Kontrolle UserExits

Beim Speichern und bei Abschluss einer Aktion werden Skripte ausgeführt. Die Skripte selber sind verschlüsselt und können im Projekt nicht angepasst werden. Die Prüfungen in den verschlüsselten Bereichen können zum Teil über Parameter an- und ausgeschaltet werden.
In den Skripten werden UserExit-Funktionen ausgeführt und in diesen UserExits kann beliebiger Code ausgeführt werden. Bei Performance-Problemen sollte dieser Code auf lange Ausführungszeiten überprüft werden.

Speichern

Im Invoice-Modul wird beim Speichern das Skript **Invoice_DF_OnSave** ausgeführt und im Skript werden UserExit-Funktionen ausgeführt. Diese befinden sich im Skript **Invoice__UserExit_DocFileLib**. Es gibt eine Funktion **ue_OnSaveStart()**, die direkt zu Beginn ausgeführt wird. Wichtig ist zum Beispiel, dass pro Skriptausführung nur ein sync()-Befehl auf den Vorgang erfolgen sollte. In den UserExits sollte generell auf sync()-Befehle verzichtet werden! Die Rechnungsdaten werden später im Skript gespeichert. Werden im UserExit andere projektspezifische Funktionen aufgerufen, dann muss man sich auch diese Funktionen ansehen.

> In den UserExits sollten keine sync()-Befehle ausgeführt werden.

Es folgen diverse Prüfungen auf Kopf-Ebene, die über diverse Parameter gesteuert werden können. Im Anschluss werden die Rechnungspositionen in das Gentable-Objekt geladen und es werden weitere Prüfungen ausgeführt. Nach den Standard-Prüfungen wird extra eine UserExit-Funktion für das Gentable bereitgestellt. Manchmal wird in den Projekten bereits im ue_OnSaveStart() das Gentable ausgelesen und zurückgeschrieben, wodurch ein unnötiges sync() ausgeführt wird. Hier kann man den Code optimieren, indem man das korrekte UserExit verwendet. Über die UserExit Funktion **Gentable.prototype.ue_Invoice_OnSave** können projektspezifische Ausführungen am Gentable-Objekt vorgenommen werden. Das Objekt muss nicht in das Gentable-Feld zurückgeschrieben werden. Das passiert automatisch im Anschluss.
Nach dem Gentable-UserExit werden die Positionsdaten in die Datenbank-Tabelle **Invoice_Posting_Pos** geschrieben. Über den Parameter **GentableSuppressWritingDataIntoDb** kann zum Beispiel im Invoice-Modul das Schreiben in die Datenbank deaktiviert werden. Wichtig ist aber, dass die Daten zu bestimmten Zeitpunkten in die Datenbank geschrieben werden. Bei Rechnungen mit Bestellbezug werden in der Regel zwingend aktualisierte Daten in den Tabellen benötigt.

Beim Zurückschreiben der Rechnungspositionen in das Gentable-Feld wird ein sync()-Befehl auf den Vorgang ausgeführt und das sollte auch der einzige im Skript ausgeführte sync()-Befehl sein! Die Kopfdaten zu einer Rechnung werden in die Tabelle **Invoice_Posting_Head** geschrieben. Dies kann über den Parameter **HeadDataSuppressWritingDataIntoDb** deaktiviert werden.

Zum Abschluss wird nach allen Prüfungen und Anpassungen die UserExit-Funktion **ue_OnSave_End()** ausgeführt. Hier muss berücksichtigt werden, dass der sync()-Befehl im Skript bereits erfolgt ist.

Im Procurement-Modul und weiteren Modulen sind die UserExit-Funktionen analog aufgebaut.

Abschluss einer Aktion

Für den Abschluss einer Aktion kann über die Aktions-Konfiguration der ausgehende Kontrollfluss als Button angezeigt werden. In dem Fall erfolgt die Weiterleitung direkt. Der ausgehende Kontrollfluss kann auch ausgeblendet werden und die Weiterleitung kann über eine im Standard enthaltene benutzerdefinierte Aktion im Skript erfolgen (Hintergrund: nur so sind aus technischen Beschränkungen andere Beschriftungen auf dem Button möglich). An den benutzerdefinierten Aktion sind die Skripte **DEXPRO_Action_FinishAction**, **DEXPRO_Action_FinishAction_Distribute** oder **DEXPRO_Action_FinishAction_Distribute_AP** hinterlegt. Die Skripte selber haben keine UserExits und leiten nur den Vorgang weiter.

Es kann aber auch sein, dass eine projektspezifische benutzerdefinierte Aktion für die Weiterleitung verwendet wird. In dem Fall kann im Skript beliebiger Code enthalten sein und der sollte bei Performance-Problemen kontrolliert werden.

Eine benutzerdefinierte Aktion kann im Bearbeitungs-Modus angezeigt werden. Wenn die benutzerdefinierte Aktion ausgeführt wird, wird zuerst der Speicher-Vorgang gestartet. Um die Laufzeit der Weiterleitung getrennt vom Speichern zu prüfen, sollte zuerst gespeichert werden und dann darf erst die benutzerdefinierte Aktion ausgeführt werden.

Bei einer Weiterleitung über eine benutzerdefinierte Aktion wird die Weiterleitung über ein DocFile.forwardFile() ausgeführt. Die komplette Weiterleitung wird jetzt im Kontext des Skripts ausgeführt, welches bei der benutzerdefinierten Aktion hinterlegt ist. Die Ausführungszeit des Skripts umfasst die komplette Weiterleitung von dem Start der Weiterleitung bis der Vorgang einen neuen Workflow-Schritt erreicht hat! Zunächst wird das Prüfskript **DEXPRO_WF_CheckActionEnd** auf dem ausgehenden Kontrollfluss ausgeführt. Auch hier gibt es wieder UserExits. Da der Workflow für alle Module derselbe ist, befinden sich die UserExits im globalen Skript **DEXPRO__UserExit_WorkflowLib**. Analog zum Speichern gibt es das UserExit **ue_OnActionEnd_Start()**, welches zu Beginn des Skriptes vor den Standard-Prüfungen ausgeführt wird. Dann finden Prüfungen im Gentable statt und die Gentable-UserExit-Funktion **Gentable.prototype.ue_OnActionEnd** befindet sich im Skript **DEXPRO__UserExit_GentableAdd**. NAch den Prüfungen und nach dem sync()-Befehl gibt es noch die UserExit-Funktion **ue_OnActionEnd_End()**. Diese UserExits müssen auf Performance geprüft werden.

Wenn die Weiterleitung nicht durch eine Fehlermeldung gestoppt wird, wird die Folge-Aktion ermittelt; die Workflow-Regeln werden ausgewertet und ggf. werden Aktionen nach Auswertung der Workflow-Regeln übersprungen und dann wird die Folge-Aktion ermittelt und dann müssen die Workflow-Regeln auch wieder zur nächsten Aktion ausgewertet werden.

Wenn der neue Workflow-Schritt erreicht ist, wird der Vorgang den neuen zuständigen Benutzern ggf. im Posteingang hinzugefügt. Wenn einer der neu zuständigen Benutzer über neue Vorgänge im Posteingang direkt informiert wird, dann wird auch noch die Mail an den neuen Benutzer versendet und all dies passiert immer noch im Kontext des DocFile.forwardFile()! (Die Mailversendung kann über die Mandanteneigenschaft **EMailAsync** auf eine asynchrone Versendung umgestellt werden. Der Nachteil ist, dass Fehler bei einer manuellen Mailversendung nicht mehr direkt an den Anwender zurückgegeben werden.)

Am Ende wird dem Anwender ggf. direkt der nächste Vorgang angezeigt. Wenn der ausgehende Kontrollfluss als Button angezeigt wird, dann kümmert sich Documents um die Anzeige. Wenn der Button als benutzerdefinierte Aktion angezeigt wird, muss via Skripting der nächste Vorgang ermittelt werden und hier kann via UserExit **NavigationReturnObject.prototype.ue_Next** eingegriffen werden.

Das DocFile.forwardFile() umfasst demnach viele andere Ausführungen und Einzelschritte. Im Support kommt bei Performance-Problemen häufig die Meldung an, dass das Skript **DEXPRO_Action_FinishAction** eine lange Laufzeiten hat und dieses Skript optimiert werden soll. Das Skript selber führt aber nicht viel mehr als das DocFile.forwardFile() aus und demnach gibt es hier nicht viel zu optimieren! Das eigentliche Problem liegt irgendwo bei den Einzelschritten im DocFile.forwardFile()!

> Die Weiterleitung zum Beispiel über das Skript **DEXPRO_Action_FinishAction** umfasst alle Workflow-Skripte; die Zuordnung des Vorgangs in die Postangangs-Körbe der neuen User und umfasst selbst die Mailversendung über neue Vorgänge im Posteingang!

Eine einfache Variante, um bei der Weiterleitung diverse Einzelschritte auszuschließen kann ereicht werden, ist eine Verzögerung. Hierzu gibt es 2 Varianten.

Parameter Workflow_DelayAfterAction

Über den Parameter **Workflow_DelayAfterAction** kann direkt nach Abschluss der Aktion eine Verzögerung angesteuert werden. Bei Abschluss der Aktion wird lediglich das Prüf-Skript **DEXPRO_WF_CheckActionEnd** ausgeführt. Der Vorgang läuft im Anschluss direkt in eine Workflow-Verzögerung von einer Minute. Die interne Documents-Job-Engine löst die Verzögerung auf. Der Nachteil ist, dass man auf die Documents-Job-Engine angewiesen ist. Sollte die Documents-Job-Engine mit der Ausführung diverser Jobs ausgelastet sein, kann es ggf. lange dauern, bis die Verzögerung aufgelöst wird. In dem Fall sollte man sich wiederum die Asuführungszeitpunkte und Laufzeiten der Jobs anschauen und hätte hier eine mögliche Ursache für die Performance-Probleme. Ein weiterer Nachteil ist, dass die Workflow-Aktion am Vorgang noch den alten Wert hat und der Filter am zugehörigen Ordner ggf. erweitert werden muss, um die Vorgänge in Verzögerung auszublenden.
Durch die Verzögerung direkt nach der Aktion entfallen diverse potentiel performancelastige Aktionen. Lediglich die Anzeige des Folge-Vorgangs wird wieder ausgeführt. Sollte die Ausführungsdauer immer noch langsam sein, wird die Ursache sehr wahrscheinlich an den UserExits bei Abschluss der Aktion liegen.

Technische Workflow-Aktionen Delay, Delay1, Delay2, ...

Bei der Variante mit einer technischen Aktion werden einige zusätzliche Aktionen ausgeführt, die sich aber nicht großartig auf die Performance auswirken sollten. Die technischen Workflow-Aktionen **Delay**, **Delay1**, **Delay2**, ... **Delay5** können zwischen bestehenden Aktionen hinzugefügt werden. Es handelt sich nicht um eine Workflow-Verzögerung! Der Vorgang liegt in einer Workflow-Aktion und wird durch die Gruppe **TechAccessProfile** gesperrt. Dieser Gruppe sollten nur technische User zugeordnet werden und die Mailversendung über neue Posteingänge sollte bei diesen Usern deaktiviert sein! Nach Abschluss der Aktion und nach Abschluss der Prüfungen wird die Folge-Aktion ermittelt und die Workflow-Aktion am Vorgang wird auf "Delay" (bzw. "Delay1", ...) umgestellt. Dadurch müssen die Filter an den bestehenden Ordnern nicht erweitert werden. Workflow-Regeln müssen für technische Aktionen nicht berechnet werden.
Für jedes Modul gibt es entsprechende Skripte, um die Verzögerung aufzulösen. Im Invoice-Modul muss zum Beispiel das Job-Skript **Invoice_JOB_ForwardDelay** als Job eingeplant werden.

Revision #12
Created 2025-12-01 13:45:45 UTC by Markus Meisner
Updated 2026-01-20 13:43:45 UTC by Markus Meisner