Programmierrichtlinie SAP

Contents

1 Einleitung

SAP-Anwendungen liefern immer einen gewissen Standard aus. Häufig genügt dieser Standard nicht den Anforderungen des Unternehmens. Einiges kann mit Hilfe von Customizing erledigt werden, für andere Anpassungen sind Entwicklungen notwendig.

Mit diesem Dokument werden Vorgaben für die Durchführung von Entwicklungen definiert. Diese sind notwendig, damit eine gleichbleibende Qualität der Entwicklungen gewährleistet werden kann. Außerdem wird durch die Definition der Standards eine einfachere, bessere Wartbarkeit des Gesamtsystems erreicht.

2 Geltungsbereich

Jeder, der in den SAP-Systemen der FFT Gruppe Anpassungen vornimmt, hat sich an die Vorgaben in diesem Dokument zu halten. Dabei greifen alle Vorgaben sowohl für interne Entwickler, als auch für externe Entwickler von Beratungshäusern.

3 Vorgehensweise

Bevor ein Entwickler in den Systemen der FFT Gruppe mit der Entwicklung beginnt, ist dieses Dokument zu lesen. Etwaige notwendige Entwicklerschlüssel werden vom entsprechenden Entwicklungsverantwortlichen bei der SAP angefordert und in den Systemen hinterlegt. Ferner werden die notwendigen Berechtigungen für die Entwickler in den benötigten Entwicklungssystemen hinterlegt.

3.1 Prozessbeteiligte

3.1.1 Systemverantwortlicher

Hierunter ist der Teamleiter des IT-Teams IT SAP Customer Competence Center zu verstehen. Zurzeit ist dies:

- Daniel Betz

3.1.2 Entwicklungsverantwortlicher

Hierunter ist der Teamleiter des IT-Teams IT Development zu verstehen. Zurzeit ist dies:

- Michael Klug

3.1.3 Modulbetreuer / Keyuser

Hierunter werden die Kollegen des IT-Teams SAP Customer Competence Centers verstanden.

Modul	Modulbetreuer	Keyuser
CO – Controlling	Tristan Maul	
FI – Finanzbuchhaltung	Tristan Maul	
HR – Personalwesen	Daniel Betz	
MM – Materialwirtschaft, Einkauf	Timo Orth	
PP – Produktionsplanung, Fertigung	Timo Orth	
PS – Projektsystem	Doreen Rogalski	
SD – Vertrieb	Doreen Rogalski	
BW – Business Warehouse	Laura Kohaupt	

3.1.4 Entwickler

Hierunter sind die Mitglieder des IT-Teams IT Development zu verstehen. Zurzeit sind dies:

- Frank Schäfer
- Laura Kohaupt
- Michael Klug

Ferner werden hier auch studentische Aushilfen und Auszubildende verstanden, die die Entwickler unterstützen. Ebenso gehören externe Berater dazu, sofern diese Entwicklungsaufgaben übernehmen.

3.2 Ablauf

Vor jeder Entwicklung ist zu prüfen, ob der SAP Standard eine Lösung für die jeweilige Problemstellung bietet. Diese Prüfung ist vom jeweiligen Modulverantwortlichen des SAP Customer Competence Centers der FFT IT durchzuführen. Erst wenn der Modulverantwortliche keine Lösung im Standard oder mit Customizing sieht, werden die Anforderungen durch den Modulbetreuer in Zusammenarbeit mit dem jeweiligen Keyuser für die Entwickler aufbereitet.

Der Entwickler prüft auf Basis der Anforderungen des Modulbetreuers ob und wie die entsprechende Anforderung umgesetzt werden kann. Hierbei sind zunächst bestehende Erweiterungsmöglichkeiten (User-Exits, Customer-Functions, BAdIs, Enhancements,…) zu prüfen. Erst wenn auch hier keine Möglichkeiten der Anpassung vorliegen, ist ein eigenes Programm zu erstellen. Sofern Modifikationen im SAP-Standard-Coding notwendig werden, sind der Entwicklungsverantwortliche und der Systemverantwortliche vorher um Freigabe zu bitten.

3.3 Allgemeine Rahmenbedingungen / Vorgaben

3.3.1 Grundsätze

Bei allen Entwicklungen ist sicherzustellen, dass Coding möglichst nicht doppelt implementiert wird. Hierdurch soll eine verbesserte Wartbarkeit des Systems erreicht werden.

Die Verwendung von SAP-APIs wie z.B. BAPIs, Funktionsbausteine, Klassen, etc. ist immer eigenen Datenbankzugriffen vorzuziehen. Dies gilt insbesondere bei der Verbuchung von Daten.

Generell sollte bei Entwicklungen immer auch das Team Berechtigungen mit einbezogen werden, um sicherzustellen, dass alle Anforderungen hinsichtlich Zugriffsschutz von Daten eingehalten werden.

3.3.2 Ablauf von Entwicklungen

Entwicklungen werden generell per Ticket angefordert. Ebenso werden auch Fehler per Ticket gemeldet. Sofern der Anwender selbst kein Ticket erstellt, wird vom jeweiligen Modulbetreuer ein entsprechendes Ticket für die Entwickler eröffnet. Die Tickets werden vom Modulbetreuer in die Ticket Queue „SAP-Entwicklungen“ überführt. Diese dient allen Entwicklern als Arbeitsliste.

Die Entwicklungen finden im Entwicklungssystem auf dem Mandanten 100 statt. Für Entwicklertests kann der Mandant 800 im Entwicklungssystem verwendet werden. Sobald der Entwickler die Entwicklungen abschließend implementiert und getestet hat, erfolgt die Übernahme der Transporte in das Qualitätssicherungssystem.

Im Qualisätssicherungssystem erfolgt der Test durch den Keyuser. Der Keyuser bestätigt den erfolgreichen Test durch entsprechende Kommentare im Ticket. Die finale Übernahme ins Produktivsystem erfolgt in Abstimmung zwischen Entwickler, Modulverantwortlichem und ggf. dem Keyuser.

Alle Tests haben mit entsprechend berechtigten Testusern zu erfolgen. Etwaige Berechtigungsanpassungen sind entsprechend abzustimmen.

Der Keyuser übernimmt die fachliche Dokumentation der Entwicklungen und auch die entsprechenden Schulungen. Ferner informiert er die Anwender über die entsprechenden Änderungen.

3.3.3 Definition von SAP Modulen

Generell werden die SAP-Anwendungskomponenten in der Bezeichnung von Objekten verwendet. Diese werden hier festgelegt und werden im weiteren Verlauf auch mit Modulen bezeichnet.

Falls Eintragungen hier fehlen, dann immer die offiziellen SAP Anwendungskomponenten benutzen.

SAP Anwendungskomponente (Modul)	Bezeichnung

BC	Basis-Komponenten
CA	Anwendungsübergreifende Komponenten
CO	Controlling
HR	Personalwesen
FI	Finanzbuchhaltung
LE	Logistics Execution
MM	Materialwirtschaft
SD	Vertrieb
PP	Produktionsplanung und -steuerung
PS	Projektsystem

3.3.4 Bezeichnung von Transportaufträgen

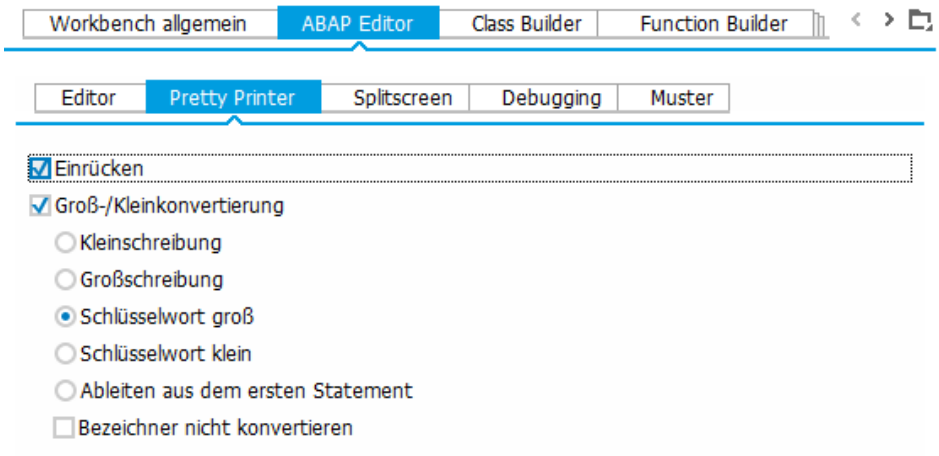
Alle Entwicklungen inkl. Korrekturen erfolgen auf Basis eines Tickets aus dem jeweils bei FFT eingesetzten Ticketsystem. Die SAP-Transporte werden mit der entsprechenden Ticketnummer, gefolgt von einem Bindestrich und einer sprechenden Beschreibung bezeichnet.

Beispiel:

2024060583990039 – Inhaltliche Beschreibung

3.3.5 Pretty Printer

Um eine einheitliche Formatierung zu erreichen ist der Pretty Printer der SAP wie folgt einzustellen:



3.3.6 Mehrsprachigkeit / Originalsprache

Alle Entwicklungen und das gesamte Customizing sind auf Mehrsprachigkeit auszulegen. Als Systemsprachen werden Deutsch und Englisch fest vorgeschrieben. Somit sind alle Texte für Endanwender zeitgleich mit der Entwicklung zu übersetzen.

Die Dokumentation im Coding kann in Deutsch erfolgen, da die SAP-Entwicklungsressourcen in Deutschland sitzen. Hieraus ergibt sich die Originalsprache Deutsch für alle Entwicklungen.

Texte im Quelltext sind zusätzlich als Textvariable zu definieren und zu übersetzen, um die Lesbarkeit zu erhöhen.

Beispiel:

```
lv_text = ‚E-Mail'(001)
```

3.3.7 Meldungstexte

Um Mehrsprachigkeit sicherzustellen, ist die Verwendung von Nachrichten (SE91) eigenen Textausgaben vorzuziehen.

3.3.8 Deklaration von Variablen

Variablen werden immer zu Beginn einer Modularisierungseinheit deklariert.

3.3.9 Kopfzeilen

Auf die Nutzung von internen Tabellen mit Kopfzeilen ist zu verzichten.

3.3.10 Programmierung von Datenbank-Updates

Datenbank-Updates sind zu kapseln. Wenn SAP-Standardbelege angelegt oder verändert werden, sollten BAPIs verwendet werden.

Bei Änderungen an Datensätzen auf der Datenbank müssen Sperren durch die Nutzung von ENQUEUE-/ DEQUEUE-Bausteinen aufgebaut werden.

Verbuchungsbausteine können zur Systementlastung eingesetzt werden.

3.3.11 Berechtigungsprüfungen

Programme sind immer mit einer eigenen Transaktion zu versehen, damit sie nicht über die SAP-Transaktionen SA38/SE38 gestartet werden müssen.

Prüfungen auf Organisationseinheiten sind entsprechend der Definition im Fachkonzept einzubauen.

Berechtigungen dürfen nie über eigene Entwicklungen (Methoden, Tabellen, …), sondern müssen immer über Berechtigungsobjekte geprüft werden.

Transaktionen sind immer durch „CALL TRANSACTION ... WITH AUTHORITY-CHECK“ durchzuführen.

Die zu verwendenden Berechtigungsobjekte sind bei der technischen Konzeption der Anwendung mit der Berechtigungsadministration (orga@fft.de) abzustimmen.

Muss ein neues Objekt angelegt werden, sollte das Design (die Felder) gemeinsam mit der Berechtigungsadministration entwickelt werden.

Sind Berechtigungsprüfungen z.B. für Organisationseinheiten wie Buchungskreis, Werk o.ä. erforderlich, so sollte wenn möglich eine Prüfung im Selektions-Dialog (also bei Eingabe der Selektionskriterien) erfolgen.

Ist es notwendig, auf Ebene der Datensätze zu prüfen (z.B. FI-Belegart oder Kostenstelle die nicht im Selektionsbild vorhanden sind), sollten die Sätze mit einem Hinweis auf die fehlende Berechtigung vor der Ausgabe entfernt werden.

3.3.12 Erweiterungen

Das SAP-System stellt für die Erweiterung des Standards verschiedene Möglichkeiten der Erweiterung zur Verfügung:

- "klassische" User-EXITS in Form von Modifikationen
- Customer-Functions (SMOD / CMOD)
- "klassische" BAdI's (SE18 / SE19)
- "neue" BAdI's im Rahmen des Ehancement Frameworks (SE18 / SE19)
- Enhancement Framework mit expliziten und impliziten Erweiterungsmöglichkeiten

Modifikationen sind generell zu vermeiden und nur nach Rücksprache mit dem SAP-Entwicklungsverantwortlichen zu tätigen. Dieser kann auch die entsprechenden Registrierungsschlüssel bei der SAP beantragen.

Ab S/4HANA sind diese Registrierungsschlüssel nicht mehr notwendig. Gerade deshalb ist hier zwingend Rücksprache zu halten, um eine etwaige Realisierungsalternative im Vieraugenprinzip zu analysieren.

3.4 Benennung von Objekten

Die Benennung aller DDIC-Objekte sollte, wo immer möglich, dem in der folgenden

Tabelle dargestellten einheitlichen Muster folgen:

Position	Beschreibung
1	Z
2 - 3	Anwendungskomponente
4	_
5 - 30	Beschreibung

Beispiel:

ZSD_INVOICE

Grundsätzlich sollten „sprechende“ und damit „selbstdokumentierende“ Namen vergeben werden.

In den folgenden Unterkapiteln werden einige sinnvolle bzw. aus SAP-Vorgaben resultierende Abweichungen und Hinweise zur hier aufgestellten Regel aufgeführt.

Ist ein Objekttyp nicht weiter erläutert, dann gilt die obige Regel.

3.4.1 Data Dictionary

3.4.1.1 Datenbanktabellen

Der gesamte Name darf laut SAP DDIC nicht länger als 16 Zeichen betragen. Es ist daher auf eine verständliche Benennung von Datenbanktabellen zu achten.

Jede Kundentabelle sollte als erste Spalte den Mandanten beinhalten. Davon sollte nur abgewichen werden, falls die Daten nachweislich mandantenunabhängig abgelegt werden sollen.

Auslieferung und Pflege:

Die Datenbanktabelle muss ihrem Verwendungszweck entsprechend als Stammdaten-, Bewegungsdaten- oder Customizing-Tabelle markiert werden. Die „Auslieferungsklasse“ ist entsprechend der Nutzung der Tabelle zu setzen.

Unter „Data Browser/Tabellensicht-Pflege“ ist, sofern möglich, „Pflege Anzeige/ Pflege erlaubt“ einzustellen, insbesondere bei Customizing-Tabellen. Eine Erweiterungskategorie ist zu setzen.

Für Customizingtabellen ist ein Pflegeview zu erstellen. Ferner soll eine Parametertransaktion angelegt werden, die anschließend in das Z-Customizing-Menü der Anwendungskomponenten aufgenommen wird.

3.4.1.2 Pflegeviews

Ein Pflegeview wird wie die zugehörige Basistabelle benannt sein, jedoch mit dem Sufix _V.

3.4.1.3 Strukturen und Tabellentypen

Zur Unterscheidung von Datenelementen, Strukturen und Tabellentypen werden Strukturen durch ein _S und Tabellentypen durch ein _T in der Bezeichnung gekennzeichnet.

3.4.1.4 Appends an Tabellen

Zur Erweiterung von SAP-Standardtabellen mit eigenen Felder dürfen ausschließlich Appends verwendet werden. Somit handelt es sich nicht um eine Modifikation, sondern um eine Erweiterung.

Eine Ausnahme bilden Customer-Includes, die im Rahmen von Customer-Functions durch die SAP ausgeliefert werden. Hier wird die vorgegebene Konvention beginnend mit CI übernommen.

Felder in Appends müssen generell mit ZZ beginnen.

3.4.1.5 Appends an Suchhilfen

Um Suchhilfen um eigene "Registerkarten" zu erweitern, sind Suchhilfen-Appends zu verwenden. Dieser Append ist wie folgt zu benennen:

Position	Beschreibung
1 - 2	ZA
4 - 30	Original Name der Suchhilfe

Beispiel:

ZAPREM

3.4.1.6 Suchhilfen

Eigene Suchhilfen werden generell nach dem folgenden Schema benannt:

Position	Beschreibung
1	Z
2 - 3	Anwendungskomponente
4	_
5 - 8	SHLP
9	_
10 - 30	Beschreibung

Beispiel:

ZSD_SHLP_VBAK

3.4.2 Objektorientierte Entwicklung

3.4.2.1 Klasse / Interface

Position	Beschreibung
1	Z
2 - 3	CL für Class CX für Ausnahmeklassen IF für Interface
4	_
5 - 6	Anwendungskomponenten
7	_
8 - 30	Beschreibung

Beispiel:

ZCL_SD_INVOICE

Die gleiche Logik ist auch für lokale Klassen zu verwenden, wobei das führende Z durch ein L für lokal zu ersetzen ist.

Beispiel:

LCL_SD_INVOICE

3.4.2.2 Methoden

Namenskonvention:

- DO_ für Aktionsmethoden.
- GET_ für das Lesen von Informationen. Im Regelfall sollte neben den Importing-Parameter ein Returning-Parameter deklariert werden und auf Exporting-Parameter nur in besonderen Situationen (mehrere Rückgabewerte) zurückgegriffen werden.
- SET_ für das Schreiben von Informationen.
- IS_ für das Abfragen von Zuständen. Rückgabe zwingend mit Returning-Parameter vom Typ ABAP_BOOL oder analog (beispielsweise WDY_BOOL bei WebDynpro).

Sichtbarkeit:

Es dürfen nur ausgewählte Methoden von außen aufgerufen werden. Diese erhalten die Sichtbarkeit public. Dabei ist auf die Erhaltung von Schnittstelle (also Methodenname, Parameter, Logik) und des Contracts (Pre- und Post-Condition) zu achten.

Methodenart:

Eine Methode sollte im Regelfall als Instanz-Methode deklariert werden. Nur bei begründetem Bedarf sind statische Methoden zu nutzen.

Attribute und Methodenparameter:

Siehe Kapitel "Benennung von Variablen"

3.4.2.3 Events

Namenskonvention:

EVT_ für selbst definierte Events.

3.4.3 Funktionale Entwicklung

3.4.3.1 Programm / Report / Modulpool

Obwohl Namen von Reports bis zu 30 Zeichen betragen dürfen, sollten nur 25 davon verwendet werden, da nur so zugehörige Includes entsprechend der Namenskonvention benannt werden können.

3.4.3.2 Includes

Position	Beschreibung
1 - 25	Name des inkludierenden Programms
26	_
27 - 30	TOP für TOP-Include SEL für Selektionsbild F01 für FORM-Routinen I01 für PAI-Module O01 für PBO-Module C01 für lokale Klassen-Implementierungen

Beispiele:

ZSD_INVOICE_TOP

ZSD_INVOICE_F01

ZSD_INVOICE_SEL

3.4.4 Funktionsbausteine

Aufgrund von SAP-Vorgaben muss für den Namen von Funktionsbausteinen an zweiter Stelle ein Unterstrich gesetzt werden.

Position	Beschreibung
1 - 2	Z_
3 - 4	Anwendungskomponente
5	_
6 - 30	Beschreibung

Beispiel:

Z_SD_INVOICE_GET_DETAILS

3.4.5 Nachrichtenklassen

Nachrichtenklassen sind ähnlich der Logik von Paketen zu behandeln. Alle logisch bzw. inhaltlich zusammengehörigen Nachrichten sind in einer Klasse zu bündeln.

3.5 Benennung von Variablen

Die Benennung von Variablen sollte soweit möglich dem SAP-Standard folgen. Für die Lesbarkeit und Eindeutigkeit sind ergänzende Namensbestandteile zu nutzen. Diese sollte jedoch aus Gründen der Handhabbarkeit so kurz wie möglich sein, ohne ihren Charakter einzubüßen.

Die folgenden Vorschläge orientieren sich an den Vorgaben der SAP:

3.5.1 Lokales Coding

1. Position		2. Position		3. Position	ab 4. Position
L	Local	V	Variable (Field)	_	Beschreibung
G	Global	C	Constant		
S	Static Variable	S	Structure		
		T	Table / Ranges		
		R	Reference		
		O	Object Reference		

Beispiele:

lv_vkorg für lokale Variable

lt_vbak für lokale Tabelle

ls_vbak für lokale Struktur

lo_order für lokale Objekt-Instanz

3.5.2 Schnittstellenparameter

1. Position					2. Pos.		3. Pos.	4. Position
		Unter-programm	**Funktions baustein**	**Klassen-methode**				
I	Importing / Using	X	X	X	V	Variable (Field)	–	Beschreibung
C	Changing	X	X	X	C	Constant		
E	Exporting		X	X	S	Structure		
R	Returning			X	T	Table / Ranges		
T	Tables (**obsolet**)	X	X		R	Reference		
EX	Ausnahmen		X		O	Object Reference		

Beispiele:

iv_vkorg für Import Parameter einer Variable

ct_vbak für Changeing Parameter einer Tabelle

es_vbak für Exporting Parameter einer Struktur

ro_order für Returning Parameter einer Objekt-Instanz

Da Tabellenparameter sowohl Import-, Export-, als auch Changingparameter sein können, werden diese entsprechend der Verwendung im Programm mit IT_, ET_ oder CT_ bezeichnet. Mittlerweile sind Tabellenparameter von der SAP als **obsolet** gekennzeichnet und sollten nicht mehr verwendet werden.

Da Funktionsbausteine innerhalb von Funktionsgruppen abgelegt werden, sind alle Variablen in Funktionsbausteinen als lokale variablen anzusehen. Daraus folgt, dass alle Variablen in Funktionsbausteinen mit l anfangen.

Werden Variablen innerhalb einer Funktionsgruppe in verschiedenen Funktionsbausteinen verwendet, so werden diese im Rahmenprogramm der Funktionsgruppe als globale Variablen angelegt und beginnen mit g.

Falls mehrere Funktionsbausteine für ein Anwendungsfall erzeugt werden muss, ist zu prüfen, ob eine neue Funktionsgruppe sinnvoll ist, da dadurch Abhängigkeiten, bzw. Blockaden zwischen mehreren Transporten und Entwickler vermieden werden.
Hinweis: Vorsicht beim Transportieren von Funktionsgruppen!

3.5.3 Selektionsbilder

Position	Beschreibung
1 - 2	PA für Parameters SO für Select-Options
3	_
4 - 8	Beschreibung

Beispiele:

pa_vkorg für Parameter

so_vkorg für Select-Option

Einige Spezielle Gestaltungsmöglichkeiten haben auch spezielle Namenskonventionen:

Position	Beschreibung
1 - 3	ZBT für Buttons ZIC für Icons ZLA für Labels
4	_
5 - 8	Beschreibung

3.5.4 Feldsymbole

Feldsymbole folgen der allgemein definierten Notation für Variablen, umschließen dabei lediglich die Variablen mit einer von der SAP geforderten spitzen Klammer.

Beispiel:

<lv_vkorg>

3.5.5 Typdefinitionen

Lokale Typdefinitionen in Programmen folgen der folgenden Namenskonvention:

Position	Beschreibung
1 - 2	TY
3	_
4 - 8	Beschreibung

Beispiel:

ty_vbak für lokalen Typ

3.6 Web-Dynpro-Entwicklung

Web-Dynpro ist ein von SAP zur Verfügung gestelltes Programmiermodell. Web-Dynpro ist in Java und in ABAP implementierbar, wobei bei FFT ausschließlich die ABAP-Variante verwendet wird.

Damit können standardisierte Benutzungsoberflächen (UIs) über einen deklarativen Ansatz generieren und so Web-Anwendungen mit einem Minimum an Aufwand realisiert werden.

Web Dynpro-Anwendungen werden mit deklarativen Programmiertechniken gebaut, die auf dem Modell des Model View Controller (MVC) basieren.

Das gesamte Coding zum Erzeugen der Benutzungsoberfläche wird dann automatisch in einem Standardlaufzeit-Framework generiert.

Dieser Absatz bildet eine Ergänzung zu den allgemeinen ABAP-Entwicklerrichtlinien. Hier werden ausschließlich Web Dynpro for ABAP spezifische Besonderheiten behandelt.

3.6.1 Namenskonventionen

Bei der Benennung von Elementen ist zu beachten, dass die Namen vom System automatisch in Großbuchstaben transformiert werden. Daher macht es keinen Sinn „CamelCase“ Benennung anzuwenden. Als Trennzeichen wird daher ein Unterstrich „_“ verwendet.

3.6.2 Komponenten

Komponenten werden nach dem Schema „ZWD_C_<name>“ benannt. „WD“ steht für Web Dynpro und das „C“ bezeichnet, dass es sich hierbei um eine Komponente handelt.

3.6.3 Applikationen

Applikationen bekommen fast den gleichen Namen wie die entsprechende Komponente. Allerdings wird das Kennzeichen „C“ durch ein „A“ (Application) ersetzt.

3.6.3.1 Views

Views bekommen den Präfix „VI_“. Gibt es nur einen Einstiegsview, soll dieser immer „VI_MAIN“ heißen. Dadurch wird die Verständlichkeit des Programms deutlich erhöht.

3.6.3.2 Windows

Windows beginnen immer mit „WI_“.

3.6.3.3 Plugs

Inbound Plugs beginnen mit „IP_“. Zur besseren Lesbarkeit sollten Inbound Plugs nach dem Präfix mit FROM_<view> fortgeführt werden. <view> sollte dabei den Ausgangsview möglichst gut beschreiben. Es muss nicht der vollständige Name sein.

Outbound Plugs beginnen mit „OP_“. Zur besseren Lesbarkeit sollte nach dem Präfix TO_<view> folgen. Wie beim Inboundplug auch, sollte <view> lediglich den View logisch beschreiben.

3.6.3.4 Methoden

Beim Anlegen von Methoden im Web Dynpro-Bereich gelten zunächst einmal die allgemeinen Bestimmungen aus den Entwicklerrichtlinien (z.B. bei der Parameterbenennung). Zusätzliche Richtlinien finden sich hier.

3.6.3.4.1 Controllermethoden

Im Controller befinden sich alle Methoden, die eine Schnittstelle zum Datenmodell darstellen. Methoden die zur Datenbeschaffung dienen, werden niemals in einer View implementiert.

Bei komplexerer Anfragelogik können die Controllermethoden wiederum auf Methoden der Assistance-Klasse zurückgreifen.

Methoden, die dazu dienen den Kontext zu befüllen, sollen mit einem vorangestellten „FILL_“ gekennzeichnet werden.

Methoden, die Attribute im Controller setzen, wird ein „SET_“ vorangestellt. Auslesende Methoden sollen mit „GET_“ gekennzeichnet sein.

3.6.3.4.2 Viewmethoden

Es sollte ein sprechender Name gewählt werden, der die Funktion der Methode gut umschreibt.

3.6.3.4.3 Ereignismethoden

Wird ein Ereignisbehandler angelegt, muss nur die Aktion beschrieben werden (z.B. Save). Das Framework stellt beim Anlegen der Methode automatisch ein „ONACTION“ voran.

3.6.3.5 Konfiguration

Anwendungskonfigurationen bekommen immer den gleichen Namen wie die Anwendung zu denen sie gehören. Sollte mehr als eine Konfiguration nötig sein, so kann ihr ein Suffix angehängt werden (Z.B.: ZWD_A_TEST_CUSTOMER und ZWD_A_TEST_EMPLOYEE). Wird eine Konfiguration zu einer bestehenden SAP Standardanwendung angelegt, so wird ihr einfach ein Z (bzw. Y) vorangestellt. Gleiches gilt auch für Komponentenkonfigurationen.

UI-Elemente

Element	Präfix
Button	BTN_
ButtonRow	BTR_
Caption	CPT_
CheckBox	CBX_
CheckBoxGroup	CBG_
DropDownByIndex	DDI_
DropDownByKey	DDK_
FormattedTextEdit	FTE_
FormattedTextView	FTV_
Group	GRP_
HorizontalGutter	HGT_
Image	IMG_
InputField	INP_
Label	LBL_
LinkToAction	LTA_
LinkToURL	LTU_
MessageArea	MSG_

RadioButton	RBT_
TabStrip	TBS_
Tab	TAB_
Table	TBL_
TextEdit	TED_
TextView	TEV_
Toggle Button	TBT_
TransparentContainer	TCN_
Tray	TRY_
Tree	TRE_
ViewContainerUIElement	VCN_

3.6.3.6 Erweiterungen

Es sollte wenn möglich immer nur eine Erweiterungsimplementierung pro Komponente geben. Die Erweiterungsimplementierung bekommt den Gleichen Namen wie die entsprechende Komponente und ein vorangestelltes Z (oder Y).

Werden UI-Elemente erweitert (also kopiert, ausgeblendet und per Erweiterung wieder eingeblendet) so bekommt das neue Element ein „_ENH“ als Suffix.

3.6.3.7 Assistance-Klassen

Assistance-Klassen unterliegen der Namensgebung:

„ ZCL_WD_<componentname>[_<Zusatz>]“

Ansonsten gelten hier die Namenskonventionen für ABAP-Objects.

3.6.3.8 Variablennamen

Es gelten die Standard Namenskonventionen aus den Entwicklungsrichtlinien. Zudem gelten noch folgende Regeln:

Kontextknoten: „lo_nd_<node>“

Kontextattribute: „lo_el_<node>“

3.7 Fiori-Entwicklung

Für die Entwicklung einer Fiori-App wird ausschließlich die SAP-WebIDE verwendet.

Namespace Fiori-Applikationen: com.fft.<Produkt>

3.7.1 Variablen

Es wird die Ungarische Notation verwendet.

3.7.2 Component

Sofern möglich, sollten die Basiseinstellungen in der mainfest.json erledigt werden.

3.7.2.1 Component name Fiori-Like app

Die Komponenten werden entsprechend des Namensraums und des App-Namens sprechend benannt.

z.B. **com.fft.zsd.inquery**.Component

3.7.2.2 Component name Extension

Erweiterungsprojekte sollten wie die original SAP-Fiori-Applikation benannt werden mit dem Zusatz "Extension" oder "Ext" falls der Name zu lange wird (entspricht dem Vorschlag der WebIDE).

z.B. “hcm.approve.leaverequest.HCM_LRQ_APV**Extension“**

3.7.3 Project Structure

Die Ordnerstruktur der WebIDE wird als Projektstruktur übernommen. Hilfsklassen können in einem Unterordner "util" erstellt werden.

3.7.4 Best Practise

UI Entwicklung

- Behalten Sie die Konfigurationsparameter des Frontends immer an einem Ort: einem Modell oder einer Json-Datei. Es sollte einen einzigen Ort der Wahrheit geben.
- Mischen Sie nicht verschiedene Konfigurationsstile, wie fest programmierte Werte mit logischen Funktionen und ein Modell irgendwo im Code.
- Verwenden Sie nirgendwo fest programmierte Werte; setzen Sie keine Logik in die View ein.
- Vertauschen Sie die Konfigurationen nicht. Besser: An dem Punkt, an dem ein ConfigModel definiert oder gefüllt wird, sollte die gesamte Logik geschrieben werden. Oder wenn es Gründe gibt, warum dies nicht möglich ist, stellen Sie es in ein ViewModel.
- Verwenden Sie die Konfiguration (json / model, whatever) in einem ViewModel (legen Sie die Logik dort ab) und binden Sie das ViewModel an die View.
- Jedes Mal, wenn Sie sich mit einer ID eines Elements erwischen, fragen Sie sich, ob es nicht besser ist, eine ViewModel- und/oder Modellbindung zu verwenden.
- Wo immer es möglich ist, verwenden Sie ein ViewModel und keine IDs!

- Legen Sie Logik in den Controller, der das ViewModel steuert. Setzen Sie keine Logik in eine View ein.
- Verwenden Sie keine IDs, um das Verhalten eines Controls zu steuern, binden Sie ein ViewModel daran.
- Verwenden Sie eine Modellbindung, wo immer es möglich ist. Speichern Sie die Daten nicht in der Value-Eigenschaft eines Controls und greifen Sie sie über die ID eines Controls.
- Verwenden Sie ein bestehendes Modell oder erstellen Sie ein Json-Modell (in model.js) und binden Sie es an das Feld, um mit dem Wert use the model property zu arbeiten. Achten Sie darauf, dass Sie die richtige Bindung OneWay / TwoWay verwenden und das Modell nach Gebrauch freimachen.
- Hinzufügen von Modellen zur Komponente, nicht zu sap.ui.getCore.....
- Es wird empfohlen, mit benannten Modellen zu arbeiten.
- Jede Applikation sollte einen MessageManager enthalten.

oData

- Halten Sie Ihre OData-Struktur so flach wie möglich.
- Wenn Sie bei jedem Aufruf einen Parameter benötigen, z.B. PersNr, dann fügt man einen Header Parameter hinzu, anstatt jede OData-Einheit mit diesem Parameter zu erweitern.
- Sortier-, Gruppen-, Auswahl- und Paging-Optionen sollten implementiert werden.
- Implementierung der Inline-Zählung ist von Vorteil.
- Nutzen Sie die Deep-Create-Funktionen.
- Auslösen von Ausnahmen mit eindeutigen Fehlern.
- Arbeiten mit Batch-Anforderungen.
- Verwendung von sy-uname und Sprachparametern.
- Optimieren Sie Ihre Auswahlanweisungen (z.B. für einen einzelnen Zählbefehl).

3.7.5 Namenskonventionen oDATA, Pakete

Die Namen beginnen immer mit **Z** (Kundennamensraum) gefolgt von dem Modulnamen und einem Unterstrich.

Bsp.: ZSD_...

3.7.6 RFC-Connection

<SYSTEMALIAS>_RFC bzw. <SYSTEMALIAS>_HTTPS

3.7.7 Kataloge

Hier wird zwischen Business- und Target-Catalog unterschieden, damit in den Berechtigungsrollen unterschieden werden kann welche Kacheln für den einzelnen Anwender sichbar sind.

Durch diesen "Trick" werden die Kacheln auch unter einer Gruppe im Launchpad angezeigt.

Business Catalog "BC" werden ohne Zielzuordnung, aber mit ALLEN Kacheln angelegt

Target Catalog "TC" werden ohne Kacheln angelegt und erhalten nur die Zielzuordnung für die jeweilige App

Z<Modul>_<[TC],[BC]>_.....

Bsp.:

ZSD_TC_INQ_Vertriebsmitarbeiter_OWN

ZSD_BC_INQ_Vertriebsmitarbeiter

3.7.8 Gruppe

Z<Modul>_<BG>_.....

bsp.: ZSD_BG_Vertriebsmitarbeiter

3.7.9 Übersetzungen

i18n-Dateien sollten nur anwendungsspezifische Übersetzungen enthalten. Sofern möglich sollten die Bezeichnungen und deren Übersetzungen über die Metadaten des oDATA-Services vom entsprechenden Backend gezogen werden um eine einheitlich Übersetzungen zwischen den Systemen sicherzustellen.

3.7.10 Dokumentation

Es ist JSDocs zu verwenden.

Kommentare müssen vor jeder Datei /Klasse und vor jeder Funktion verwendet werden, inline nur dann, wenn schwierige Codierungen durchgeführt werden, die man mit einem schnellen Lesen nicht verstehen kann.

Verwenden Sie die folgende Kommentarnotation

/**

*

**/

Inline verwenden Sie die Notation //.

3.8 Spezielle S/4HANA-Entwicklungsthemen

3.8.1 Virtuelles Datenmodell (VDM)

Ein Datenmodell stellt den Rahmen dar, welche Beziehungen in einer Datenbank sind. Datenmodelle sind das Fundament der Softwareentwicklung. Sie bieten eine standardisierte

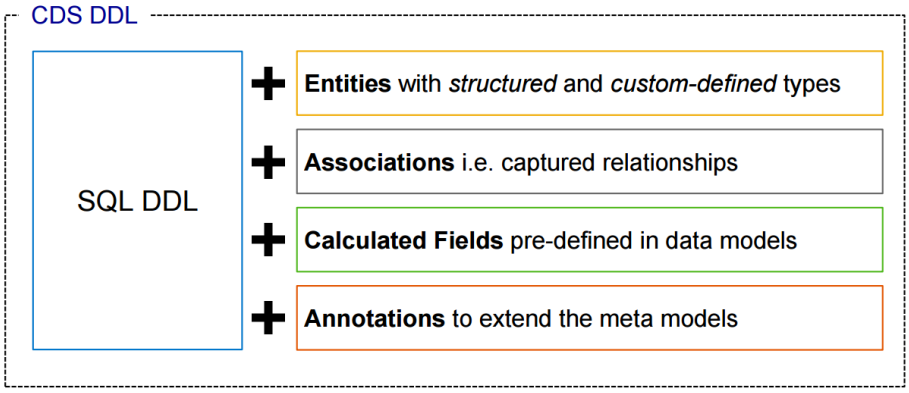
Methode zur Definition und Formatierung von Datenbankinhalten konsequent über Systeme hinweg, so dass verschiedene Anwendungen dieselben Daten gemeinsam nutzen können.

S/4HANA unterstützt ein neues Datenmodell, indem es einen gebrauchsfertigen Inhalt mit CDS-Views bereitstellt.

Mit der Verfügbarkeit der SAP HANA-Plattform ist ein Paradigmenwechsel in der Art und Weise, wie Geschäftsanwendungen bei SAP entwickelt werden.

Die Faustregel ist: Tun Sie so viel wie Sie können in der Datenbank, um die beste Leistung zu bekommen.

Um SAP HANA für die Anwendungsentwicklung nutzen zu können, hat SAP eine neue Datenmodellierungsinfrastruktur eingeführt, die als Kerndatendienste be-kannt ist. Mit CDS werden Datenmodelle auf der Datenbank und nicht auf dem Applikationsserver definiert und konsumiert. CDS bietet auch Möglichkeiten über die traditionellen Datenmodellierungswerkzeuge hinaus, einschließlich der Unter-stützung von konzeptionellen Modellierungs- und Beziehungsdefinitionen, inte-grierten Funktionen und Erweiterungen.



- Virtuelle Datenmodelle befinden sich in Paketen nach Komponenten geordnet
  - Hauptpaket APPL

3.8.2 VDM Namenskonventionen

Präfix	Beschreibung
ZI_	Public Views (Interfaces) • direkte Nutzung in Custom Views • Beispiel: I_SalesArea
ZP_	Private Views • nicht direkt zu nutzen, nur in Custom Views • Beisiel: P_Sales_Office_ValueHelp
ZC_	Consumption Views • direkt in Reporting Anwendungen komsumierbar • Beispiel: C_SalesOrder_
*ValueHelp	• Wertehilfe • Beisiel: P_Sales_Office_ValueHelp
*QRY oder *Q	• Ready to use analytische Query • OData Service Ready

3.8.3 CDS Namenskonventionen

Für die Definition der CDS Views sollte sich syntaktisch am SAP Modell orientiertieren. D.h.

1. Definition der View Namen, wie in Kapiel VDM Namenskonventionen beschrieben.
2. Parameterwerte mit P_ kennzeichnen
3. Sprechende Namen in der Projektionsliste:
    1. VBELN → SalesDocument
    2. POSNR → SalesDocumentItem
4. CDS-Extensions
    1. Bei Erweiterung von Standardviews wird Z + <Name des SAP-View (ggf. gekürzt)> + _E verwendet
    2. Eigene Felder beginnen analog den Tabellen-Appends mit ZZ_<FELDNAME>

3.8.4 HANA Konventionen ABAP Entwicklung

SAP HANA ist ein von SAP zur Verfügung gestelltes Programmiermodell für die Entwicklung im HANA Umfeld, mit Erweiterungen im ABAP Umfeld.

Dieser Absatz bildet eine Ergänzung zu den allgemeinen ABAP-Entwicklerrichtlinien. Hier werden ausschließlich SAP HANA for ABAP spezifische Besonderheiten behandelt.

3.8.4.1 Proxy Objekte

Proxy Objekt dienen die Bereitstellung von HANA Objekten im ABAP Context. Es können mittels Externem View (Dictionary View) HANA Views (Attribut, Analytic, Calculation View) im ABAP Context verfügbar gemacht werden.

Mit dem DB Proxy können Stored Procedures (SQL Script) im ABAP zur Verfügung gestellt werden.

Im folgenden werden die Namenskonventionen für die Objekte, sowie Ein- / Ausgabe Parameter genannt.

ABAP Zugriff	Präfix	Beispiel
Externer View	ZEV_	ZEV_VBAK
Database Procedure Proxy	ZDP_	ZDP_EXEC_DATA_CALC
Interface zum DP Proxy	ZIF_	ZIF_EXEC_DATA_CALC

3.8.4.1.1 CDS Documents

Element	Beschreibung
NameSpace Deklaration	Name Space Deklaration muss die erste Anweisung sein und representiert den absoluten Package Pfad im Repository. Daher gelten die Konventionen auch für die Package Definition.
Schema Definition	Das Schema gibt den Ort an in welchem die definierten Objekte gespeichert werden.
CDS Artifact Definition	Modelelemente wie Contexte,Entitäten,Typen und Views.

Beispiele:
fft.com.hana.cds.<Anwendungskomponente>.data
fft.com.hana.cds.sd.data

3.8.4.1.2 CDS Naming

CDS Dokumente werden mit der Dateiendung .hdbdd angelegt -> MyContext.hdbdd

Folgende Zeichen sind für die Definition von CDS Documenten und packages erlaubt:

- Groß- / Kleinbuchstaben (aA-zZ)
- Unterstrich (_)
- Zahlen (0-9)

Verboten sind:

- Bindestrich (-)
- Punkt (.)
- Zahlen als erstes Zeichen
- nur Zahlen

4 Weitere Dokumente

- SAP Leitfaden | DSAG - Deutschsprachige SAP ® Anwendergruppe e.V.
- dsag_handlungsempfehlung_abap_2016_0.pdf