String | IN | The Guid of the user to check. |
| isHeadClientApprover | Boolean | OUT | Whether or not the user is an Auto Approver for the Head Client. |

##### Pseudocode

*None Shown*

#### getUnconfidentialJobAutoApprovers

This private function will return a list of the users who are Auto Approvers for Unconfidential Jobs beneath a Sub Client.

If a user name is passed in then it will check if the user is Partner and/or Manager for any Unconfidential Jobs under the Sub Client, returning the user’s name within the list if they are, or leaving the list blank if they aren’t.

If the user name is null then it will return all Unconfidential Job Auto Approvers.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idSubClient | IDfId | IN | The Sub Client to get all Unconfidential Auto Approvers for Jobs below. |
| strUserName | String | IN | The user name to check if they are an Unconfidential Auto Approver for a Job within the Sub Client. If null it will return all Auto Approvers for Unconfidential Jobs. |
| lstUnconfidentialAutoApprovers | IDfList | OUT | A list of all the users who are Unconfidential Auto Approvers for Jobs beneath the Sub Client. No user name will appear more than once in the list. |

##### Pseudocode

*None Shown*

#### getUnconfidentialProjectAutoApprovers

This private function will return a list of the users who are Auto Approvers for Unconfidential Projects beneath a Job.

If a user name is passed in then it will check if the user is Partner and/or Manager for any Unconfidential Projects under the Projects, returning the user’s name within the list if they are, or leaving the list blank if they aren’t.

If the user name is null then it will return all Unconfidential Project Auto Approvers.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strJobId | String | IN | The Job to get all Unconfidential Auto Approvers for Projects below. |
| lstUsersToRegister | IDfList | IN & OUT | This is an existing list of users that any Unconfidential Auto Approvers for the Project will be added to if they aren’t already within the list.  This list is returned once one all Unconfidential Auto Approvers have been added. |
| strUserName | String | IN | The user name to check if they are an Unconfidential Auto Approver for a Project within the Job. If null it will return all Auto Approvers for Unconfidential Projects. |

##### Pseudocode

*None Shown*

#### isAutoApprover

This private function will return whether or not a user is an Auto Approver for the given folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistentObject | IN | The folder object to check if the user is an Auto Apprver. |
| strUserName | String | IN | The Guid of the user to check. |
| blnIsAutoApprover | Boolean | OUT | Whether or not the user is an Auto Approver for the folder. |

##### Pseudocode

IF folder is confidential

RETURN isConfidentialApprover passing sobjTarget and strUserName

ELSE RETURN CALL getUnconfidentialAutoApprovers from RegistrationApprovers passing sobjTarget and strUserName

#### isInSecurityGroup

This private function will return whether or not the user is a member of the Security Administrator’s group.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strUserName | String | IN | The Guid of the user to check. |
| blnIsSecurityGroup | Boolean | OUT | Whether or not the user is a member of the Security Group. |

##### Pseudocode

*None Shown*

#### isJobApprover

This private function will return whether or not the user is an Auto Approver for any Jobs or Projects within a Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strUserName | String | IN | The Guid of the user to check. |
| strJobID | String | IN | The object of the Job folder to check within. |
| blnIsJobApprover | Boolean | OUT | Whether or not the user is an Auto Approver for a Job or Project within the given Client. |

##### Pseudocode

*None Shown*

#### isProjectApprover

This private function will return whether or not the user is an Auto Approver for any Projects within a Job.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strUserName | String | IN | The Guid of the user to check. |
| strJobID | String | IN | The object of id of the Job folder to check within. |
| blnIsProjectApprover | Boolean | OUT | Whether or not the user is an Auto Approver for a Project within the given Job. |

##### Pseudocode

*None Shown*

#### isUnconfidentialJobApprover

This private function will return whether or not a user is an Auto Approver for any Unconfidential Jobs within the given Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strUserName | String | IN | The Guid of the user to check. |
| strClientId | String | IN | The object of the Client folder to check within. |
| blnIsUnconfidentialJobApprover | Boolean | OUT | Whether or not the user is an Unconfidential Auto Approver for any Jobs within the given Client. |

##### Pseudocode

*None Shown*

#### isUnconfidentialProjectApprover

This private function will return whether or not a user is an Auto Approver for any Unconfidential Projects within the given Job.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strUserName | String | IN | The Guid of the user to check. |
| strJobId | String | IN | The object of the Job folder to check within. |
| blnIsUnconfidentialJobApprover | Boolean | OUT | Whether or not the user is an Unconfidential Auto Approver for any Projects within the given Job. |

##### Pseudocode

*None Shown*

#### deleteAutoApprovers

This private method will only ever be called for Restricted folders from the setAutoApprovers method.

The function will determine based on the configuration item ‘DefaultAccessEnabled', if the given Auto Approvers for the given folder and its subfolders need to be removed from the Head Client / Restricted parent folder of the folder and assign a Registration for the default time period. If the configuration item is enabled and if these Auto Approvers are not Auto Approvers for Unrestricted folders beneath the Restricted parent / Head Client, then their Auto Approver registration will be deleted from the Restricted parent / Head Client and will be assigned a Registration for the default time period. If the configuration item is not enabled, then their approver registration will be deleted completely from the restricted parent / Head Client as it is today.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| restrictedParent | IDfSysObject | IN | The first Restricted parent above sobjTarget, or the Head Client if none are Restricted. |
| strarrUsers | String[] | IN | An array of Auto Approvers to be removed from RestrictedParent if they are no longer Auto Approvers for Unrestricted folders beneath RestrictedParent |

##### Pseudocode

*Find all Auto Approvers for Unrestricted folders below the Restricted parent*

idsUnconfidentialAutoApproversBelowParent = CALL getUnconfidentialAutoApprovers from RegistrationApprovers passing RestrictedParent

Retrieve the value for the configuration item 'DefaultAccessEnabled'

FOR each Auto Approver in strarrUsers Do

IF idsUnconfidentialAutoApproversBelowParent does not contain Auto Approver Then

IF configuration item is set to FALSE THEN

*Remove Auto Approver from RestrictedParent if it is present*

ELSE

CALL removeApprover from RegistrationLib to assign a Registration for default time

period

END IF

END FOR

#### getUnconfidentialChildren

This private function will find all unconfidential children for a folder, and their unconfidential children.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget |  | IN | The folder for which unconfidential children will be found |

##### Pseudocode

Create a new list containing only sobjTarget

IF sobjTarget is a Head Client RETURN CALL getUnconfidentialChildrenForHeadClients

ELSE IF sobjTarget is a sub Client RETURN CALL getUnconfidentialChildrenForClients

ELSE IF sobjTarget is a Job RETURN CALL getUnconfidentialChildrenForJobs

ELSE IF sobjTarget is a Project RETURN empty list.

#### getUnconfidentialChildrenForHeadClients

This private function will find all unconfidential children for the Head Client, and its unconfidential children.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTargets | IDfList | IN | A list of folder IDs for which unconfidential children will be found. Note: these must all be of the same object type. |

##### Pseudocode

objectIds = New empty list

*Biuld query to get Jobs*

Query = “ SELECT r\_object\_id

FROM tls\_Job

WHERE t\_client\_id in '< sobjTargets >'

AND t\_confidential = 'false' “

Execute query, and add results to objectIds.

Add (CALL getUnconfidentialChildrenForJobs passing results from query) to objectIds

*Biuld query to get Sub-clients*

Query = “ SELECT r\_object\_id

FROM tls\_client

WHERE t\_head\_client\_code in '< sobjTargets >'

AND t\_confidential = 'false' “

Execute query, and add results to objectIds.

Add (CALL getUnconfidentialChildrenForClient passing results from query) to objectIds

RETURN objectIds.

**Construction Hints**

May need to utilise t\_head\_client\_code to ensure correct results are returned.

#### getUnconfidentialChildrenForClients

This private function will find all unconfidential children for a list of sub Clients and their unconfidential children.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTargets | IDfList | IN | A list of folder IDs for which unconfidential children will be found. Note: these must all be of the same object type. |

##### Pseudocode

objectIds = New empty list

*Biuld query to get Jobs*

Query = “ SELECT r\_object\_id

FROM tls\_Job

WHERE t\_client\_id in '< objTarget >'

AND t\_confidential = 'false' “

Execute query, and add results to objectIds.

Add (CALL getUnconfidentialChildrenForJobs passing results from query) to objectIds

RETURN objectIds.

#### getUnconfidentialChildrenForJobs

This private function will find all unconfidential children for a list of Jobs and their unconfidential children.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTargets | IDfList | IN | A list of folder IDs for which unconfidential children will be found. Note: these must all be of the same object type. |

##### Pseudocode

objectIds = New empty list

*Biuld query to get Jobs*

Query = “ SELECT r\_object\_id

FROM tls\_Project

WHERE t\_Job\_id in '< sobjTargets >'

AND t\_confidential = 'false' “

Execute query, and add results to objectIds.

RETURN objectIds.

#### setAutoApproversForClient

This private function will setup Registrations for the Auto Approvers of a sub Client folder.

If the Client is Confidential it will give Auto Approvers AUTO Registrations to the Client, and remove any no longer required Auto Approver registrations at the Head Client. If the Client is Unrestricted it will give the Auto Approvers AUTO Registrations to the Head Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfSysObject | IN | The folder object to set the Auto Approvers for. |

##### Pseudocode

SET headClient = CALL TlsRegistration method getHeadClient

IF sobjTarget is Restricted Then

CALL addAutoApprovers passing approver and sobjTarget for Client’s Auto Approvers

CALL deleteAutoApprovers passing headClient and Client’s Auto Approvers

ELSE

CALL addAutoApprovers passing approver and headClient for Client’s Auto Approvers

END IF

**Construction Hints**

Abstract method to check Client Auto Approvers and use configuration to ensure correct Auto Approvers are used.

#### setAutoApproversForHeadClient

This private function will setup Registrations for the Auto Approvers of a Head Client folder.

It will give the Auto Approvers, AUTO Registrations to the Head Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfSysObject | IN | The folder object to set the Auto Approvers for. |

##### Pseudocode

*None Shown*

**Construction Hints**

Abstract method to check Client Auto Approvers and use configuration to ensure correct Auto Approvers are used.

#### setAutoApproversForJob

This private function will setup Registrations for the Auto Approvers of a Job folder.

If the Job is Restricted it will give Auto Approvers AUTO Registrations to the Job, and remove any no longer required Auto Approver registrations at the Head Client/restricted parent. If the Job is Unrestricted and is within a Restricted parent, it will add the Auto Approvers’ non-expiring registration to the first Restricted parent above it. If the Job is Unrestricted and is not within a Restricted parent, it will add an Auto Approver registration to the Head Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfSysObject | IN | The folder object to set the Auto Approvers for. |

##### Pseudocode

SET confidentialParent = CALL TlsRegistration method getConfidentialParent passing   
 sobjTarget’s parent

IF sobjTarget is Restricted Then

CALL addAutoApprovers passing approver and sobjTarget for Job’s Auto Approvers

CALL deleteAutoApprovers passing confidentialParent and Job’s Auto Approvers

ELSE

IF confidentialParent is Head Client Then

CALL addAutoApprovers passing confidentialParent and Job’s Auto Approvers

ELSE

CALL addNonExpiryRegistration passing confidentialParent and Job’s auto   
 approvers

END IF

#### setAutoApproversForProject

This private function will setup Registrations for the Auto Approvers of a Project folder.

It will give the Auto Approvers, AUTO Registrations to the Head Client, and if the Project is Confidential it will also give them AUTO Registrations to that. If the Project is Unconfidential and under a Confidential Sub Client/Job it will give them non expiring Registrations to the first Confidential Parent folder.

If the Project is Restricted it will give Auto Approvers AUTO Registrations to the Project, and remove any no longer required Auto Approver registrations at the Head Client/restricted parent. If the Project is Unrestricted and is within a Restricted parent, it will add the Auto Approvers’ non-expiring registration to the first Restricted parent above it. If the Project is Unrestricted and is not within a Restricted parent, it will add an Auto Approver registration to the Head Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfSysObject | IN | The folder object to set the Auto Approvers for. |

##### Pseudocode

SET confidentialParent = CALL TlsRegistration method getConfidentialParent passing   
 sobjTarget’s parent

IF sobjTarget is Restricted Then

CALL addAutoApprovers passing approver, sobjTarget for Project’s Auto Approvers

CALL deleteAutoApprovers passing confidentialParent and Project’s Auto Approvers

ELSE

IF confidentialParent is Head Client Then

CALL addAutoApprovers passing confidentialParent and Projects’s  
Auto Approvers

ELSE

CALL addNonExpiryRegistration passing confidentialParent and Project’s  
Auto Approvers

END IF

### ConfidentialLib.java

This is a server side class which is responsible for marking and unmarking folders as Confidential.

#### markConfidential

This function will mark a folder as Confidential, by setting the t\_confidential flag on the folder to true.

When marking a folder as Confidential it will create the Registration group for the Confidential folder, and create Registrations for the Auto Approvers of the Confidential folder.

Non expiring Registrations will also be created for all Auto Approvers of Unconfidential folders without a Confidential folder above them, to the newly made Confidential folder. Unconfidneital children below the newly marked confidential folder will have their Auto Approver registrations removed from their old Restricted parent if they are no longer Auto Approvers for Unrestricted folders under their old Restricted parent, and they will have Auto Approver registrations added to the new Restricted parent.

It will also add the newly created Confidential Registration group as a sub group of the Parent group. If there are any Confidential folders below the newly marked Confidential folder, their Registration groups will be added as sub groups of the newly created Confidential group.

If the configuration item “/REG\_B01/ LogFolderRestrictionChange” is set to TRUE after successfuly marking the folder Confidential the function will log the following information to the Registration Log table (t\_registration\_log):

* + - 1. The Object id of the folder Marked Confidential.
      2. The user who marked the folder Confidential.
      3. The date and time the folder was Marked Confidential.
      4. The action (in this case ‘Marked Restricted’).

For more information on Registration logging and associated function refer to section [3.8.8](#_RegistrationLog.java) of this document

Note: The non expiring Registrations for Unconfidential Auto Approver should always live at the first Confidential folder above the Unconfidential folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistentObject | IN | The folder object that will be marked Confidential. |
| strRequestor | String | IN | The user marking the folder Confidential. |

##### Pseudocode

Set the t\_confidential flag of sojbTarget to TRUE.

If sobjTarget type = tls\_job

Set the t\_product\_group\_access to FALSE

END IF

*Change the Read Only groups up to the top of the folder structure to include the new confidential group*

SET idParentFolderId = Get the parent folder of sobjTarget.

*Read-only group updates now performed by Security and ACL commands.*

~~CALL updateReadOnlyGroup passing idParentFolderId and strCurrentFolderGroup~~

*Create and set the group of Auto Approvers for the newly made Confidential folder*

SET strCurrentFolderGroup = CALL createConfidentialGroup passing sobjTarget ObjectId

*For all Unrestricted children under the newly made confidential folder, delete their registrations to their old Restricted parent if they are no longer Auto Approvers for Unrestricted folders under their old Restricted parent.*

SET oldConfidentialParent = CALL TlsRegistration method getConfidentialParent passing sobjTarget

SET unconfidentialAutoApproversOfOldParent = CALL getUnconfidentialAutoApprovers passing oldConfidentialParent

CALL deleteAutoApprovers passing oldConfidentialParent and unconfidentialAutoApproversOfOldParent

IF config item “/REG\_B01/ LogFolderRestrictionChange” Is TRUE

CALL RegistrationLog.logMark method to log the information to registration log table by passing the object id of the folder, username and ‘MARKED RESTRICTED’ as the reason.

ENDIF

*Find all Unrestricted children of the newly made confidential folder, and call setAutoApprovers in order to give the children’s Auto Approvers non-expiring registrations to the newly Restricted folder.*

idsUnconfidentialChildren = CALL getUnconfidentialChildren passing sobjTarget

For each unconfidential child Do

CALL setAutoApprovers on child

END FOR

CALL applySecurity from SvSecurityLib passing sobjTarget

#### updateReadOnlyGroup

~~This private method will firstly add the given group to the Read Only group of the given folder. If the folder is the Head Client, or the folder’s parent’s Read Only group already contains the folder’s Read Only group, the function will return. Otherwise, the function will call itself to update the folder’s parent with the current folder’s Read Only group. In this way, the tree is traversed, adding the ‘Read only’ group of each folder to the ‘Read only’ group of its parent folder until the Head Client folder’s ‘Read only’ group is updated, or a parent folder already contains the ‘Read only’ group of updated folder.~~

##### ~~Parameters~~

| ~~Name~~ | ~~Data Type~~ | ~~Direction~~ | ~~Description~~ |
| --- | --- | --- | --- |
| ~~strParentId~~ | ~~String~~ | ~~IN~~ | ~~The Id of the parent folder whose Read Only group must be updated.~~ |
| ~~strGroupName~~ | ~~String~~ | ~~IN~~ | ~~The name of the group that must be added to the Parent folder.~~ |

##### ~~Pseudocode~~

*~~Initialise the Boolean value to determine whether to continue up the tree~~*

~~SET blnContinueUpwards = TRUE~~

*~~Get the Read Only group for this folder~~*

~~SET grpReadOnlyGroup = CALL getGroup passing   
 TlsRegistration.READ\_ONLY\_GROUP\_PREFIX + idFolder.getId()~~

*~~If the Read Only group exists, no need to cascade upwards~~*

~~IF grpReadOnlyGroup == null Then~~

~~SET strReadOnlyGroup = CALL createReadOnlyGroup passing strParentId~~

~~SET grpReadOnlyGroup = CALL getGroup, passing strReadOnlyGroup~~

~~ELSE blnContinueUpwards = FALSE~~

~~END IF~~

*~~Add the given group to the Read Only groups~~*

~~grpReadOnlyGroup.addGroup passing strGroupName~~

~~SET parentsParent = Get the parent folder of the parent folder, passing strParentId~~

*~~Determine whether to continue updating Read Only upwards through the folder hierarchy~~*

~~IF blnContinueUpwards = TRUE Then~~

~~IF parentsParent is Head Client Then~~

~~blnContinueUpwards = FALSE~~

~~ELSE~~

*~~Get the Read only group of the parent~~*

~~SET strParentReadOnlyGroup = CALL getReadOnlyGroup passing parentsParent~~

*~~If the parent does not contain the current readOnlyGroup, do not continue~~*

~~IF strParentReadOnlyGroupArray != null~~

~~IF strParentReadOnlyGroupArray contains strReadOnlyGroup Then~~

~~blnContinueUpwards = FALSE~~

~~END IF~~

~~END IF~~

~~END IF~~

~~END IF~~

*~~Continue upwards if required~~*

~~IF blnContinueUpwards = TRUE Then~~

~~CALL updateReadOnlyGroup passing parentsParent and strReadOnlyGroup~~

~~ELSE~~

*~~Call apply security. This will then cascade all changes downwards.~~*

~~CALL applySecurity from SvSecurityLib passing strParentId~~

~~END IF~~

#### unmarkConfidential

This function will unmark a folder as Confidential, by setting the t\_confidential flag on a folder to false.

This method will call addRegistrationsToParent to move the Registrations from the unmarked folder to the Parent folder.

It will then delete all Registrations that existed for the unmarked folder.

It will then remove the unmarked folder Registration group from the Read Only group of the Parent folder, and add all the Confidential sub-groups that were (if there were any) with the unmarked Registration group to the Parent folder group.

Finally, it will call setAutoApprovers to add Auto Approver registrations for the folder and its unconfidential sub folders to the new Restricted parent, then will call applySecurity.

If the configuration item “/REG\_B01/LogFolderRestrictionChange”” is set to TRUE after sucessfuly unmarking the folder confidential the function will log the following information to the Registration Log table (t\_registration\_log):

* + - 1. The Object id of the folder Unmarked Confidential.
      2. The user who unmarked the folder Confidential.
      3. The date and time the folder was Marked Confidential.
      4. The action (in this case ‘Unmarked Restricted’).

For more information on Registration logging and associated function refer to section [3.8.8](#_RegistrationLog.java) of this document

Note: The Parent folder will either be the first Confidential folder above the unmarked folder, or the Head Client if there are no Confidential folders above it.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfPersistentObject | IN | The folder object that will be unmarked Confidential. |
| strRequestor | String | IN | The user unmarking the folder Confidential. |

##### Pseudocode

Set the t\_confidential flag of sojbTarget to FALSE

If sobjTarget type = tls\_job

IF isAllowedProductSpecificAccess THEN

Set the t\_product\_group\_access to TRUE

END IF

END IF

*Get the confidential group which was created*

SET grpConfidentialGroup = CALL getGroup passing   
 TlsRegistration.REGISTRATION\_GROUP\_PREFIX + sobjTarget ObjectId

*Create registration for users on the confidential group.*

CALL addRegistrationsToParent passing grpConfidentialGroup, sobjTarget, strRequestor

*Delete all registrations for the current folder*

SET mobjRemove = Create new instance of RemoveRegistration

CALL mobjRemove.deleteRegistrationsForFolder passing sobjTarget ID and strRequestor

*Get the parent group to remove the current registration group being unmarked.*

SET idParent = GET the parent of the sobjTarget

SET grpParentReadOnly =

CALL getGroup passing TlsRegistration.READ\_ONLY\_GROUP\_PREFIX + idParent.getId

*Remove the unmarked group from the parent.*

CALL grpParentReadOnly.removeGroup passing grpConfidentialGroup.getGroupName()

*Call setAutoApprovers on the folder to add Auto Approvers to first Restricted parent*

CALL setAutoApprovers on sobjTarget

*Find all Unrestricted children of the unmarked folder, and call setAutoApprovers in order to give the children’s Auto Approvers can be non-expiring registrations to the new Restricted parent or Auto Approver registration to the Head Client.*

idsUnconfidentialChildren = CALL getUnconfidentialChildren passing sobjTarget

For each unconfidential child Do

CALL setAutoApprovers on child

END FOR

CALL applySecurity from SvSecurityLib passing sobjTarget

IF config item “/REG\_B01/ LogFolderRestrictionChange” is TRUE

CALL RegistrationLog.logUnmark method to log the information to registration log table by passing the object id of the folder, username and ‘UNMARKED RESTRICTED’ as the reason.

ENDIF

#### createConfidentialGroup

This function will check if the Registration group for the folder being marked Confidential exists. If it doesn’t exist the group will be created.

Once the group is created the Auto Approvers will be set for the newly made Confidential folder and the name of the group returned.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The id of the folder to create the Confidential group for. |
| strGroupName | String | OUT | The name of folders Registration group. |

##### Pseudocode

*None Shown*

#### createReadOnlyGroup

This function will check if the Read Only group for the given folder exists. If it doesn’t exist the group will be created. The name of the Read Only group is then returned.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idFolder | IDfId | IN | The id of the folder to create the Confidential group for. |
| strGroupName | String | OUT | The name of folders Registration group. |

##### Pseudocode

*Set the new group name*

strNewGroup = TlsRegistration.READ\_ONLY\_GROUP\_PREFIX + idFolder.getId()

*Create a new group object*

grpRegistrationGroup = (IDfGroup) create new object of type “dm\_group"

*Set the group name*

grpRegistrationGroup.setGroupName(strNewGroup);

grpRegistrationGroup.save();

Return strNewGroup

**Construction Hint**

Products will share the read-only group of their parent Client.

#### addLists

This private function adds two lists together.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstA | IDfList | IN | The first list to add together. |
| lstB | IDfList | IN | The second list to add together. |
| concatList | IDfList | OUT | The two lists added together. |

##### Pseudocode

*None Shown*

#### addRegistrationsToConfidentialParent

This function will add the Registration from a folder that has just been unmarked to the Confidential Parent.

It will loop through all the Registrations for the folder being unmarked and check if it needs to create non expiring or Default Registrations to the Confidential Parent.

Note: This is called by addRegistrationsToParent.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| idSource | IDfId | IN | The id of the folder unmarked Confidential. |
| sobjParent | IDfSysObject | IN | The Confidential Parent folder object. |
| strRequestor | String | IN | The user who unmarked the folder Confidential. |

##### Pseudocode

*None Shown*

#### addRegistrationsToParent

This function will add the Registrations from the unmarked Confidential folder to the Parent folder.

It will check if the Registrations are being added to a Confidential Parent, and if they are call the addRegistrationsToConfidentialParent.

If they aren’t being added to the Confidential Parent, then it means they are being added to the Head Client and it will get all the users from the unmarked Registration group and give them Registrations for a Default period of time to the Head Client.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| grpConfidentialGroup | IDfGroup | IN | The Registration group for the folder that has been unmarked as Confidential. |
| sobjTarget | IDfPersistentObject | IN | The folder object, for the folder being unmarked as Confidential. |
| strReguester | String | IN | User unmarking the folder as Confidential. |

##### Pseudocode

*None Shown*

#### getConfidentialChildren

This private function will return a list of all the Confidential folders below the passed in folder. If there are no Confidential folders below it, it will return null.

This is called by the markConfidential and unmarkConfidential methods.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| sobjTarget | IDfSysObject | IN | The folder object to get all Confidential folders below. |
| lstConfidentialChildren | IDfList | OUT | A list containing the object ids of all the Confidential folders below the passed in folder. |

##### Pseudocode

*None Shown*

#### getConfidentialJobsForSubClient

This private function will return a list of Confidential Jobs beneath a Sub Client.

If no Confidential Jobs are found then null will be returned.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strClientId | String | IN | The object id of the Sub Client to get all Confidential Jobs for. |
| lstConfidentialObjects | IDfList | OUT | A list containing the object ids of all the Confidential Jobs below the given Client. |

##### Pseudocode

*None Shown*

#### getConfidentialProjectsForJob

This private function will return a list of all Confidential Projects beneath a Job.

If no Confidential Projects are found then null will be returned.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strJobId | String | IN | The object id of the Job to get all Confidential Projects for. |
| lstConfidentialObjects | IDfList | OUT | A list containing the object ids of all the Confidential Projects below the given Job. |

##### Pseudocode

*None Shown*

#### removeNonExpiringRegistrationsFromConfidentialParent

This private function will remove the non expiring Registrations from a Confidential Parent of a folder that has just been marked Confidential, where the users have been given non expiring or Auto Approver Registrations to the folder marked Confidential.

This method calls deleteUnconfidentialAutoApprover within RemoveRegsitration.java, which will carry out the validation to check if the user’s non expiring access should be removed from the Confidential Parent folder.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| lstUserToRemove | IDfList | IN | A list of users to check if their non expiring Registrations should be removed. |
| idParent | IDfId | IN | The object id of the Confidential Parent folder. |
| strRequestor | String | IN | The user marking the folder as Confidential. |

##### Pseudocode

*None Shown*

### RegistrationLog.java

This is a server side class which is responsible for updating the tls\_registration\_log table when logging Registration events.

#### logAdd

This function will call the log(String, String, String, String) method to add a record to the tls\_registration\_log recording the details about a newly created Registration.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strPerformer | String | IN | The Guid of the user performing the action. |
| strUserName | String | IN | The Guid of the user the Registration created for. |
| strFolderId | String | IN | The folder the Registration was created to. |

##### Pseudocode

*None Shown*

#### logAdd

This function will call the log(String, String, String, String, String) method to add a record to the tls\_registration\_log recording the details about a newly created Registration.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strPerformer | String | IN | The Guid of the user performing the action. |
| strUserName | String | IN | The Guid of the user the Registration is being created for. |
| strFolderId | String | IN | The folder the Registration was created to. |
| strReason | String | IN | The reason the Registration is being added. |

##### Pseudocode

*None Shown*

#### logDelete

This function will call the log(String, String, String, String, String) method to add a record to the tls\_registration\_log recording the details about a deleted Registration.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strPerformer | String | IN | The Guid of the user performing the action. |
| strUserName | String | IN | The Guid of the user who’s Registration is being deleted. |
| strFolderId | String | IN | The folder the Registration was deleted from. |
| strDescp | String | IN | The reason the Registration is being deleted. |

##### Pseudocode

*None Shown*

#### logExtend

This function will call the log(String, String, String, String) method to add a record to the tls\_registration\_log recording the details about a Registration being extended.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strPerformer | String | IN | The Guid of the user performing the action. |
| strUserName | String | IN | The Guid of the user who’s Registration is being extended. |
| strFolderId | String | IN | The folder the Registration is being extend to. |

##### Pseudocode

*None Shown*

#### logRemove

This function will call the log(String, String, String, String) method to add a record to the tls\_registration\_log recording the details about a Registration being expired.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strPerformer | String | IN | The Guid of the user performing the action. |
| strUserName | String | IN | The Guid of the user who’s Registration is being expired. |
| strFolderId | String | IN | The folder the Registration is being expired from. |

##### Pseudocode

*None Shown*

#### logRemove

This function will call the log(String, String, String, String, String) method to add a record to the tls\_registration\_log recording the details about a Registration being expired.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strPerformer | String | IN | The Guid of the user performing the action. |
| strUserName | String | IN | The Guid of the user who’s Registration is being expired. |
| strFolderId | String | IN | The folder the Registration was expired from. |
| strDescp | String | IN | The reason the Registration is being expired. |

##### Pseudocode

*None Shown*

#### logUpdate

This function will call the log(String, String, String, String, String) method to add a record to the tls\_registration\_log recording the details about a Registration being updated.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strPerformer | String | IN | The Guid of the user performing the action. |
| strUserName | String | IN | The Guid of the user who’s Registration is being updated. |
| strFolderId | String | IN | The folder the Registration is being updated to. |
| strDescription | String | IN | The reason the Registration is being update. |

##### Pseudocode

*None Shown*

#### logMark

This function will call the log(String, String, String, String, String) method to add a record to the tls\_registration\_log recording the details about a folder being marked Restricted.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strPerformer | String | IN | The Guid of the user performing the action. |
| strUserName | String | IN | This field will be left blank. |
| strFolderId | String | IN | The folder being marked restricted |
| strDescription | String | IN | The reason the folder is being marked Restricted. |

##### Pseudocode

*None Shown*

**Construction Hints:**

When updating the Registration Log table (t\_registration\_log) for marked/unmarked restricted function, the t\_user\_name value is left blank. The details of the user who performs the action will be added to the t\_performer field of the log table.

#### logUnmark

This function will call the log(String, String, String, String, String) method to add a record to the tls\_registration\_log recording the details about a folder being unmarked restricted.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strPerformer | String | IN | The Guid of the user performing the action. |
| strUserName | String | IN | This field will be left blank |
| strFolderId | String | IN | The folder being unmarked restricted. |
| strDescription | String | IN | The reason the folder is being unmarked Restricted. |

##### Pseudocode

*None Shown*

**Construction Hints:**

When updating the Registration Log table (t\_registration\_log) for marked/unmarked restricted function, the t\_user\_name value is left blank. The details of the user who performs the action will be added to the t\_performer field of the log table.

#### log

This private function will call the log(String, String, String, String, String) method to update update the tls\_registration\_log table without adding any details within the t\_action\_details column.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strPerformer | String | IN | The Guid of the user performing the action. |
| strEvent | String | IN | The event that occurred |
| strUserName | String | IN | The Guid of the user who had Registration created or modified. |
| strFolderId | String | IN | The folder the Registration was attached to. |

##### Pseudocode

*None Shown*

#### log

This private function will update the tls\_registration\_log table with the passed in arguments.

##### Parameters

| Name | Data Type | Direction | Description |
| --- | --- | --- | --- |
| strPerformer | String | IN | The Guid of the user performing the action. |
| strEvent | String | IN | The event that occurred |
| strUserName | String | IN | The Guid of the user who had Registration created or modified. |
| strFolderId | String | IN | The folder the Registration was attached to. |
| strEventDescription | String | IN | A description on why the Registration is being created or updated. |

##### Pseudocode

*None Shown*

## ModifyDefaultGroupCommand

#### validateArguments

This method is called by the execute method in Command class. It validates that a properties object has been passed in and contains required values.

##### Parameters

None.

##### Pseudocode

VALIDATE that this.arguments contains the following fields:

r\_object\_id

t\_user\_name

group\_name

VALIDATE that Object Type of r\_object\_id is one of (tls\_client, tls\_job)

VALIDATE can modify group using TlsRegistrations. canModifyDefaultGroup

**Construction Hint**

Validation of type and canModifyDefaultGroup to be performed in the preTransactionSteps() method.

#### protected executeAction

This command will update the default\_group attributes for Clients and Jobs.

##### Parameters

None.

##### Pseudocode

Get objectType from r\_object\_id

IF objectType = tls\_client THEN

EXECUTE Query 2 with the new group\_name and the r\_object\_id

ELSE IF objectType = tls\_job THEN

EXECUTE Query 2 with the new group\_name and the r\_object\_id

ELSE

Throw exception

END IF

## CreateClientDefaultGroupRegCommand

#### validateArguments

This method is called by the execute method in Command class. It validates that a properties object has been passed in and contains required values.

##### Parameters

None.

##### Pseudocode

VALIDATE that this.arguments contains the following fields:

r\_object\_id

object\_type

group\_name

VALIDATE that object\_ is one of (tls\_client)

**Construction Hint**

Validation of type to be performed in the preTransactionSteps() method.

#### protected executeAction

This command will create Registrations for all users in the defined group at the object level. This command will be used during Client creation process to create initial Registrations for group.

##### Parameters

None.

##### Pseudocode

Get all users in group “group\_name”

LOOP users

CALL TlsRegistrations.addClientDefaultGroupReg using r\_object\_id and user

END LOOP

## ModifyRegApproverAccessCommand

#### validateArguments

This method is called by the execute method in Command class. It validates that a properties object has been passed in and contains required values.

##### Parameters

None.

##### Pseudocode

VALIDATE that this.arguments contains the following fields:

r\_object\_id

reg\_approver\_access\_name

user\_guid

VALIDATE that Object Type of r\_object\_id is one of (tls\_client, tls\_job, tls\_project)

VALIDATE can modify Registration Approver Access using TlsRegistrations. canModifyRegApproverAccess

**Construction Hint**

Validation of type and canModifyRegApproverAccess to be performed in the preTransactionSteps() method.

#### protected executeAction

This command will update the Registration Approver Access setting as well as the GUID of the Registration Approver Acces last modified user on a Client, Job or Project.

##### Parameters

None.

##### Pseudocode

Get Object Type from the r\_object\_id

EXECUTE Query 3 with the object type, new approver setting and the user\_guid

## UpdateProductGroupRegCommand

#### validateArguments

This method is called by the execute method in Command class. It validates that a properties object has been passed in and contains required values.

##### Parameters

None.

##### Pseudocode

VALIDATE that this.arguments contains the following fields:

product\_item\_id

#### protected executeAction

This command will create Registrations for all users in the defined group to the Product Registration group for the given Product Item Id.

##### Parameters

None.

##### Pseudocode

group\_name = get name from configuration

‘/Registration/ProductSpecificAccess/ProductGroup’ for the product

Get all users in group ‘group\_name’

LOOP users

CALL TlsRegistrations.addProductRegistration product\_item\_id and user

END LOOP

Get all users in reg\_group

Check if there are any users listed within the reg\_group which are not in group\_name

IF there are THEN

LOOP through the list of users

CALL TlsRegistration.removeProductRegistration passing product\_item\_id and user

END LOOP

END IF

## SetUnsetProductSpecificAccessCommand

#### validateArguments

This method is called by the execute method in Command class. It validates that a properties object has been passed in and contains required values.

##### Parameters

None.

##### Pseudocode

VALIDATE that this.arguments contains the following fields:

List of Object Ids

Action (“set” or “unset”)

#### protected executeAction

This command will set or unset Product Specific Access for a list of Jobs.

##### Parameters

None

##### Pseudocode

IF action == “set” THEN

SET blnSetProductAccess = true

ELSE IF action == “unset” THEN

SET blnSetProductAccess =false

ELSE

Throw error as invalid action called.

END IF

FOR EACH jobObjectId IN Object Id List

strObjectType = getObjectType of objectId

IF strObjectType != “tls\_job”

Throw error as object is not a Job.

END IF

blnAllowedPSA = CALL isAllowedProductSpecificAccess(jobObjectId)

IF NOT blnAllowedPSA THEN

*Job cannot have Product Specific Access*

Throw error with Message Id 65003

ELSE

blnUserCanSet = CALL canSetUnsetProductSpecificAccess(jobObjectId, user)

IF NOT blnUserCanSet THEN

*User does not have permission to set/unset Product Specific Access*

Throw error with Message Id 65002

ELSE

IF blnSetProductAccess THEN

CALL setProductSpecificAccess passing jobObjectId

ELSE

CALL unsetProductSpecificAccess passing jobObjectId

END IF

END IF

END IF

END LOOP

**Construction Hint**

Possibly move loop to calling method instead of command to allow for correct messaging.

#### protected setProductSpecificAccess

This command will set Product Specific Access for a Job.

##### Parameters

jobObjectId

##### Pseudocode

*Get current Product Specific Access setting of Job*

blnCurrGoupAccess = EXECUTE Query 7 passing jobObjectId

IF blnCurrGroupAccess == true THEN

*Cannot set as is already set for the Job*

Throw error with Message Id 65004

ELSE

*Set the t\_product\_group\_access flag and call apply security*

Execute Query 5 passing jobObjectId and blnGroupAccess

Call applySecurity

END IF

#### protected unsetProductSpecificAccess

This command will unset Product Specific Access for a Job.

##### Parameters

jobObjectId

##### Pseudocode

*Get current Product Specific Access setting of Job*

blnCurrGoupAccess = EXECUTE Query 7 passing jobObjectId

IF blnCurrGroupAccess == false THEN

*Cannot unset as is already unset for the Job*

Throw error with Message Id 65005

ELSE

*Unset the t\_product\_group\_access flag and call apply security*

Execute Query 5 passing jobObjectId and blnGroupAccess

Call applySecurity

END IF

## UpdateProductSpecificAccessExclusionCommand

#### validateArguments

This method is called by the execute method in Command class. It validates that a properties object has been passed in and contains required values.

##### Parameters

None.

##### Pseudocode

VALIDATE that this.arguments contains the following fields:

add\_clients OR remove\_clients

requester\_guid

VALIDATE the requester\_guid is a user contained within one of the groups in ‘/REG\_W11/AuthorisedGroups’ if not throw error with Message Id ‘65050’

**Construction Hints**

1. The command will return an error if the user is not within the authorised groups or if there is an issue updating the registered table.
2. Either one of add\_clients and remove\_clients can be empty but at least one Client Object Id must be passed in to action.

#### protected executeAction

This command will update the tls\_product\_specific\_exclusion table within the DMS based on the Clients passed for adding and removing.

##### Parameters

None.

##### Pseudocode

current\_date = Get System Date

requester = requester\_guid

IF add\_clients IS NOT NULL AND NOT EMPTY THEN

Call addClientsForProductSpecificAccessExclusion passing add\_clients,

requester and current\_date

END IF

IF remove\_clients IS NOT NULL AND NOT EMPTY THEN

Call removeClientsFromProductSpecificAccessExclusion passing remove\_clients

END IF

#### protected addClientsForProductSpecificAccessExclusion

This method will execute the DQL queries to add entries to the tls\_product\_specific\_exclusion registered table.

##### Parameters

add\_clients - String containing Client Id’s separated by commas.

current\_date - Date containing the date that the update was processed.

requester - String contain the GUID of the user who requested the update.

##### Pseudocode

colResults = EXECUTE Query 10

FOR EACH client\_id in add\_clients AND NOT in colResults

IF EXECUTE Query 11 passing client\_id

EXECUTE Query 8 passing client\_id, current\_date, requester

END IF

END FOR

#### protected removeClientsFromProductSpecificAccessExclusion

This method will execute the DQL queries to remove entries from the tls\_product\_specific\_exclusion registered table.

##### Parameters

remove\_clients - String containing Client Id’s separated by commas.

##### Pseudocode

colResults = EXECUTE Query 10

FOR EACH client\_id in remove\_clients AND in colResults

EXECUTE Query 9 passing client\_id

END FOR