EMu Documentation

Dynamic Security

Document Version 1.3

EMu Version 4.1

www.kesoftware.com

Contents

SECTION 1

Dynamic Security

The default security model used by EMu is static in nature. User and Group privileges are defined in the EMu Registry and loaded into a module when it is invoked. Once invoked the security of the module remains the same throughout its lifetime. If a user can change the contents of a given field, then they can change it for all records (assuming Record Level Security allows the record to be modified). In some instances it would be useful to allow some security settings to be altered based on information stored in the current record.

For example:

1. Storage staff in a group called Storage are able to change all fields for records of type `Crate` and `Frame`, while for `Object` records they can only update the dimensions information. The current EMu security model does not provide a mechanism for implementing this requirement via Registry settings. It is possible to "hard wire" such functionality into the EMu client, however it becomes very difficult to change as new requirements arise.
   What would be useful is a mechanism that allows access to a column to be modified based on the contents of the record.
2. Similarly, you may require certain fields to be filled depending on the type of record. For `Object` records you may require the *Main Title* to be specified, while for `Crate` and `Frame` records the title should not be specified (in fact, it should not even be shown). The EMu Mandatory Registry entry allows a field to be defined as mandatory, however it is not possible to use this Registry entry to specify conditional mandatory settings. As with the first example, it would be useful to allow the mandatory setting for a field to be set based on the contents of the record.
3. Alternatively, you may want to alter Record Level Security settings based on the contents of data within the record when the record is saved. One such requirement may be that records whose record status is set to `Retired` may only be edited by users in group Admin. Such a feature can be "hard wired" into the database server. However a solution that uses the Registry would provide a more flexible mechanism.

With the addition of three Registry settings, EMu 4.1 implements a flexible and dynamic security model which can adapt based on the data stored within a record:

- The Column Access Modifier Registry entry handles the first example above, that is the ability to alter column access based on the contents of the record.
- The Mandatory Modifier Registry entry handles the second example above, that is the ability to adjust mandatory settings based on the contents of the record.
- The Security|Update Registry entry provides for the third example, that is the ability to change Record Level Security based on the contents of the current record.

In this document we look at all three Registry entries and explain how they may be used to provide a dynamic security model, i.e. one that changes based on the data in the current record.

SECTION 2

Security Update

The Security Update Registry entry allows the contents of one or more fields to be modified based on the value in a field whenever a record is saved (inserted or modified). Importantly, the Record Level Security (RLS) fields:

- `SecCanDisplay`
- `SecCanEdit`
- `SecCanDelete`

may be adjusted, allowing the record security to be modified. However, the Registry entry is not restricted to RLS fields, and any combination of fields may be adjusted.

The format of the entry is:

`User|`*user*`|Table|`*table*`|Security|Update|`*column*`|`*value*`|`*settings*

`User|`*user*`|Table|Default|Security|Update|`*column*`|`*value*`|`*settings*

`Group|`*group*`|Table|`*table*`|Security|Update|`*column*`|`*value*`|`*settings*

`Group|`*group*`|Table|Default|Security|Update|`*column*`|`*value*`|`*settings*

`Group|Default|Table|`*table*`|Security|Update|`*column*`|`*value*`|`*settings*

`Group|Default|Table|Default|Security|Update|`*column*`|`*value*`|`*settings*

where:

column defines which field should be consulted to look for a matching *value*.

value is an EMu based search pattern.

If there is a match of the data in *column* with the *value* query, then the Registry entry is used and *settings* is applied.

If *column* contains a table of values, each entry is checked against *value*.

Since *value* is a pattern, particular attention must be paid if you want to match the complete contents of a field.

For example, to match `Collection` but not `Non-collection`, the pattern `^Collection$` should be used. It is important to remember that *value* operates in the same way as an EMu search term.

All comparisons of *value* are case insensitive.

settings is a semi-colon separated list of assignments to columns that is applied if there is a match of the data in *column* with the *value* query.

The format of *settings* is:

column=[+/-]*term*:[+/-]*term*:...;*column*=[+/-]*term*:...

where:

column is the name of the column to be modified.

term is the value to be set in *column.*

- If *term* is preceded by a plus symbol (+), the value is added to the existing list of values.
- If *term* is preceded by a minus sign (-), the value is removed from the existing list of values.
- If no symbol precedes a *term*, the current contents of *column* are removed and replaced with *term*.
- If more than one *term* is supplied for a given column, separated by colons (:), each *term* is applied one after the other.
- If more than one column is to be modified, each set of column settings is separated by a semi-colon (;).

Examples

If we want Record Level Security to be adjusted so that users in group Admin are the only ones allowed to edit and delete records that have a record status of *Retired*, the following entry can be used:

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`Default`
Key 5	`Security`
Key 6	`Update`
Key 7	`SecRecordStatus`
Key 8	`^Retired$`
Value	`SecCanEdit=Group Admin;SecCanDelete=Group Admin`

Keys 7 and 8 indicate that the entry only applies where the *SecRecordStatus* field matches the pattern `^Retired$` (i.e. where the field contains `Retired` only). If this is not the case, then the Registry entry is ignored. Where a record does match, the *SecCanEdit* field is set to `Group Admin`. As a leading plus or minus is not supplied, the contents of *SecCanEdit* are replaced with `Group Admin`. A similar update occurs for *SecCanDelete*.

A more complex example would involve removing edit access for all users in groups Conservation and Storage when an object is deaccessioned. Let's assume that the field *RecObjectStatus* contains the word `Deaccessioned` for objects that are no longer part of our collection. A suitable Registry entry would be:

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ecatalogue`
Key 5	`Security`
Key 6	`Update`
Key 7	`RecObjectStatus`
Key 8	`^Deaccessioned$`
Value	`SecCanEdit=-Group Conservation:-Group Storage`

Notice how the terms to set have a leading minus sign, indicating the term (in this case the group name) is to be removed from the field *SecCanEdit*.

A third example requires all records with an object valuation of `High` to have group Student removed and group Valuers added for both displaying and editing the record. The field containing the object valuation is *ValValuationCode*. A suitable Registry entry is:

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ecatalogue`
Key 5	`Security`
Key 6	`Update`
Key 7	`ValValuationCode`
Key 8	`^High$`
Value	`SecCanDisplay=-Group Student:+Group Valuers; SecCanEdit=-Group Student:+Group Valuers`

Notice how more than one field may be updated with a single Registry entry. Note also:

- If a term has a leading plus symbol and the term already appears in the field, it is not added again.
- Similarly, if a term has a preceding minus and it does not appear in the field, it is ignored.

In this final example we restrict the display privilege for records that have not been approved for viewing on the intranet to groups Admin, Curator, Storage and Conservation. Once the record has been approved for viewing on the intranet we will allow all users to view the record. In this case two Registry entries are required:

- The first restricts access for records not available on the intranet.
- The second allows access to all users for record available on the intranet.

Suitable Registry entries are:

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ecatalogue`
Key 5	`Security`
Key 6	`Update`
Key 7	`AdmPublishWebPasswordFlag`
Key 8	`N`
Value	`SecCanDisplay=Group Admin:+Group Curator:+Group Storage:+Group Conservation`

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ecatalogue`
Key 5	`Security`
Key 6	`Update`
Key 7	`AdmPublishWebPasswordFlag`
Key 8	`Y`
Value	`SecCanDisplay=Group Default`

Notice how the first term in the first Registry entry (`Group Admin`) does not have a leading plus or minus, meaning it replaces the current contents of *SecCanDisplay*. The following terms require a leading plus symbol otherwise they will also clear the current contents rather than adding to the first term. The second Registry entry replaces the contents of *SecCanDisplay* with group Default (this allows access for everyone).

By using a combination of Registry entries it is possible to produce quite sophisticated and dynamic privilege changes.

What's happening behind the scenes

The Security Update Registry entry is a Record Level Security based Registry entry and like all RLS Registry entries it is enforced by the database engine. The `security` file stored in the table directory provides the configuration used for RLS. The XML format of the `security` file has been extended to allow the contents of the Security Update Registry entry to be accommodated. Consider this entry:

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ecatalogue`
Key 5	`Security`
Key 6	`Update`
Key 7	`SecRecordStatus`
Key 8	`^Retired$`
Value	`SecCanEdit=Group Admin:+Group Registration;SecCanDelete=Group Admin:+Group Registration`

The extra XML generated in the `security` file for this entry is:

```
<updates>
  <update name="SecRecordStatus" value="^Retired$">
    <columns>
      <column name="SecCanEdit">
        <values>
          <value operation="replace" term="Group Admin"/>
          <value operation="add" term="Group Registration"/>
        </values>
      </column>
      <column name="SecCanDelete">
        <values>
          <value operation="replace" term="Group Admin"/>
          <value operation="add" term="Group Registration"/>
        </values>
      </column>
    </columns>
  </update>
</updates>
```

As you can see the XML follows the sequencing of the Registry entry. If multiple Registry entries exist, the `<update>` tags are repeated.

The good news is that you do not need to add the XML to the `security` file. Whenever a security based Registry is added or modified in EMu, the `security` file is rebuilt automatically, and all you need to do is add the Registry entries.

The order of processing

The database server applies the Security Update settings whenever a record is saved (i.e. for all insertions and updates). The entries are applied after assignment expressions have been executed and **before** validation is run. This means that assignment expressions may be used to build a composite value that may be tested by Security Update settings. For example, it is possible to concatenate two fields into one that may then be checked.

It is also possible to apply sophisticated formula to calculate a field to be checked. For example, you may want to set Security Update Registry entries based on a range of valuations for an object. You could use an assignment expression to set a value in a field based on the ranges:

Range	Value
$0 - $1000	`Cheap`
$1001 - $10000	`Average`
$10001 -	`Pricey`

You may then use the three values, `Cheap`, `Average` and `Pricey` in Security Update Registry entries. Note that you cannot use validation code to compute values as the Security Update entries are applied before validation code is executed.

SECTION 3

Column Access Modifier

As the name suggests, the Column Access Modifier Registry entry is used to modify the default Column Access values based on data found in the current record.

The format of the Registry entry is:

`User|`*user*`|Table|`*table*`|Column Access Modifier|`*column*`|`*value*`|`*settings*

`User|`*user*`|Table|Default|Column Access Modifier|`*column*`|`*value*`|`*settings*

`Group|`*group*`|Table|`*table*`|Column Access Modifier|`*column*`|`*value*`|`*settings*

`Group|`*group*`|Table|Default|Column Access Modifier|`*column*`|`*value*`|`*settings*

`Group|Default|Table|`*table*`|Column Access Modifier|`*column*`|`*value*`|`*settings*

`Group|Default|Table|Default|Column Access Modifier|`*column*`|`*value*`|`*settings*

where:

column defines which field should be consulted to look for a matching *value*.

value is a case-insensitive match (i.e. patterns/wildcards are not allowed unlike with the Security Update entry). The *value* is checked against the complete contents of *column*. The comparison is case insensitive. If *column* is a table of values, then each entry in the table is tested. If one matches, the Registry entry applies. Values of `NULL` and `NOT NULL` may be used to represent empty and non-empty values respectively.

settings is a semi-colon separated list of assignments to columns that is applied if there is a match of the data in *column* with the *value* query.

The format of *settings* is:

column`=[+/-]`*perm*`:[+/-]`*perm*`:...;`*column*`=[+/-]`*perm*`:...`

where:

column is the name of the column to be modified.

perm is the permission to be adjusted.

- A leading plus sign is used to add a permission.
- A leading minus sign removes a permission.
- If no sign is supplied, the current permissions are replaced (this works the same as for the Security Update entry).

The list of possible values for *perm* is:

- `dvDisplay` - see column while viewing a record.
- `dvEdit` - see column while modifying a record.
- `dvInsert` - see column while inserting a record.

- `dvQuery` - see column while searching for a record.
- `duEdit` - change column while modifying a record.
- `duInsert` - change column while inserting a record.
- `duQuery` - change column while searching for a record.
- `duReplace` - use column in a global replace command.

Examples

For this example, we want to ensure the current location field cannot be updated for deaccessioned objects, i.e. we want to remove the `duEdit` permission where the object status is `deaccessioned`. A suitable entry is:

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ecatalogue`
Key 5	`Column Access Modifier`
Key 6	`RecObjectStatus`
Key 7	`Deaccessioned`
Value	`LocCurrentLocation=-duEdit`

Now when deaccessioned objects are displayed, the current location field will be greyed out (indicating that it cannot be modified). To restrict the above entry to users in group Curator only, you would need to change Key 2 to `Curator`.

The Column Access Modifier Registry entry is generally set on a group or user basis.

Unlike the Security Update Registry entry, where changes are applied when a record is saved, Column Access Modifier entries are applied immediately. If the content of a column is changed and it matches an entry, the entry is applied at once. Also, after applying any Column Access Modifier entries, any column affected by previous changes but not updated with the current changes will be reset to its default setting (as defined by the Column Access Registry setting).

In this next example, we want group Student to be able to change the *Notes* field for records that are not deaccessioned. In order to provide this setting the default Column Access settings must have `duEdit` enabled for group Student. The Registry entries required are:

Field	Value
Key 1	`Group`
Key 2	`Student`
Key 3	`Table`
Key 4	`ecatalogue`
Key 5	`Column Access`
Key 6	`NotNotes`
Value	`dvQuery;dvDisplay;dvEdit;dvInsert;duEdit;duInsert;duQuery;duReplace`

Field	Value
Key 1	`Group`
Key 2	`Student`
Key 3	`Table`
Key 4	`ecatalogue`
Key 5	`Column Access Modifier`
Key 6	`RecObjectStatus`
Key 7	`Deaccessioned`
Value	`NotNotes=-duEdit`

The first entry sets the default permissions for the column *NotNotes*. Notice that `duEdit` is enabled by default. This allows users in group Student to change the contents of the field.

The second entry modifies the default settings to turn off `duEdit` where the object is deaccessioned.

The above entries show how the Column Access Modifier Registry entry may be used with the Column Access Registry entry to provide a predicable set of permissions for all object status values.

In this last example we disable editing for all users in group Curator of the *Other Titles* field until a value is entered into the *Title* field. The restriction is to apply when creating records as well as modifying records. The following entry could be used:

Field	Value
Key 1	`Group`
Key 2	`Curator`
Key 3	`Table`
Key 4	`ecatalogue`
Key 5	`Column Access Modifier`
Key 6	`RecMainTitle`
Key 7	`NULL`
Value	`RecOtherTitles=-duEdit:-duInsert`

You may have been tempted to use the following entries:

Field	Value
Key 1	Group
Key 2	Curator
Key 3	Table
Key 4	ecatalogue
Key 5	Column Access Modifier
Key 6	RecMainTitle
Key 7	NULL
Value	RecOtherTitles=-duEdit:-duInsert

Field	Value
Key 1	Group
Key 2	Curator
Key 3	Table
Key 4	ecatalogue
Key 5	Column Access Modifier
Key 6	RecMainTitle
Key 7	NOT NULL
Value	RecOtherTitles=+duEdit:+duInsert

to ensure that users can edit the *Other Titles* field if the main title is filled. The second Registry entry forces the edit and insert privileges to be enabled even if the default Column Access settings do not allow it. It may be your intention to have this behavior, however the original brief only stipulated that edit and insert permissions should be disabled for other titles if the main title is empty. It is not expressed what the permissions should be if the main title is filled. Another way of writing the second set of Registry entries is:

Field	Value
Key 1	Group
Key 2	Curator
Key 3	Table
Key 4	ecatalogue
Key 5	Column Access
Key 6	RecOtherTitles
Value	dvQuery;dvDisplay;dvEdit;dvInsert;duEdit;duInsert;duQuery;duReplace

Field	Value
Key 1	`Group`
Key 2	`Curator`
Key 3	`Table`
Key 4	`ecatalogue`
Key 5	`Column Access Modifier`
Key 6	`RecMainTitle`
Key 7	`NULL`
Value	`RecOtherTitles=-duEdit:-duInsert`

In this case the default column permissions enable edit and insert privileges, while the second Registry entry disables edit and insert if the main title is empty. If the main title is filled, the default permissions are applied, hence enabling edit and insert privileges.

SECTION 4

Mandatory Modifier

The Mandatory Modifier Registry entry is used to modify the mandatory setting for a given field based on data found in the current record.

The format of the Registry entry is:

`User`|*user*|`Table`|*table*|`Mandatory Modifier`|*column*|*value*|*settings*

`User`|*user*|`Table`|`Default`|`Mandatory Modifier`|*column*|*value*|*settings*

`Group`|*group*|`Table`|*table*|`Mandatory Modifier`|*column*|*value*|*settings*

`Group`|*group*|`Table`|`Default`|`Mandatory Modifier`|*column*|*value*|*settings*

`Group`|`Default`|`Table`|*table*|`Mandatory Modifier`|*column*|*value*|*settings*

`Group`|`Default`|`Table`|`Default`|`Mandatory Modifier`|*column*|*value*|*settings*

where:

column defines which field should be consulted to look for a matching *value*.

value is a case-insensitive match (i.e. patterns/wildcards are not allowed unlike with the Security|Update entry). The *value* is checked against the complete contents of *column*. The comparison is case insensitive. If *column* is a table of values, then each entry in the table is tested. If one matches, the Registry entry applies. Values of `NULL` and `NOT NULL` may be used to represent empty and non-empty values respectively.

settings is a semi-colon separated list of assignments to columns that is applied if there is a match of the data in *column* with the *value* query.

The format of *settings* is:

column=*setting*;*column*= *setting*;. . .

where:

column is the name of the column to be modified.

setting is `true` (the column should be mandatory) or `false` (the column should not be mandatory, i.e. remove the mandatory setting).

Examples

Example 1

For this example, we want the *Main Title* field to be mandatory, but only if the record type is `Object`. A suitable entry is:

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ecatalogue`
Key 5	`Mandatory Modifier`
Key 6	`RecObjectType`
Key 7	`Object`
Value	`TitMainTitle=true`

Now when the record type is set to `Object` the *Main Title* field must be filled before the record can be saved successfully. If *Main Title* is not completed, the standard error message is displayed. You may tailor the error message shown using the Mandatory Registry entry. For example:

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ecatalogue`
Key 5	`Mandatory`
Key 6	`TitMainTitle`
Value	`False;Please enter a Main Title for the Object`

will display the error message *Please enter a Main Title for the Object* if the field is not filled while the mandatory setting is `true`. A `false` value indicates the field is not mandatory by default.

Example 2

The above example shows how to alter the mandatory setting for a field (*TitMainTitle*) based on the value in another field (*RecObjectType*). In this example the mandatory setting for more than one field is altered based on the values of multiple fields. If the record type is set to `Object` and the object status is set to `Accessioned`, then the *Accession Number, Date Accessioned* and *Accession Lot* fields must be supplied:

Field	Value
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ecatalogue`
Key 5	`Mandatory Modifier`
Key 6	`RecObjectType`
Key 7	`Object`
Value	`TitAccessionNo=true;TitAccessionDate=true;TitAccessionLot=true`

Field	Value
Field	`Value`
Key 1	`Group`
Key 2	`Default`
Key 3	`Table`
Key 4	`ecatalogue`
Key 5	`Mandatory Modifier`
Key 6	`TitObjectStatus`
Key 7	`Accessioned`
Value	`TitAccessionNo=true;TitAccessionDate=true;TitAccessionLot=true`

Where multiple values are to be checked, a Mandatory Modifier Registry entry is required for each value for each column. Where multiple Registry entries match based on the values within fields (as with the previous example where both the object type and object status matched) the mandatory settings are ANDed together. This means that unless all the settings for a given field are `true`, mandatory is set to `false`.

SECTION 5

Conclusion

EMu 4.1 sees the addition of three new Registry entries. The first, Security|Update, allows record level permissions to be altered when a record is saved, based on the contents of the record. The second, Column Access Modifier, allows field based privileges to be altered based on the contents of the record. The third, Mandatory Modifier, allows mandatory fields to be specified based on the contents of the record. The combination of the three facilities provides a useful mechanism for altering the EMu security settings dynamically. The use of dynamic security allows very flexible security models to be implemented.

Index