



IBYKUS AP/®

Release Notes

VERSION 2.13

Application Platform for the Front Office

The logo features a stylized graphic of five curved, overlapping lines in shades of gray and white, resembling a wave or a series of steps, positioned to the left of the word "IBYKUS" in a bold, dark red serif font.

Herausgeber:

IBYKUS AG für Informationstechnologie

Herman-Hollerith-Straße 1

99099 Erfurt

Tel.: 0361 / 44 10 - 0

Fax: 0361 / 44 10 - 410

E-Mail: info@ibykus.de

Herausgabedatum:

September 2010.

Änderungen in diesem Dokument sind vorbehalten.

Gültigkeit:

Diese Release Notes sind gültig für IBYKUS AP/ 2.13.

Eingetragene Marken:

IBYKUS[®] und AP/[®] sind eingetragene Marken der IBYKUS AG für Informationstechnologie.

Excel[®], Microsoft[®], Outlook[®], PowerPoint[®], Visio[®] und Windows[®] sind eingetragene Marken der Microsoft Corporation.

Alle anderen Marken- und Produktnamen sind Marken oder eingetragene Marken der jeweiligen Markeninhaber.

Inhalt

1	Modellgetriebene Software-Entwicklung mit IBYKUS AP/.....	5
2	IBYKUS AP/ 2.13.....	6
3	Für Anwender	7
3.1	Dateien im Formular speichern und anzeigen	7
3.2	Hierarchische Kataloge	8
3.3	Datumseingabe per Kalender.....	9
3.4	Linien zur Gestaltung von Formular und Suche.....	9
3.5	Erweiterte Anzeige von Meldungen.....	10
3.6	Erweiterte Eingabeunterstützung konfigurierbar	10
3.7	Sonstiges	11
3.7.1	Suche.....	11
3.7.2	Liste	11
3.7.3	Formular.....	11
4	Für Software-Entwickler	12
4.1	Neue Sachverhalte.....	12
4.1.1	Verwaltung von Konfigurationsdaten	12
4.1.2	View-Untersachverhalte.....	12
4.2	LOBs in Formularen	13
4.2.1	Konfiguration.....	13
4.2.2	Zugriff auf LOB-Felder in Actions	15
4.3	Hierarchische Kataloge	16
4.3.1	View-Definition	16
4.3.2	Konfiguration.....	17
4.4	Anzeige von Sachverhaltsarten sperren.....	18
4.5	Zugriffsrechte (USVH ACS).....	19
4.6	Freie Kommandos und die Rolle LESER.....	20
4.7	Standard-Makros für RMI	20

4.8	Standard-Makros für Protokollierung	21
4.9	Standard-Makros für Debugging	22
5	Für Administratoren	25
6	Weitere Unterstützung	26

1 Modellgetriebene Software-Entwicklung mit IBYKUS AP/

IBYKUS AP/ ist ein Framework für die effiziente und ressourcenschonende Software-Entwicklung. Auf Basis dieser modellgetriebenen Entwicklungstechnologie realisiert IBYKUS sowohl kundenindividuelle Frameworks als auch komplette Anwendungen.

IBYKUS AP/ besteht aus mehreren Komponenten, hält alle Informationen zentral auf einer Datenbank und zeichnet sich durch eine offene Architektur sowie freie Konfigurierbarkeit aus. Der modulare Aufbau birgt den Vorteil, dass auf Änderungsanforderungen äußerst schnell reagiert werden kann.

Zielgruppe sind zum einen technisch versierte Programmierer, die mit Hilfe von IBYKUS AP/ organisationseigene Anforderungen an ihre Software-Lösungen umsetzen und ihre Anwendungen somit selbst weiterentwickeln können. Als Werkzeug für die effiziente Software-Entwicklung hat sich das Framework seit Jahren bewährt.

Zum anderen realisieren die Entwickler der IBYKUS AG auf Grundlage von IBYKUS AP/ maßgeschneiderte Komplettlösungen für Kunden aus Industrie und Verwaltung. Daher stehen ebenso Endanwender, die ihr spezifisches Fachgebiet Software-gestützt bearbeiten, im Fokus.

Insbesondere alle vorgangsorientierten Geschäftsprozesse lassen sich mit IBYKUS AP/ optimal abbilden. Auch besonders dynamische Themen, die häufigen inhaltlichen Änderungen unterliegen, können mit dem Framework realisiert werden. Die mit Hilfe von AP abbildbaren Fachgebiete sind sehr vielschichtig:

- Vorgangsbearbeitung
- Finanzbereich (Budget-, Fördermittelverwaltung)
- Personalthemen (Personal-, Auszubildendenverwaltung)
- Ideenmanagement
- Customer Service Management

Mit IBYKUS AP/ beschreitet IBYKUS einen Weg in der Anwendungserstellung, der die Vorteile von Standard- und Individualsoftware in sich vereinigt.

Anwendungen auf Basis von IBYKUS AP/ betten sich perfekt in bestehende IT-Landschaften ein und ermöglichen durch ihre Schnittstellen die nahtlose Interaktion mit bereits vorhandenen Lösungen.

2 IBYKUS AP/ 2.13

Seit April 2010 ist die Version 2.13 von IBYKUS AP/ verfügbar.

Im Zuge der fortwährenden Marktbeobachtung und Forschungsarbeit fließen in IBYKUS AP/ kontinuierlich neue und markterprobte Technologien ein. Für die neue Version 2.13 wurde beispielsweise die Unterstützung von Microsoft Windows 7 und Oracle Server 11g weiter ausgebaut.

Dadurch steht ein funktional stark erweitertes Framework für die effektive Software-Entwicklung bereit. Auch für den Endanwender werden einige der Neuerungen bei der Bedienung der Software durch Nutzung des AP/ Client spürbar.

In den vorliegenden Release Notes möchten wir Endanwendern, Applikationsentwicklern und Administratoren gleichermaßen einen Überblick darüber geben, was sich im Vergleich zur Vorgänger-Version verändert hat.

3 Für Anwender

3.1 Dateien im Formular speichern und anzeigen

Dateien ließen sich mit AP/Client bis jetzt im Dokumentenmanagement von IBYKUS AP/ verwalten. Seit der Version 2.13 bietet IBYKUS AP/ die Möglichkeit, Dateien verschiedener Formate mit Schaltflächen oder unmittelbar in Formularfeldern zu speichern.

Die Anzeige und Bearbeitung der Dateien erfolgt abhängig vom Dateityp und von der Konfiguration der Anwendung mit der internen Dateianzeige oder einem externen Programm.



The screenshot displays a web form with the following fields and content:

- Name:** Mustermann
- Vorname:** Max
- Geburtsdatum:** 01.01.1971
- Familienstand:** verheiratet
- Wohnort:** Neustadt
- PLZ:** 99999
- Straße, Nr.:** Wohnstraße 13
- Foto:** A black and white portrait of a man.
- Vertrag (PDF):** A preview of a PDF document titled "Arbeitsvertrag". The text in the preview includes:
 - Zwischen
 - Firma:** ABC GmbH
 - Straße:** Industriestraße 11
- Daten:** A button with an Excel icon and the text "Daten".

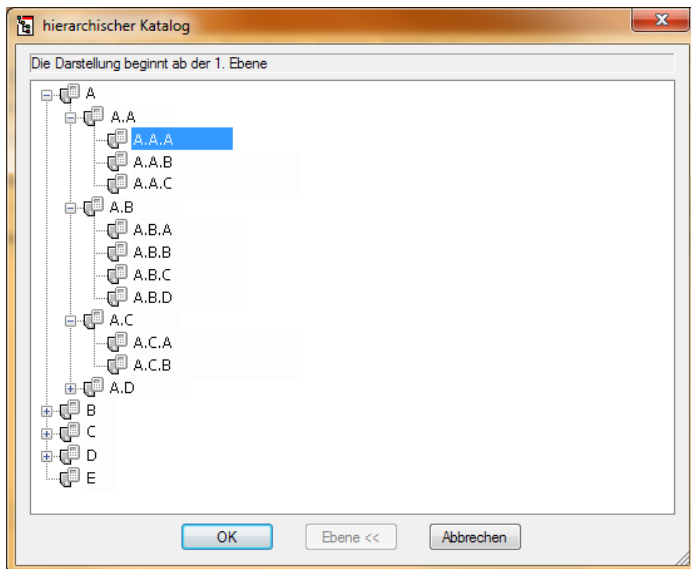
Unterstützt werden u.a. die folgenden Dateitypen:

- Bilddateien (BMP, JPG und PNG)
- Microsoft-Office-Dokumente (Word, Excel, PowerPoint und Visio)
- PDF
- RTF
- TXT
- XML
- HTML

3.2 Hierarchische Kataloge

Es wurde eine neue Katalogart eingeführt: Der hierarchische Katalog.

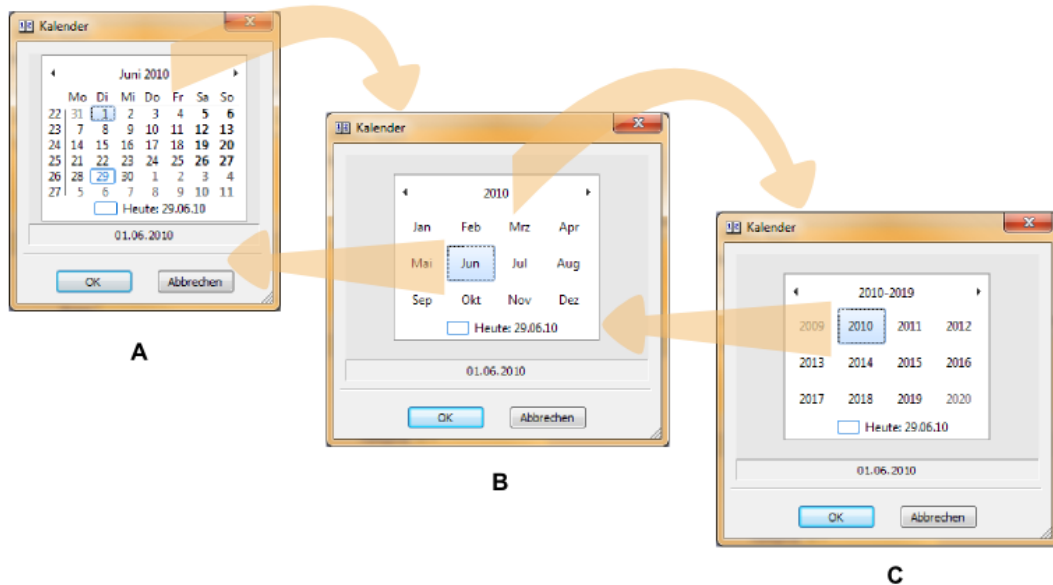
Dort werden die Werte hierarchisch geordnet dargestellt.



Wird der hierarchische Katalog aus einem leeren Feld geöffnet, sind alle Ebenen der Hierarchie sichtbar. Anderenfalls beginnt die Darstellung der Werte mit der Ebene, in der sich der im Feld eingetragene Wert befindet.

3.3 Datumseingabe per Kalender

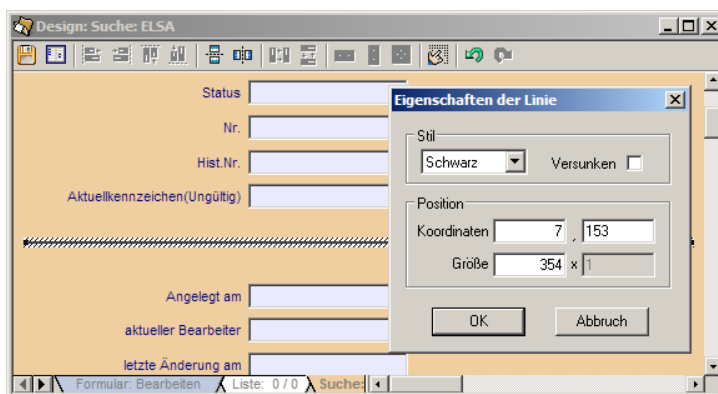
Der Kalender zur Eingabe eines Datums ähnelt hinsichtlich Design und Funktion dem Kalender aus Microsoft Office Outlook.



Der Nutzen für den Anwender: Er ist intuitiver zu bedienen als der bisherige Kalender.

3.4 Linien zur Gestaltung von Formular und Suche

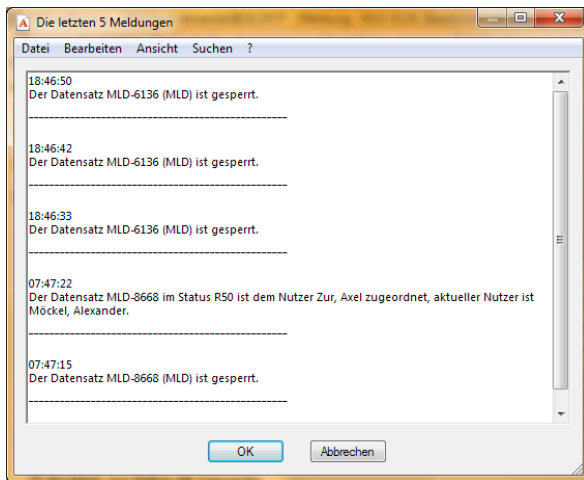
Die Möglichkeiten zur Gestaltung von Dialogen, Formularen und Suchmasken wurden erweitert. Neben Rechtecken, Rahmen, statischen Texten und Bildern lassen sich jetzt auch *vertikale und horizontale Linien* verwenden.



3.5 Erweiterte Anzeige von Meldungen

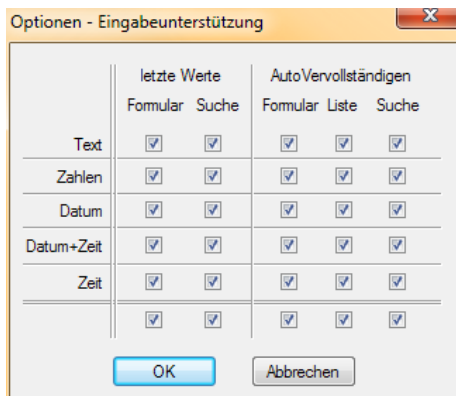
Die Möglichkeiten zur Anzeige von Meldungen der Fachanwendung sind erweitert worden. Für diese Funktionen stehen Kommandos im Menü [Ansicht → Rückmeldungen] zur Verfügung.

- Die letzte Meldung lässt sich erneut anzeigen.
- Alle seit der letzten Anmeldung erhaltenen Meldungen können in einem Protokollfenster angezeigt werden.
- Meldungen sind alternativ zum Meldungsfenster in der Verlaufsleiste anzeigbar.



3.6 Erweiterte Eingabeunterstützung konfigurierbar

Die Eingabe von Werten über die Quick-Liste (Auto-Vervollständigen) und den Dialog „Letzte Werte“ ist konfigurierbar.



Jeder Nutzer hat mit dem Kommando **[Optionen → Eingabeunterstützung]** die Möglichkeit einzustellen, welche Eingabeunterstützung in Formular, Liste und Suche zur Verfügung stehen soll. Die Einstellungen lassen sich für die verschiedenen Datentypen separat vornehmen.

3.7 Sonstiges

- Das Menü **[Zentrale Kommandos]** wird auch in der Liste und der Suche angezeigt.
- Das Umschalten in den Modus „Bearbeiten“ ist mit der Taste **[F2]** möglich.
- Die Hintergrundfarbe von Dialogen lässt sich einstellen.

3.7.1 Suche

- In der Suche wurde die Eingabeunterstützung „Abfrage-Text“ um die Option *exakt* erweitert. Mit dieser Option kann nach Zeichenketten gesucht werden, die Platzhalterzeichen wie * und ? enthalten. Die Platzhalterzeichen müssen in diesem Fall nicht mit dem Zeichen \ maskiert werden.

3.7.2 Liste

- Mit den Tasten **[Strg]+[D]** und dem Kommando **[Darstellung → Auswahl]** kann aus der Liste zu einer beliebigen Suche umgeschaltet werden.
- Der gesamte Inhalt einer Liste lässt sich mit den Tasten **[Strg] + [A]** markieren.
- Mit den Tasten **[Umschalt]+[Strg]+[C]** und dem Kommando **[Bearbeiten → Kopieren mit Überschriften]** können Sie den Inhalt einer Liste einschließlich der Spaltenüberschriften kopieren.

3.7.3 Formular

- In Formularen kann mit den Tasten **[Strg] + [G]** nach Feldern (Prompts) gesucht werden.
- Das fokussierte Feld wird mit derselben Farbe umrahmt, die der Prompt des fokussierten Feldes hat.

4 Für Software-Entwickler

4.1 Neue Sachverhalte

4.1.1 Verwaltung von Konfigurationsdaten

Neue Sachverhalte wurden für die Verwaltung von Daten zur Online-Konfiguration einer Anwendung, für die Online-Katalogpflege und die Verwaltung strukturierter Kenngrößen geschaffen.

Die neuen Sachverhalte können nicht mit einem Bearbeitungsablauf verbunden werden.

Sachverhalt C (KONFIGURATIONSDATEN)

- Parameter: IDENT, DESC, TY_NAME, HNR, *FREE*, HRA_*, Fußabdruck-Parameter
- Parametergruppen sind zulässig
- Historisierung über den Sachverhalt HC

Sachverhalt UC (UNTERGEORDNETE KONFIGURATIONSDATEN)

- Parameter analog zu Sachverhalt C konfigurierbar
- Parametergruppen sind zulässig
- Es sind 5 RoR zum Sachverhalt C konfigurierbar und ein ConUSvh zu UC
- Historisierung über Sachverhalt HUC

4.1.2 View-Untersachverhalte

Als frei definierbare Sachverhalte bezeichnet man die View-Sachverhalte, die eine Sicht auf beliebige Datenstrukturen auf Basis selbst definierter Views ermöglichen.

Ergänzend zu den View-Sachverhalten V und VD wurden die frei definierbaren Untersachverhalte UV und UVD eingeführt.

Der Untersuchverhalt UV ist ein statischer View-Untersachverhalt, an dem nur **FX**-Parameter konfiguriert werden können. Der Untersuchverhalt UVD ein dynamischer View-Untersachverhalt, an dem neben **FX**-Parametern auch **PR**-Parameter zulässig sind.

Beide können als Untersuchverhalt von V, VD, UV und UVD konfiguriert werden. Bei beiden sind Parametergruppen zulässig.

4.2 LOBs in Formularen

Ziel der neuen Funktionalität ist es, die Voraussetzung für ein Dokumentenmanagement als Alternative zum bekannten Dokumentenmanagement (DocM) zu schaffen: LOBs (Large Objects) lassen sich direkt in Formularfeldern speichern, anzeigen und bearbeiten (siehe „Dateien im Formular speichern und anzeigen“).

Im Unterschied zu DocM können nun beliebig viele Parameter zur Speicherung von Dokumenten definiert werden.

Die Fachfunktionalität eines Dokumentenmanagements (Ein- und Auschecken, Transaktionen, Historisierung, Versionierung etc.) muss dagegen in der entsprechenden Fachanwendung implementiert werden.

4.2.1 Konfiguration

Neu in der Parameterbeschreibung sind die Datentypen `blob` und `clob` für `PAD_DATY`: Daten können als BLOB oder CLOB gespeichert werden.

```
PAD . . A 30 PR PIC 'Bild' blob n n . . .
```

Das Parameterattribut `FLTY` bestimmt den Dateityp, der vom Parameter akzeptiert wird.

```
PAT A FLTY 'Picture'
```

Die erlaubten Werte sind:

Attributwert	Beschreibung
RTF	RTF-Anzeige/Editor (Rich Text Format)
TXT	Text-Anzeige/Editor
XML	XML-Anzeige/Editor (Extensible Markup Language)

HTML	HTML-Anzeige/Editor (Hypertext Markup Language)
PDF	PDF-Anzeige/Editor (Portable Document Format)
Word	Microsoft-Office-Word-Anzeige/Editor
Excel	Microsoft-Office-Excel-Anzeige/Editor
PowerPoint	Microsoft-Office-PowerPoint-Anzeige/Editor
Picture	Bild-Anzeige (PNG, BMP, JPG)
Protocol	Protokoll-Anzeige/Editor
Raw	System-Anzeige/Editor

Soll der Parameter als Schaltfläche dargestellt werden, kann über das Parameterattribut PROMPT signalisiert werden, ob Daten hinterlegt sind.

```
PAT A PROMPT 'Leer^Daten'
```

Die weiteren Eigenschaften sind im Element EATW des Elements EMNG zu definieren.

```
EMNGA A 'PAC.PARAM PAD.PIC

EATW A 'Quali' 'X' n n
!Feldinhalt wird an QUALI-Action übergeben. Wertebereich X = Ja; - = Nein

EATW A 'Checkout' 'X' n n
!Feldinhalt muss zur Bearbeitung ausgecheckt werden

EATW A 'ControlLook' 'Button' n n
!Darstellungsart des Controls. Wertebereich:
!Button Alle Aktionen erfolgen über eine Schaltfläche auf dem Formular
!Preview Ansicht direkt im Formularfeld; zum Bearbeiten muss [F9] gedrückt werden
!Direct Ansicht und Bearbeiten des Inhaltes direkt im Formularfeld

EATW A 'Copy' 'X' n n
!Daten dürfen temporär in das Dateisystem geschrieben werden, zum Beispiel zum
!Bearbeiten mit externen Programmen. Wertebereich X = Ja; - = Nein

EATW A 'StyleFiles' 'X' n n
!css bzw. xsl für die Darstellung von HTML oder XML verwenden?
!Wertebereich X = Ja; - = Nein
!css- und xsl-Dateien müssen im Pfad <?*SrbVlg%>\<APApp>\@style liegen

EATW A 'Extensions' 'doc,docx' n n
!Kommaseparierte Liste der zulässigen Dateinamenerweiterungen beim Laden/Speichern

EATW A 'TextLimit' '100000' n n
!Anzahl der Zeichen, die bearbeitet werden dürfen (Standardwert: 32767)
```

4.2.2 Zugriff auf LOB-Felder in Actions

Der Zugriff auf den Feldinhalt mit der QUALI-Action ist wie bei anderen Feldern auch möglich.

```
declare
    recDataSet <#DS()#>;
begin
    <#DS_ReadDIAFields(recDataSet, <#GetSVH()#>, <#GetSVHType()#>)#>;
    if(<#GetQualiField()#> = <#GetDIAName.CLOB_RTF()#>) then
        <#DS_Put(recDataSet, 'CLOB_RTF2', <#DS_GetClob(recDataSet, 'CLOB_RTF')#>)#>;
    end if;
    if(<#GetQualiField()#> = <#GetDIAName.PIC()#>) then
        <#DS_Put(recDataSet, 'PIC2', <#DS_GetBlob(recDataSet, 'PIC')#>)#>;
    end if;
    <#DS_WriteDIAFields(recDataSet)#>;
end;
```

Dasselbe trifft auch für andere Action-Klassen zu:

```
declare
    clobContentclob := null;
    blobContentblob := null;
begin
    <:l_str.CacheKey:> := <#DC_GetKey('C', <*>)#>;
    <#DC_Begin_Read(<:l_str.CacheKey:>)#>;
    <#DC_CallNEWVERS(<:l_str.CacheKey:>)#>;
    clobContent:= <#DC_GetClob(<:l_str.CacheKey:>, 'RTF')#>;
    blobContent:= <#DC_GetBlob(<:l_str.CacheKey:>, 'PIC')#>;
    <#DC_Put(<:l_str.CacheKey:>, 'RTF2', clobContent)#>;
    <#DC_Put(<:l_str.CacheKey:>, 'PIC2', blobContent)#>;
    <#DC_End_Write()#>;
end;
```

4.3 Hierarchische Kataloge

4.3.1 View-Definition

```
create or replace
view cat_v_h(
  HIERA_VALUE --VARCHAR2
, HIERA_LABEL --VARCHAR2
, HIERA_IDENT --VARCHAR2; not null
, HIERA_PAR_IDENT --VARCHAR2
, HIERA_ORD --NUMBER; not null
, HIERA_ICON --CHAR
, HIERA_AKT--CHAR
) as select
  o.o_NAME
, o.o_DESC
, o.o_IDENT
, o_o.o_IDENT
, o.o_id, 'RT_B'
, o.o_AKT
from IBKAP2_cd_o o, IBKAP2_cd_o o_o
where o.o_opa_name = 'CODECAT'
  and o.o_ol_id = o_o.o_id(+)
```

4.3.2 Konfiguration

Ein hierarchischer Katalog wird wie jeder andere freie Katalog (Kategorie **FREE**) definiert.

```

PVS      A      FREE      CODECAT 'Code' N      'CODE' .      'cat_v_h'      'D'      .
  PAD     A      n        PVS      AKT      'Gültig'      c      1      .
    PAT   A      PROMPT  'Gültig'
  PAD     A      6        PVS      ICON     .      str      4      .
  PAD     A      n        PVS      IDENT   'Ident' str      2000   .
    PAT   A      PROMPT  'Ident'
  PAD     A      n        PVS      LABEL   'Bezeichnung' str      2000   .
    PAT   A      PROMPT  'Bezeichnung'
  PAD     A      5        PVS      ORD     .      n      9      .
  PAD     A      n        PVS      PAR_IDENT 'Parent' str      2000   .
    PAT   A      PROMPT  'Parent'
  PAD     A      n        PVS      VALUE   'Key' str      2000   .
    PAT   A      ACCESS  'N'      PAT     A      FCT     '#'      PAT     A      PROMPT 'Key'

```

Aus dem Katalog wird erst dadurch ein hierarchischer, dass die Syntax für die hierarchische Darstellung der Werte im Element EATW des Elements EMNGA (Attribut `Hiera.Syntax`) festgelegt wird.

Die weiteren Attribute sind optional.

- `Hiera.MinLevel` bestimmt, dass der im Katalog gewählte Wert mindestens das angegebene Level innerhalb der Hierarchie haben muss.
- `Hiera.WithIcon` legt fest, dass vor jedem Katalogwert ein Icon angezeigt wird.

```

EMNGA    A      'PVS.CODECAT'
  EATW    A      'Hiera.Syntax'      '{0}.{1}.{n}' n      n
  EATW    A      'Hiera.MinLevel'    '2'      n      n      .
  ..EATW  A      'Hiera.WithIcon'    'X'      n      n      .

```

4.4 Anzeige von Sachverhaltsarten sperren

Anforderung:

Häufig gibt es die Anforderung, Mitarbeitern bestimmte Sachverhaltsarten nicht anzuzeigen. Das Ziel kann man erreichen, indem man allen anderen Arten, die sichtbar sein sollen, Nutzergruppen zuordnet, die zugleich allen Mitarbeitern zugeordnet werden, die die Sachverhaltsarten sehen dürfen.

Es gibt jedoch Fälle, in denen dieser Weg nicht praktikabel ist - beispielsweise auf Grund des Umfanges der Anwendung.

Lösung:

- Konfiguration einer Nutzergruppe mit dem Attribut `ExcludeTy`
- Zuordnung der zu sperrenden Arten zu dieser Nutzergruppe
- Zuordnung der Nutzergruppe zu Mitarbeitern, die die Sachverhaltsarten nicht sehen dürfen
- Mit dem Standardmakro `<#checkTYVisibility(i_strSvh, i_strTy) #>` kann überprüft werden, ob die Art bzw. der gesamte Sachverhalt an der Oberfläche ausgeblendet ist. Das kann beispielsweise in GoTo-Kommandos notwendig sein.

Beispiel:

```
USG  A      n      HIDDEN_TY      'Ausgeschlossene Arten'      .      L      .
EMNGA  A      'USG.HIDDEN_TY'
EATW  A      'ExcludeTy'      'Y08,Y09,Y10'  n      n
```

Ergebnis:

- Art erscheint nicht im Menü und nicht im Dialog zur Auswahl der Sachverhaltsart.
- Suchen zu dieser Art erscheinen nicht im Dialog zur Auswahl der Suche.
- Datensätze dieser Art werden auch über Standard-Suchen nicht gefunden.
- Sachverhalte, zu denen für einen Nutzer alle Arten ausgeblendet sind, erscheinen nicht im Menü.

4.5 Zugriffsrechte (USVH ACS)

Anforderung:

Es bestand die Anforderung für eine mehrstufige Rechtesteuerung, bei der mehrere Regeln für Zugriffsrechte in einer festgelegten Reihenfolge ausgewertet werden sollen. Die Lösung musste abwärtskompatibel sein, d.h. ohne Änderung des DABS-Dialektes auskommen.

Lösung:

Die Sortierung der ACS-Regeln erfolgt an Hand einer internen Ordnungsnummer, die beim Laden der Regeln generiert wird.

Das bedeutet: Nach ACS-Regeln, die nicht entscheidbar sind (`ACS_DEF_ACCESS = *`), wird für die Suche der nächsten auszuwertenden Regel die Reihenfolge der ACS-Zeilen im CON- bzw. RTC-File verwendet.

Die richtige Reihenfolge bei der Auswertung der Regeln muss also durch die Reihenfolge in der CON-Datei sichergestellt werden.

Beispiel:

```
ACS    A      'PAD.P_FREE1' 'AZ'   'PAD'   'P_FREE2'   'A'    '*'    .
ACS    A      'PAD.P_FREE1' 'ST'   'PAD'   'P_FREE2'   'E'    '-'    .
```

- Wenn `P_FREE1 = "AZ"`, dann wird `P_FREE2` nur angezeigt
- Ist `P_FREE1` nicht `"AZ"`, dann ist die Regel auf Grund von `*` für den Sonst-Fall nicht entscheidbar. Also wird die nächste Regel für die Auswertung gesucht
- Wenn `P_FREE1 = "ST"`, dann besteht bei `P_FREE2` Eingabepflicht.
- Ist `P_FREE1` weder `"AZ"` noch `"ST"`, wird `P_FREE2` ausgeblendet, da der Sonst-Fall mit `-` eindeutig definiert ist.

4.6 Freie Kommandos und die Rolle LESER

Für AP/Client-Nutzer wurde die Rolle LESER eingeführt (siehe „Für Administratoren“).

Diese Rolle gibt Nutzern das Recht, in allen Sachverhalten Daten zu sehen und die Kommandos auszuführen, die mit dem Recht „show“ versehen sind.

Im diesem Zusammenhang ist zu beachten, dass freie Kommandos, die keine explizite Definition eines Ausführungsrechtes haben, automatisch mit dem Recht „show“ versehen werden. Diese können somit von Nutzern mit dieser Nutzergruppe auch ausgeführt werden. Dadurch besteht die Möglichkeit, dass Nutzer mit der Rolle LESER Daten verändern.

Um das zu verhindern, müssen freie Kommandos mit einem anderen Recht als „show“ definiert werden, beispielsweise mit dem Recht „update“:

```
EMNGA  A      'TY.AUFTRAEGE>CMM.CALCULATE'
      EATW  A      'Right' 'U'      n      n
```

4.7 Standard-Makros für RMI

Zweck dieser Makros ist der Aufruf von Methoden in einem anderen AP-Kern, der auch auf einer entfernten Datenbank liegen kann (Remote Method Invocation - RMI). Die Makros sind auch für das lokale System verwendbar.

Einloggen in ein anderes System und Remote-Session eröffnen:

```
<#RMI_Login( i_strSystem      -- Systemname des Remote-Systems; Default=null
             , i_strDBLink    -- Database-Link zum Remote-System; Default=null
             , i_strUserName  -- Nutzer zur Anmeldung am lokalen System; Default=null
             )#>;
```

Ausloggen und Remote-Session schließen:

```
<#RMI_Logout()#>;
```

Wechseln der aktuellen AP-Umgebung in der Remote-Session:

```
<#RMI_SetApp( i_strAPApp
              , i_strDdr      -- Default=null
              , i_strDBUsr    -- Default=null
              , i_strSys      -- Default=null
              , i_strRtsPrj   -- Default=null
              )#>;
```

Wiederherstellen der vorherigen AP-Umgebung in der Remote-Session:

```
<#RMI_ResetApp()#>;
```

Methoden ermitteln und aufrufen:

```
-- gibt den PL/SQL-Namen der Methode i_strMTH auf dem Remote-System zurück:
<:l_str.DbfMth:> := <#RMI_GetMTHDBName(i_strMTH) #>;

-- liefert den vollqualifizierten Namen von i_strObject in der aktuellen RMI-Session in
-- der Form Objektname@Systemname
<:l_str.DbfMth:> := <#RMI_GetDbfObjectName(i_strObject) #>;

-- ruft Methode im Remote-System auf
<#RMI_CallMTH(<:l_str.DbfMth:>, i_nParaCount, i_strPara1, [...]) #>;
```

Debug-Level in der Remote-Anwendung setzen:

```
<#RMI_SetDebugLevel(i_nLvl          --0=nie,1=grob,2=mittel,3=fein
                   , i_strTitle    --Default=null
                   ) #>;
```

Protokoll aus der Remote-Anwendung kopieren:

```
<#RMI_CopyProtocol(p_nPrintID
                  , i_nRemotePrintId
                  , i_bDelete      -- Protokoll remote löschen; Default=true
                  ) #>;
```

Stacktrace aus der Remote-Anwendung kopieren:

```
<#RMI_CopyStackTrace() #>;
```

4.8 Standard-Makros für Protokollierung

Die neuen Makros bezwecken, dass

- der Typ eines Protokoll-Eintrags erkennbar ist (beispielsweise Warnung, Fehler, Debug oder Info),
- im Protokoll neben einer für den Anwender lesbaren Form auch für andere Programme auswertbare Informationen abgelegt werden können,
- Informationen aus dem Stacktrace in vereinfachter Form in das Protokoll ausgegeben werden können.

Erweiterte persistente Debug-Information schreiben

```
<#DebugInfoExt( i_strType          -- N, T, S[<n>], D[<n>]
                , i_nLvl          -- -1=immer,0=nie, 1=grob, 2=mittel, 3=fein
                , i_strInfoType    -- Typ - z.B: E,W,I (Error, Warning, Info)
                , i_strInfoPara    -- Info für andere Programme in Protokoll schreiben
                , i_strFormat
                [, i_strPara1
                , i_strPara10]
                ) #>;
```

Die Parameter *i_strType*, *i_nLvl*, *i_strFormat* und *i_strPara1* bis *i_strPara10* entsprechen dem Makro `DebugInfo` (siehe Nutzerhandbuch AP/Macro 2.12).

Erweiterten persistenten Protokolleintrag schreiben:

```
<#PrintPersistentExt(i_strInfoType      -- Typ - z.B: E,W,I (Error, Warning, Info)
                    , i_strInfoPara    -- Info für andere Programme in Protokoll schreiben
                    , i_strFormat
                    [, i_strPara1
                    , i_strPara10]
                    )#>;
```

Die Parameter *i_strFormat* und *i_strPara1* bis *i_strPara10* entsprechen dem Makro `DebugInfo` (siehe Nutzerhandbuch AP/Macro 2.12).

Erweiterten Protokolleintrag schreiben:

```
<#PrintExt(
            i_strInfoType      -- Typ - z.B: E,W,I (Error, Warning, Info)
            , i_strInfoPara    -- Info für andere Programme in Protokoll schreiben
            , i_strFormat
            [, i_strPara1
            , i_strPara10]
            )#>;
```

Die Parameter *i_strFormat* und *i_strPara1* bis *i_strPara10* entsprechen dem Makro `DebugInfo` (siehe Nutzerhandbuch AP/Macro 2.12).

Returncode und -meldung sowie Callstack-Inhalt persistent auf das Protokoll schreiben:

```
<#PrintStackTrace(  <#GetError()#>
                   , <#GetMessage()#>
                   , <#GetMTH()#>
                   , <#GetStmt()#>
                   , i_strCmd          -- I=Ibykus, O=Oracle-Callstack, default=IO
                   )#>;
```

4.9 Standard-Makros für Debugging

Die Makros dieses Komplexes sollen es einfacher machen, Debug-Meldungen in Methoden auszugeben.

Außerdem soll visualisiert werden, in welcher Ebene der Aufruf-Hierarchie sich eine Methode befindet.

Debug starten:

Die Routine `DebugBegin` schreibt eine „Debug-Begin-Zeile“ in das Protokoll. Name der Methode, Debug-Level und Prefix werden gemerkt.

```
<#DebugBegin(i_nMthLvl          -- Level; 1, 2, 3
             , i_strMth
             , i_strFormat      --Text der Debug-Begin-Zeile
             [, i_strPara1
             , ...
             , i_strPara10
             , i_strPrefix ]    -- Zeichen für den Zeilenanfang einer Protokollzeile
             )#>;
```

Anschließend kann ein PL/SQL-Block geöffnet werden.

Schreiben einer Debug-Information innerhalb eines Debug-Blockes:

Innerhalb eines Debug-Blocks (also zwischen `DebugBegin` und `DebugEnd`) kann mit `DebugStep` eine Debug-Information zur aktuellen Methode geschrieben werden.

```
<#DebugStep( i_strStepNr -- Text für Schritt in der Protokollzeile
             , i_strFormat --Text der Debug-Info
             [, i_strPara1
             , ...
             , i_strPara10]
             )#>;
```

Die Parameter *i_strFormat* und *i_strPara1* bis *i_strPara10* entsprechen dem Makro `DebugInfo` (siehe Nutzerhandbuch AP/Macro 2.12).

Die Protokollzeile wird nach folgendem Muster gebildet:

```
<Prefix> || ' Step: ' ||<Methodenname> ||' -' ||i_strStepNr||'- ' i_strFormat
<Prefix> wurde mit DebugBegin definiert, <Methodenname> wird automatisch eingesetzt.
```

Debug beenden:

Ohne Meldungstext

```
<#DebugEnd()#>;
```

Mit Meldungstext

```
<#DebugEnd(i_strFormat [, i_strPara1, ..., i_strPara10])#>;
```

Alle Parameter entsprechen dem Makro `DebugInfo` (siehe Nutzerhandbuch AP/Macro 2.12).

Debug-Level speichern und wiederherstellen:

Debug-Level und AP-Debug-Bits werden vor Beginn eines PL/SQL-Blocks mit `StoreDebugLevel` gespeichert und nach dem Ende des PL/SQL-Blocks mit `RestoreDebugLevel` wiederhergestellt.

Die beiden Methoden müssen gemeinsam verwendet werden, da sie den Anfang und das Ende eines PL/SQL-Blocks bilden.

```
<#StoreDebugLevel () #>;
```

```
<#RestoreDebugLevel () #>;
```

5 Für Administratoren

Anforderung:

Es soll die Möglichkeit geben, Nutzern ausschließlich lesenden Zugriff auf die Daten einer Fachanwendung zu geben.

Lösung:

Es wurde die Rolle LESER für AP/Client-Nutzer eingeführt. Dieser Rolle ist für alle Sachverhalte das Recht „show“ gegeben. Das heißt, dass Nutzer mit der Rolle LESER in allen Sachverhalten Daten sehen und die Kommandos ausführen dürfen, die mit dem Recht „show“ versehen sind.

HINWEIS: Bei der Entwicklung von Anwendungen ist zu beachten, dass freie Kommandos standardmäßig von Nutzern mit der Rolle LESER ausgeführt werden können (siehe „Freie Kommandos und die Rolle LESER“).

6 Weitere Unterstützung

Professional Training:

Eine umfassende Darstellung aller funktionalen Neuerungen von IBYKUS AP/ 2.13 ist im Rahmen unserer Release Notes nicht zu leisten.

Wenn Sie Fragen zu einzelnen Bedien- und/oder Programmierelementen haben, so wenden Sie sich bitte an Ihren Ansprechpartner bei IBYKUS.

Unser [Professional Training](#) bietet außerdem regelmäßig Schulungen zur Thematik an. Auf Anfrage teilen wir Ihnen gerne mit, wann das nächste Seminar zu den „New Features“ geplant ist.

HINWEIS: Über aktuelle Schulungstermine können Sie sich auch über unseren [Online-Schulungskalender](#) informieren, den wir quartalsweise aktualisieren.
