TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: B2612 – Elektrotechnika a informatika
Studijní obor: 2612R011 – Elektronické informační a řídicí systémy

Srovnávací studie nasazení systému SAP

Using SAP Comparative study

Bakalářská práce

Autor: **Jiří Danyi**
Vedoucí práce: Ing. Martin Vlasák

V Markvarticích 10. 12. 2008

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

Katedra Akademický rok: 2007/2008

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jméno a příjmení:

studijní program:

obor:

Vedoucí katedry Vám ve smyslu zákona o vysokých školách č.111/1998 Sb. určuje tuto bakalářskou práci:

Název tématu:

Zásady pro vypracování:

1.

2.

3.

4.

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

10.12.2008

Jiří Danyi

Poděkování

Rád bych touto formou poděkoval Ing. Martinu Vlasákovi za odborné vedení, poskytnuté cenné rady a zkušenosti, především však svým rodičům, bez jejichž podpory při studiu by tato práce nikdy nevznikla.

Anotace

Úlohou této práce bylo porovnání verzí systému SAP z pohledu programátora. Jsou zde popsány nejdůležitější novinky které přináší verze ERP 2005 oproti verzi 4.6C.

V úvodních kapitolách lze nalézt podstatu pojmu ERP jakožto komplexní informační systémy organizací, zastřešující činnosti související s výrobou, financemi, účetnictvím a dodavatelskými řetězci. V příloze je program, využívající nové funkcionality. Jsou tam použity regulární výrazy pro hledání programů obsahující zadané texty v celém systému SAP podle vybraných parametrů.

Annotation

Confrontation od SAP version from programmers point of view was a main aim for this task. There is a description of the most important changes of version ERP2005 in comparisons with 4.6.C.

In the opening chapters can be found explannation of the ERP systeme as a comprehensive information systems of organization inclusive of production, finance, accounting and supply chain. Appendix contains a program taking an advantage of new funcions. There are regular expressions used to search for programs containing specified text in the SAP system by selected parameters.

Obsah

Seznam zkratek

ERP	Enterprise Resource Planning
SAP	Systems Applications and Products
ABAP	Advenced Business Application Programming
IS	Information System
OOP	Object Oriented Programming
RTTI	Run Time Type Identification
RTTC	Run Time Type Creation
HR	Human Resources
IT	Information Technology
ALV	Abap List Viewer

Úvod

Dříve byli ERP systémy brány jistou dávkou nedůvěry. Postupně se stávali po implementaci ve firmách významnou konkurenční výhodou a nyní jsou pro velké firmy nezbytné. Prakticky každá velká společnost dnes využívá nějaké ERP řešení. Nyní se ERP řešení přesouvá i do segmentu malých a středních společností.

ERP systém se nemůže ve firmě pouze nainstalovat jako jiný software, ale musí se přizpůsobit firemním potřebám. Výrobní procesy a technologie jsou pro každou firmu specifické a proto nemůže existovat jedno univerzální nastavení. Každá společnost se také postupně vyvíjí a optimalizuje provoz. ERP systém se proto musí stále vyvíjet a upravovat také. O to se starají vývojové týmy lidí ať už uvnitř společnosti, nebo externě najaté firmy. Do těchto týmů patří také programátoři využívající jazyk ABAP.

Systém SAP umožňuje jakýkoli vývoj zákaznických řešení při dodržování určitých pravidel jako je jmenná konvence atd. Systém SAP se průběžně vyvíjí a nabízí nové možnosti řešení problémů. Přizpůsobuje se také novým informačním technologiím (např. webové rozhraní a internetové obchody).

Nevyvíjí se pouze různé moduly řešení, ale také nástroje pro vývoj. Práce na zákaznickém vývoji se tím zlepšuje. Společnost SAP se snaží držet krok s novými technologiemi v oblasti software. Proto je patrný vývoj nejen v možnostech jazyka ABAP, ale také v transakcích sloužících k vývoji (např. nový editor).

Veliký posun udělala společnost SAP mezi starší verzí 4.6C a verzí 7.0 značenou jako ERP 2005. Tato práce se zabývá popisem nejvýznamnějších změn a ukazuje je v praktickém použití.

1 ERP systémy

1.1 Úvod do problematiky

Hlavními vlastnostmi ERP jsou schopnost automatizovat a integrovat základní podnikové procesy, sdílet společná data a zpracovávat je v rámci celého podniku, vytvářet a zpřístupňovat informace v reálném čase. Bývá chybou, že podnikový informační systém je zužován na pojem ERP nebo se s ním kryje. Podstata systému ERP nespočívá v jeho zkratce, k plánování má daleko, zapomíná na zdroje, odkládá termíny. Předmětem ERP jsou jednotlivé podnikové procesy. ERP se snaží sloučit různé oblasti činností a funkcí napříč celým podnikem (organizací nebo institucí) až k jednotlivým programovým úlohám sloužícím různým potřebám organizačních složek podniku. Všechny útvary mají svůj vlastní systém práce, užívají optimalizovaný program pro svoji specifickou činnost. ERP spojuje jejich práci dohromady, integrovaný software pracuje s databázemi tak, že různá oddělení či resorty snadněji sdílí informace a navzájem komunikují.

Příklad typického podnikového procesu je zpracování objednávky. Před zavedením ERP obíhala objednávka z poštovní přihrádky jednoho útvaru do dalšího přes celý podnik. Zpoždění narůstalo, vznikaly chyby při zadávání údajů z objednávky do jednotlivých programů, v extrémním případě se objednávka ztratila. Nikdo v podniku nevěděl, v jakém stavu právě objednávka je, např. ve finančním útvaru netušili, zda je objednaný materiál na skladě. ERP překonává osamocené systémy ve financích, výrobě, skladu či HR a nahrazuje je integrovaným software rozděleným do modulů, které se zhruba přibližují původním samostatným systémům. Ve financích, výrobě a skladu mají tak všichni uživatelé "svůj" vlastní software, jenž je navíc spojuje. Lidé ve financích mohou zjistit, je-li položka objednávky ve skladu. Po zadání objednávky ERP připraví algoritmus postupných kroků k jejímu splnění, uživatel systému má všechny informace nutné pro zpracování objednávky (platební disciplinu a historii objednávek z finančního modulu, soupis zboží ve skladu, připraví objednávku materiálu v logistickém modulu). Pracovníci z různých oddělení mají k dispozici stejné informace, které mohou aktualizovat. Po ukončení práce s objednávkou v jednom úseku je objednávka automatizovaně přesměrována do dalšího útvaru. Pro zjištění stavu transakce je nutné pouze vyhledat záznam v ERP systému a sledovat ho. Proces zpracování objednávky

ilustruje význam ERP související s organizací řízení podniku. ERP pracuje obdobně s ostatními i složitějšími firemními procesy ve všech oblastech činností podniku.

Takto se jeví ERP naprosto ideální, realita je však mnohem složitější. Dá se to ukázat u výše uvedeného příkladu s plnými přihrádkami s objednávkami. Proces jejich zpracování nemohl být pohotovější, ale byl jednoduchý. Účetní dělali svoji práci, ve skladě dělali své, a pokud něco zabloudilo mimo oddělení, byl to problém někoho jiného.

Se zavedením ERP se nezmění organizace práce jenom tím, že uživatel systému zadá další údaje do počítače a stiskne kouzelnou klávesu. Najednou z obrazovky bliká platební morálka zákazníka z útvaru financí, ze skladu je vidět stav zboží. Zaplatí zákazník včas? Bude se schopen objednat včas materiál? Tato rozhodnutí nemusel dříve dělat pracovník zákaznického servisu (dříve odbytu), navíc se jeho reakce projeví v útvaru skladu, kde dříve měli lidé stavy skladu v hlavě nebo na svém počítači. Vzájemná informovanost, schopnost komunikace a odpovědnost za podnik nikdy před zavedením ERP nebyla na takové úrovni.

Lidé nemají rádi změny, ale ERP vyžaduje změny ve způsobu jejich práce, v úrovni řízení organizace. A to je pravá hodnota ERP. Software není tak důležitý jako změna způsobu organizace řízení podniku. Přispěje-li ERP ke zlepšení způsobu práce uživatelů, zlepšení organizace výroby, efektivnímu nákupu a k efektivnímu využití účetních informací, pak bude teprve měřitelná hodnota software. Je-li jednoduše instalován nějaký modul ERP bez změny pracovních postupů, nebude hodnota software žádná. Nový systém pak naopak může zpomalit jednoduchý starý program, zvlášť užívá-li jen málokdo jeho nové funkční možnosti.

Důraz na změnu způsobu práce si můžeme ilustrovat na výše uvedeném procesu zpracování objednávky. Tento proces se skládá z řady operací nebo činností, které provádí obecně několik různých pracovníků a každý z nich používá programové aplikace s odlišnými funkcemi. Funkční moduly pro jednotlivé operace mohou být např.: evidence objednávek, evidence odběratelů, vydané faktury, výdejky ze skladu, účtování faktur, evidence pohledávek, účtování výdeje ze skladu, evidence skladu, sledování stavu skladu, požadavek pro výrobu, bankovní výpisy, párování plateb s fakturami. Každý z funkčních modulů může pracovat se svou vlastní databází, data lze přepisovat z jednoho souboru do druhého, příp. některé moduly mohou užívat stejnou

databázi. Síťovým propojením počítačů z různých podnikových útvarů již vznikne určitý integrovaný software a uživatelé mohou spustit jednotlivé programy na svém počítači. Pouhým integrováním jednotlivých modulů do jednoho systému bez změny obsahu práce lidí se však stalo pouze to, že lidé používají počítače. Cíle ERP jsou však jiné, musí být v podniku tolik oddělení a lidí pracujících se stejnými informacemi? Mnoho činností (nebo programů) lze spojit, automatizovat, databáze propojit, záznamy databází uchovávat pro další využití. I jednoduchý proces zpracování objednávky může vyvolat řadu až fantastických představ a námětů řešení. Objednávka, zakázka nebo obchodní případ tak postupuje celým ERP řešením, přirozené procesy nejsou rozdělovány do nějakých sestav pro jednotlivé pracovníky, ale informace jsou všemi sdíleny.

Podnikový IS je výsledkem integrace mnoha systémů a podsystémů a při pohledu na ERP systém nemůžeme vynechat vzájemné souvislosti. Základní podsystémy ERP jsou závislé na typu podniku, IS výrobního podniku a IS obchodní organizace obsahují jak specifické podsystémy, tak i podsystémy funkčně obdobné. Základem ERP řešení je však jejich vzájemná integrace. IS/IT v podniku by měl být v souladu s podnikovými cíli a metodami řízení podniku, nelze je chápat odděleně. Podnik a jeho IS je ve vztahu reálného systému a jeho modelu, čím dokonalejší technologie a znalosti se používají, tím je vztah IS a podniku samotného užší. Existují i trendy vytvářet a podřizovat podnik podle IS. Systém obecně je soubor objektů a jejich vzájemných vztahů, který v souhrnu vykazuje určité chování popsané např. funkcemi systému.

Procesy jsou posloupnosti změn stavů systému. Jakýkoliv IS je otevřený, je součástí systému nebo je obklopen dalšími systémy, v nichž probíhá výměna informací a zpráv. Každý systém je také dynamický nejen ve smyslu vzájemné komunikace mezi objekty a systémy, ale i ve smyslu vzájemného ovlivňování struktury systému a jeho vývoje. Tyto věty patří jistě do úvodu k formální teorii systémů. Při pohledu na IS se však musejí mít na zřeteli tyto obecné vlastnosti a nesoustředit se jen na některé aspekty chování, funkcí nebo struktury, ale mělo by se nahlížet na IS jako na celek, který je součástí vyššího hierarchického systému, má však své typy uživatelů a očekávají se od něho určité specifické funkce.

Architektura IS podniku bývá znázorňována jako pyramida složená z několika vrstev podle úrovně řízení organizace. V návaznosti na hierarchii systému řízení

podniku (jakékoliv instituce) vyplývá použití různých částí IS na jednotlivých úrovních řízení od transakčních systémů pro operativní řízení přes systém pro podporu řízení, pro podporu rozhodování až k úrovni systémů pro strategické vrcholové řízení. Stavba architektury má několik pohledů a dimenzí, minimálně funkční, procesní, datovou. ERP systém tvoří v této stavbě její základní kameny, které sice nejsou vidět, ale o to důležitější význam má ERP při budování, provozu a dalším vývoji podnikového IS. Informační strategie podniku, ať už orientována produktově, odbytově nebo zákaznicky, musí stát na pevných základech.

Každý podnik je jedinečný a specifický stejně jako jeho IS. Systém netvoří jednotlivosti, systém je druh celku, kde je každé části přiděleno své místo a funkce.

1.2 Významní výrobci ERP

- SAP
- Lawson
- Oracle Applications
- The Sage Group
- Microsoft Dynamics
- SSA Global Technologies

2 SAP

Během tří desetiletí se firma SAP vyvinula z malého regionálního podniku na špičku mezi mezinárodní společnosti. Dnes je SAP lídrem na světovém trhu v rámci ERP systémů. Společnost nyní zaměstnává více než 50 000 lidí.

2.1 Historie

2.1.1 70tá léta - Vize

V roce 1972 byla společnost založena pěti bývalými zaměstnanci firmy IBM - Dietmar Hopp, Hans-Werner Hector, Hasso Plattner, Klaus Tschira a Claus Wellenreuther – pod jménem Systems Applications and Products. Jejich vizí bylo vytvořit standardní aplikační software pro obchodní zpracování.

O rok později byl vytvořen první účetní software, který tvořil základ pro další vývoj. Později byl znám jako „systém R/1“. „R“ znamená zpracování v reálném čase.

Na konci tohoto desetiletí se díky intenzivnímu zkoumání IBM databází a systému kontroly dialogů vyvíjí SAP R/2.

2.1.2 80tá léta - Rychlý růst

Společnost SAP se přesouvá do své první budovy v průmyslové části města Walldorf nedaleko Heidelbergu. Oblast vývoje softwaru a jeho 50 terminálu je nyní pod jednou střechou. Padesát ze sta největších německých průmyslových firem je nyní zákazníky SAPu.

Systém SAP R/2 dosahuje vysoké stability programů. Je třeba mít na paměti jeho nadnárodní zákazníky, zvládnutí různých jazyků a měn. V těchto a dalších inovacích systému SAP R/2 společnost SAP vidí rychlý růst.

Do poloviny tohoto desetiletí SAP zakládá svou první obchodní organizaci mimo území Německa, v Rakousku. Společnost taká poprvé vystupuje na veletrhu CeBIT v Německém Hanoveru. Tržby dosahují 100 milionů DM (přibližně $ 52 mil. EUR) dříve než se očekávalo.

V srpnu 1988 se SAP stává akciovou společností. Začátkem listopadu uvádí na Frankfurt a Stuttgart burzách cenných papírů 1,2 milionů akcií.

Německý renomovaný obchodní časopis oceňuje SAP jako firmu roku. Toto ocenění firma získává ještě dvakrát během několika následujících let.

Firma SAP zakládá dceřiné společností v Dánsku, Švédsku, Itálii a ve Spojených státech. Tato mezinárodní expanze znamená obrovský krok dopředu.

2.1.3 90tá léta - nový přístup k softwaru a řešení

Uvolňuje se na trh SAP R/3. Jedná se o koncept klient-server. Jednotný vzhled grafického rozhraní, důsledné používání relačních databází a schopnost běhu na počítačích od různých dodavatelů se setkává s obrovským ohlasem. S příchodem verze SAP R/3 se přechází na novou generaci podnikového softwaru. Od sálové výpočetní techniky na tří generační architekturu - databázi, aplikační a uživatelské rozhraní. Dodnes je klient-server architektura je běžně používaná v obchodním softwaru.

Zřizuje se stále více dceřiných společností mimo Walldorf. Otevírá se nové vývojové a obchodní centrum ve Walldorfu. Toto symbolizuje celosvětový úspěch společnosti. Poprvé SAP získává podílem na trhu mimo Německo více než 50 procent z celkových tržeb.

V roce 1996 společnost získává 1089 nových zákazníků systému SAP R/3. Na konci roku je instalováno více než 9000 systémů po celém světě.

SAP slaví pětadvacáté výročí v roce 1997 a zaměstnává přibližně 12.900 lidí. Pokračuje se v posílení a rozšíření nových specifických průmyslových řešení.

Vzhledem k tomu, že se desetiletí chýlí ke svému konci, generální ředitel a spoluzakladatel Hasso Platner, představuje strategii mySAP.com. Oznamuje se začátek nového směru společnosti zaměřeného na webové technologie.

2.1.4 Inovace pro nové tisíciletí

Vzhledem k internetu se společnost zaměřuje na webové aplikace. SAP vyvíjí SAP Workplace a dláždí tím cestu pro ideu o podnikovém portálu a specifickým přístupům k informacím.

V současné době je více než 12 milionů uživatelů pracujících každý den se SAP řešením. K dispozici je nyní 121.000 instalací na celém světě, více než 1500 partnerů SAP, více než 25 specifických business řešeních a více než 75.000 zákazníků ve 120 zemích. SAP je celosvětově třetí největší nezávislý prodejce softwaru.

2.2 Moduly

- FI (Financial Accounting) Finanční účetnictví
- CO (Controlling) Kontroling
- AM (Asset Management) Evidence majetku
- PS (Project system) Plánování dlouhodobých projektů
- WF (Workflow) Řízení oběhu dokumentů
- IS (Industry Solutions) Specifická řešení různých odvětví
- HR (Human Resources) Řízení lidských zdrojů
- PM (Plant Maintenance) Údržba
- MM (Materials Management) Skladové hospodářství a logistika
- QM (Quality Management) Management kvality
- PP (Production Planning) Plánování výroby
- SD (Sales and Distribution) Podpora prodeje

2.3 Verze SAPu

Vývoj systému SAP od verze 4.6 do ERP 2005 je rozdělen do dvou částí. Zvlášť se vyvíjí báze a zvlášť jednotlivé moduly.

Báze	Moduly	Marketingové označení
4.6	4.6	release 4.6
4.10	4.6	release 4.6
6.20 WAS	4.7	enterprice release
6.40 WAS	ECC 5.0	ERP 2004
7.0 SAP NW	ECC 6.0	ERP 2005

Tab. 1 : Vývoj systému SAP

3 ABAP

ABAP je jazyk čtvrté generace první vyvinutý v osmdesátých letech. Je to jeden z prvních jazyků zahrnující koncept logických databází, který poskytuje vysokou úroveň abstrakce na úrovni databáze.

Programovací jazyk ABAP byl původně používán pro vývoj platformy SAP R/3. Je také využíván pro rozšíření základních aplikací zákazníky nebo pro vývoj nových zákaznických řešeních. Pro plné zvládnutí jazyka ABAP je důležitá znalost relačních databází.

3.1 Rozdíly mezi SAP R/3 4.6C a SAP ERP 2005

3.1.1 Koncept balíčků

Balíky (tzv. package) nahrazují staré vývojové třídy a zachovávají jejich staré vlastnosti. Některé vlastnosti jsou nové:

- Vnoření více balíků do sebe.
- Objekty jsou viditelné jen v rámci jednoho balíku, případně z vnořených balíků. Nelze vidět objekty opačně.

3.1.2 SLIN

Vylepšená verze transakce SLIN, sloužící pro statickou kontrolu programů, funkčních modulů, tříd a interfaců. Výsledek kontroly je zobrazen ve stromové struktuře.

3.1.3 Zprávy pomocí příkazu MESSAGE

Přibyla nová možnost syntaxe pro příkaz MESSAGE. Nemusí se již používat třída message jako dříve, řetězec zprávy lze odeslat přímo z programu jako text nebo proměnná.

```
MESSAGE ´Toto je chybová hláška´ type ´E´.
```

3.1.4 Nové generické datové typy

Dříve se mohl použít pro generické datové typy pouze příkaz ´TYPE ANY´. Nyní je možností více.

- numeric (obsahuje typy i, p, f)
- xsequence (obsahuje x, xstring)
- csequence (obsahuje n, d, t)
- clike (obsahuje n, d, t, csequence)

Použitím nových generických typů se může zachovat odlišení typů od sebe, ale stále se programuje genericky.

3.1.5 Logické výrazy

Přibyla možnost volat logické operátory s negací vložením NOT.

- IS NOT INITIAL
- IS NOT BOUND
- IS NOT ASSIGNED
- IS NOT REQUESTED
- IS NOT SUPPLIED
- NOT IN
- NOT BETWEEN

3.1.6 Přímé volání aritmetických funkcí

Všechny operandy v matematických výrazech mohou byt nahrazeny přímo funkcí.

Dříve:

```
N = strlen( string_var ).
IF N eq 10.
* do something
ENDIF.
```

Nyní:

```
IF strlen( string_var ).
* do something
ENDIF
```

3.1.7 Nové matematické agregační metody

S použitím nových funkčních metod se může najít rozsah hodnot v elementárních datových typech.

```
cl_abap_exceptional_values=>get_max_value
cl_abap_exceptional_values=>get_min_value
cl_abap_math=>rounf_f_to_15_decs
```

3.1.8 Podmíněné ukončení bloku kódu

Na rozdíl od příkazů CHECK a EXIT nový příkaz RETURN okamžitě opouští celý funkční modul a program pokračuje ve zpracování dalších příkazů v programu.

Dříve:

- CHECK
- EXIT
- CONTINUE

Nyní:

- CHECK
- EXIT
- CONTINUE
- RETURN

3.1.9 ABAP editor

ABAP editor se přejmenovává na CODE editor a přináší nové možnosti. Pro nový editor musí být nainstalována novější verze SAP GUI než 6.40 patch level 9. V nastavení se může kdykoliv přepínat mezi novým a starým editorem.

Některá vylepšení:

- Zobrazené číslování řádků.
- Označení změněných řádků programu.
- Bloky programu se mohou zabalit a rozbalit pro větší přehled. Zabalené bloky se zobrazí ihned po umístění kurzoru myši na text.
- Zvýraznění syntaxe programu může být upraveno uživatelem.
- Záložky v kódu a rychlá navigace mezi nimi.
- Automatické doplňování kódu.
- Vzorové příklady kódu pro rychlé vložení.

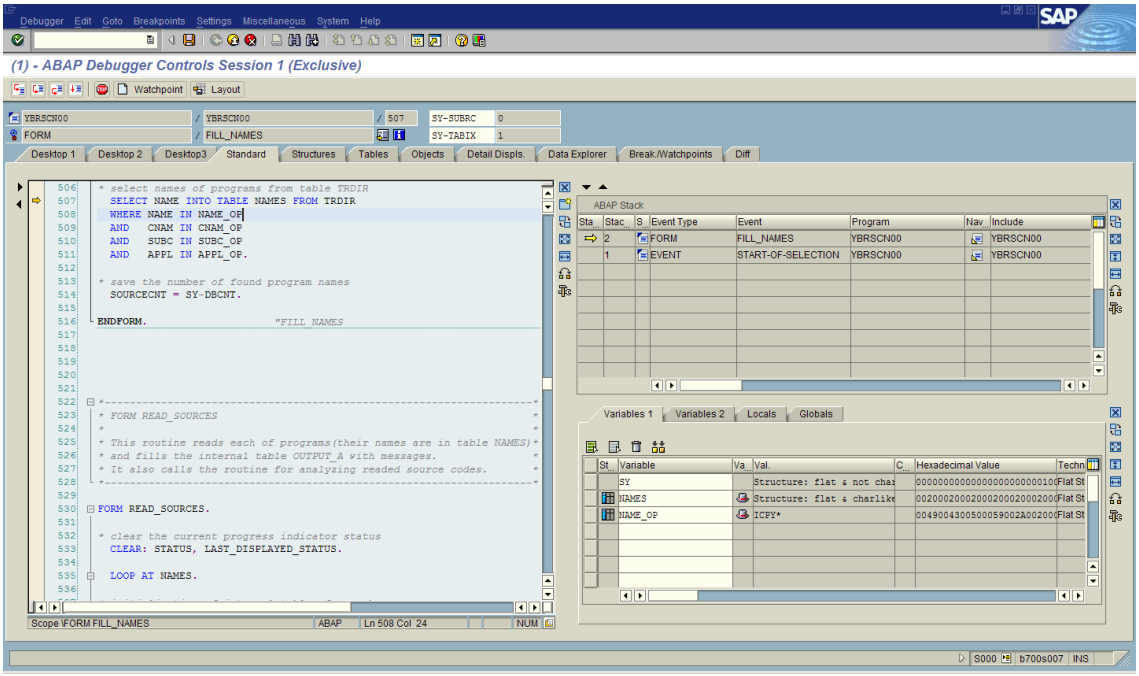
3.1.10 ABAP debuger

Ladící mód programu přináší řadu nových vylepšení.

Nejdůležitější vylepšení:

- Ladící mód se spouští v novém okně čímž šetří paměť.
- Více desktopů které jsou nyní plně konfigurovatelné uživatelem.
- Pomočí nové funkcionality GOTO se může přeskakovat na libovolný řádek kódu při ladění programu.

- Ukládání bodů přerušení běhu programu.
- Externí a interní body přerušení.
- Integrovaný nástroj prohlížení paměti.
- Porovnávací nástroj pro datové struktury.



Obr.1: Abap debuger

3.1.11 Unikód

Unikód znamená použití dvou bytů pro jeden znak. Tím se rozšířila znaková sada z 256 na 65536 znaků. To způsobuje problémy v programech při přechodu na unikód tam, kde se používá pevně stanovená délka řetězce a přesun do interních tabulek.

V programech se uvádí zda mají pracovat v BYTE nebo CHAR módu. CHAR mód je nastaven standardně a CHAR mód se používá pouze pro hexadecimální proměnné.

Následující předdefinované datové typy jsou interpretovány jako znakové: C, N, D, T a STRING. Bytové typy proměnných jsou X a XSTRING.

Nové varianty pro unikódové porovnávací operandy jsou BYTE-CO, BYTE-CA a BYTE-CP místo CO, CA a CP.

3.1.12 Nové OO transakce

Vzniká možnost vytvářet instance třídy přímo pomocí transakčního kódu.

3.1.13 Nové odchycení výjimek

Nyní je postaveno na objektově orientovaných třídách. Původní forma výjimek je zachována a může být dál používána. Informace o výjimce je dostupná pomocí předdefinovaných metod třídy výjimek (GET_SOURCE_POSITION, GET_TEXT).

Pomocí transakce SE24 je možnost definovat vlastní třídu výjimek, stavy a chybové texty. Pro vyvolání výjimky se používá příkaz RAISING a pro vymazání CLEANUP. To se používá pro stornování transakce, uvolnění zámků, mazání dočasných souborů atd.

Odchytávání výjimek se vkládá do kódu ohraničeného příkazy:

- TRY
- CATCH
- ENDTRY

```
* příklad odchycení výjimek
REPORT  YBR_EXCEPTION_EXAMPLE.

DATA: REF_EXC TYPE REF TO CX_ROOT,
      TEXT TYPE STRING.
TRY.

    PERFORM CALCULATION.

* odchytnutí výjimky
  CATCH CX_ROOT INTO REF_EXC.
    TEXT = REF_EXC->GET_TEXT( ).
    MESSAGE TEXT TYPE 'I'.
ENDTRY.

* volaná subroutina
```

```
FORM CALCULATION RAISING CX_SY_ZERODIVIDE.
  DATA: NUM TYPE F.
* operátor vyvolávající výjimku
  NUM = 10 / 0.
* výjimka není vyvolána zde, ale o úroveň výš
ENDFORM.
```

3.1.14 OOP

Je nová možnost jak volat metody tříd. Už není nutné používat příkaz CALL METHOD jako dříve. Syntaxe je nyní podobna jazyku C++.

```
objekt->metoda( ).
objekt=>metoda( ).
```

3.1.15 RTTI a dynamické programování

Nová verze přichází s řadou vylepšení týkající se těchto moderních programovacích možnostech.

ABAP prostředí nyní poskytuje speciální set standardních tříd, které mohou být použity pro identifikaci instancí datových typů během běhu programu. RTTI - je zkratka pro „Run Time Type Identification“). Tato funkce je velmi užitečná v dynamických konstrukcích složitějších obchodních případech.

RTTI používá třídy CL_ABAP_TYPEDESCR, CL_ABAP_OBJECTDESCR, CL_ABAP_DATADESCR a třídy z nich zděděné.

Nový ABAP má také techniky pro dynamickou tvorbu datových typů (RTTC – je zkratka pro „Run Time Type Creation“). Díky tomu nyní můžeme:

- Vytvářet dynamické datové typy.
- Vytvářet dynamické instance tříd.
- Vytvářet odkaz na datový dynamický typ (reference).
- Dereference odkazu (získání instance od reference).
- Dynamické volání metody.
- Používání up-castu a down-castu pomocí polymorfismu.

```
REPORT  YBR_DYNPROG.
* vytvoření dynamické struktury založené na DDIC
* a vytváření dynamických interních tabulek založených
* na dynamických strukturách

* field symbol pro strukturu
FIELD-SYMBOLS: <FS_STRUKTURA> TYPE ANY.
* reference na dynamickou strukturu
DATA: R_STRUCT TYPE REF TO DATA.
* reference na popis instance pro strukturu
DATA: R_STRUCT_DESCR TYPE REF TO CL_ABAP_STRUCTDESCR.

* získání sinstance struktury SPFLI
R_STRUCT_DESCR ?= CL_ABAP_TYPEDESCR=>DESCRIBE_BY_NAME('SPFLI').
* vytvoření struktury instance
CREATE DATA R_STRUCT TYPE HANDLE R_STRUCT_DESCR.
* přiřazení field symbolu
ASSIGN R_STRUCT->* TO <FS_STRUKTURA>.

* nyní je připravena dynamická struktura podle tabulky SPFLI
* a teď se z ni udělá dynamická tabulka

FIELD-SYMBOLS: <FS_TABULKA> TYPE ANY.
DATA: R_TABLE_DESCR TYPE REF TO CL_ABAP_TABLEDESCR.
DATA: R_TABLE TYPE REF TO DATA.
DATA: KEY TYPE ABAP_KEYDESCR_TAB.
DATA: R_CX_ROOT TYPE REF TO CX_ROOT.
DATA: MESS TYPE STRING.

TRY.
    R_TABLE_DESCR = CL_ABAP_TABLEDESCR=>CREATE(
    P_LINE_TYPE = R_STRUCT_DESCR
    P_TABLE_KIND = CL_ABAP_TABLEDESCR=>TABLEKIND_SORTED
    P_UNIQUE = ABAP_TRUE
    ).
  CATCH CX_SY_TABLE_CREATION INTO R_CX_ROOT.
    MESS = R_CX_ROOT->GET_TEXT( ).
ENDTRY.
CREATE DATA R_TABLE TYPE HANDLE R_TABLE_DESCR.
ASSIGN R_TABLE->* TO <FS_TABULKA>.
```

3.1.16 Dereference

```
REPORT  YBR_DEREFERENCING.

DATA:
* datový objekt cityfrom je stejný typ jako spfli-cityfrom
      CITYFROM TYPE SPFLI-CITYFROM VALUE 'Paris',

* datový objekt který může být referencí na ref_cityto je stejný
* Typ jako spfli-cityto
      REF_CITYTO TYPE REF TO SPFLI-CITYTO.


FIELD-SYMBOLS <FS> TYPE ANY.

* reference ref_cityto bude ukazovat na objekt, který je stejný
* typ jako spfli-cityfrom, ale přesto je atributem typu
* spfli-cityto
GET REFERENCE OF CITYFROM INTO REF_CITYTO.
ASSIGN REF_CITYTO->* TO <FS>.
WRITE: <FS>.

* pokud se klikne na vypsané slovo na výstupu, získá se
* nápověda pro pole spfli-cityto a NE pro pole spfli-cityfrom!
* Důvodem je skutečnost, že referenční proměnná byla typována
* staticky.
```

3.1.17 Up-casting a down-casting

Jedná se o přetypování datových typů. Tato technika může být efektivně využita například v user-exitech v případě a to v případě kdy je uvnitř použitý parametr jako hodnota a je potřeba používat jeho strukturu.

```
REPORT  YBR_TYPE_CASTING.

DATA:  WA_VBAK TYPE VBAK.

SELECT SINGLE * FROM VBAK INTO WA_VBAK
WHERE VBELN EQ '0001000003'.
```

```
PERFORM DO_SOMETHING USING WA_VBAK.

*---------------------------------------------------------------*
*      Form  do_something
*---------------------------------------------------------------*
FORM DO_SOMETHING USING VALUE(P1).

* výpis: p1-mandt.
* chyba:  datový objekt "P1" nemá žádnou strukturu a proto
* nemůže být žádná komponenta použita přímo!!!

* Místo toho musíme přiřadit P1 objektu se strukturou VBAK
  DATA: REF_P1 TYPE REF TO DATA,
        REF_TO_VBAK TYPE REF TO VBAK.
  GET REFERENCE OF P1 INTO REF_P1.
  REF_TO_VBAK ?= REF_P1.

* nyní se pracuje s výrazem "ref_to_vbak->*" ve stejném
* významu jako s wa_vbak, příklad:

  WRITE: REF_TO_VBAK->*-ERDAT.
ENDFORM.
```

3.1.18 ASSERT

Logická podmínka ASSERT umožňuje ověření konzistentního stavu programu.

```
ASSERT <log_expr>.
```

Během zpracování programu musí být pravdivá. V opačném případě přeruší program a vyvolá výjimku ASSERTION_FAILED. Toto přerušení běhu omezuje špatný vliv chyb. Toto lze využít zejména ve výrobních programech, které provádějí databázové změny. Pomáhá to nalézt chyby, které dočasně způsobí nesprávný stav programu a pomáhá to zlepšit kvalitu programování tím, že se hledá více chyb.

3.1.19 Logy

```
LOG-POINT ID <group> FIELS <f1 f2 f3>
```

Toto je nejjednodušší cesta jak uchovat logy z aplikace. Logy se prohlíží v transakci SAAB. Tento způsob logu je určený jen pro vývojáře a může být aktivován a deaktivován pomocí kontrolních skupin.

Další způsoby logování:

- Aplikační log (pro uživatele).
- Systémový log (pro administrátory báze).

Pro tyto typy je několik nových funkčních modulů:

- BAL_LOG_CREATE – vytváří aplikační log
- BAL_LOG_MSG_ADD – přidává zprávu do aplikačního logu
- BAL_LOG_EXCEPTION_ADD – přidává informace o výjimkách
- BAL_DSP_LOG_DISPLAY – zobrazuje aplikační log
- RSLG_WRITE_SYSLOG_ENTRY – zapisuje systémový log

```
REPORT  YBR_APPLICATION_LOG.
DATA LOG TYPE BAL_S_LOG.
DATA LOG_HANDLE TYPE BALLOGHNDL.


* nastavení hlavičkových dat logu
LOG-EXTNUMBER = 'Application Log Demo'.
LOG-ALUSER = SY-UNAME.
LOG-ALPROG = SY-REPID.


* vytvoření aplikačního logu
* -> LOG_HANDLE bude vracen
CALL FUNCTION 'BAL_LOG_CREATE'
  EXPORTING
    I_S_LOG      = LOG
  IMPORTING
    E_LOG_HANDLE = LOG_HANDLE
  EXCEPTIONS
    OTHERS       = 1.
```

```
* příklad logování uživatelské zprávy
DATA: MSG TYPE BAL_S_MSG.
MESSAGE S162(YMM).
MSG-MSGTY = SY-MSGTY.
MSG-MSGID = SY-MSGID.
MSG-MSGNO = SY-MSGNO.
MSG-MSGV1 = SY-MSGV1.
MSG-MSGV2 = SY-MSGV2.
MSG-MSGV3 = SY-MSGV3.
MSG-MSGV4 = SY-MSGV4.

CALL FUNCTION 'BAL_LOG_MSG_ADD'
  EXPORTING
    I_S_MSG       = MSG
    I_LOG_HANDLE  = LOG_HANDLE
  EXCEPTIONS
    LOG_NOT_FOUND = 1
    OTHERS        = 2.

* zobrazení logu
CALL FUNCTION 'BAL_DSP_LOG_DISPLAY'.
```

3.1.20 Automatický test logiky kódu ABAP

ABAP nyní umožňuje vytvořit části kódů sloužících pro testování funkcionality formů, metod a dalších částí kódu. Tento kód trvale zůstává v programu a test může být spuštěn kdykoliv se program změní. To dává jistotu, že změna nenarušuje stávající kód, a že všechny rozhodující části kódu pracují jak se očekávalo po nových změnách.

Testovací třídy musí být vždy definovány jako lokální a nemůžou být definovány v transakci SE24.

Metody testovacích tříd používají jednu z meto třídy CL_AUNIT_ASSER:

- Assert_initial
- Assert_not_initial
- Assert_bound
- Assert_not_bound
- Assert_equals
- Assert_differs
- Assert_subrc

Můžou se nastavovat různé úrovně rizika pro zpracování v různých systémech. V testovacím systému bývají zapnuty všechny úrovně.

```
REPORT  YBR_ABAP_UNIT.
* hlavní program
DATA: RESULT TYPE I.
PERFORM ADD_TWO_INTEGERS USING 1 2 CHANGING RESULT.
WRITE: RESULT.

* tento form by mel být automaticky testován
FORM ADD_TWO_INTEGERS USING NUMBER1  NUMBER2 CHANGING NUMBER3.
  NUMBER3 = NUMBER1 + NUMBER2.
ENDFORM.

* třída pro automatické testování (test test bývá spuštěn pomocí
* menu Program -> Test -> Unit test)
CLASS LCL_TEST DEFINITION FOR TESTING. "#AU Risk_Level Harmless
  PRIVATE SECTION.
    METHODS TESTMETHOD1 FOR TESTING.
ENDCLASS.

CLASS LCL_TEST IMPLEMENTATION.
  METHOD TESTMETHOD1.
    DATA: RES TYPE I.

    PERFORM ADD_TWO_INTEGERS USING 1 2 CHANGING RES.
    CL_AUNIT_ASSERT=>ASSERT_EQUALS(
```

```
    ACT =  RES
    EXP =  3
    MSG = 'Form SETCI uz neumi scitat!!!'
    QUIT = CL_AUNIT_ASSERT=>IF_AUNIT_CONSTANTS~NO ).
  ENDMETHOD.
ENDCLASS.
```

3.1.21 Nová standardní třída pro ALV list

Tento nový ALV list je nyní plně objektově orientovaný. V předchozích verzích se musel definovat katalog polí, ale nyní tento katalog již není nutný. Již může být vytvořen automaticky pomocí standardních metod třídy CL_SALV_TABLE. Nový ALV list není povolen pro vstupní pole (pole starší verze ALV se může použít jako vstupní pole přestože to nebylo SAPem nikdy podporováno). Zastaralé ALV listy vytvořené pomocí funkčního modelu REUSE_ALV_LIST_DISPLAY nebo třídy CL_GUI_ALV_GRID jsou stále v systému a funkční, ale nebudou podporovány SAPem a už bychom je neměli používat.

Nyní je celý ALV koncept je založen na využívání třídy CL_SALV_TABLE.

```
REPORT  YBR_ALV_OOP.
TABLES: MARA.
SELECT-OPTIONS: MATNR_OP FOR MARA-MATNR OBLIGATORY.

* interni tabulka, která bude zobrazena v ALV listu
DATA: ITAB TYPE TABLE OF MARA.

* reference na ALV instanci (CL_ALV_TABLE)
DATA: R_ALVTAB TYPE REF TO CL_SALV_TABLE.


* naplnění interní tabulky daty
SELECT * FROM MARA INTO TABLE ITAB
WHERE MATNR IN MATNR_OP.

TRY.
* vytvoření nové instance
    CALL METHOD CL_SALV_TABLE=>FACTORY
* EXPORTING
```

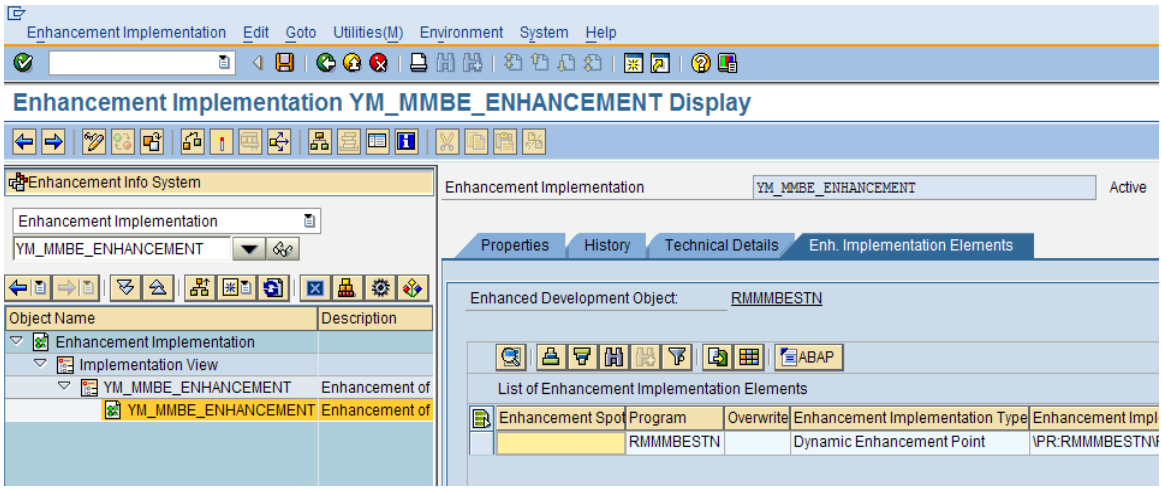
```
* list_display = 'X'
    IMPORTING
    R_SALV_TABLE = R_ALVTAB
    CHANGING
    T_TABLE = ITAB.

* zobrazení ALV listu
    CALL METHOD R_ALVTAB->DISPLAY.
  CATCH CX_ROOT.
ENDTRY.
```

Je-li struktura řádku zobrazující se vnitřní tabulky převzata z DDIC, nemusí se vytvářet žádné textové popisky a jejich překlady (všechny texty jsou převzaty z DDIC datových prvků). V opačném případě se musí použít speciální metody instancí sloupců pro definování těchto řetězců. Je také možné definovat různé barvy řádků a buněk.

3.1.22 Nový způsob rozšíření (enhancement)

Systém SAP podporuje nové metody standardního rozšíření. Ve skutečnosti jsou vkládány malé zákaznické části kódu do standardních programů bez vlastní modifikace. Toto rozšíření může být aktivováno a deaktivováno odděleně od standardního kódu. Standardní kód není zasažen jakoukoli změnou a můžou se na něj dále aplikovat aktualizace. Rozšíření se nemůže vkládat kamkoliv do kódu ale jen na místa pro to určená. Zobrazit seznam rozšíření můžeme pomocí transakce SPAU_ENH.

Ačkoli je tento způsob rozšířený velice mocný nástroj, měl by se používat pouze pokud neexistuje standardní řešení pomocí aktualizací a měl by být použit jen pro malé modifikace.



Obr.2: Transakce SPAU_ENH

3.1.23 Regulární výrazy

Nová verze systému SAP nabízí velmi silný nástroj pro hledání nebo modifikaci textů známí jako regulární výrazy. Ve staré verzi SAPu musel programátor použít příkazy FIND a REPLACE. Tyto příkazy mají velká omezení.

Regulární výrazy slouží ke zpracovávání textových řetězců, a to tak, že se definuje maska, které vyhoví jen určité řetězce. Jednodušeji řečeno, regulární výraz je řetězec několika znaků. Textový řetězec pak tomuto regulárnímu výrazu vyhovuje jen tehdy, jestliže obsahuje stejné znaky, které byly definovány v regulárním výrazu. K definování regulárních výrazů se navíc používají operátory (tzv. metaznaky), které mají speciální význam.

Pro práci s regulárními výrazy se používají příkazy FING REGEX a REPLACE REGEX nebo standardní třídy CL_ABAP_REGEX a CL_ABAP_MATCHER.

Regulární výrazy se skládají z:

- Literál : a b c 1 2 3 , = / …
- Operátor (metaznak): . * + ? \ ^ $ ( ) [ ] { }

Operátor	Význam	Příklad	Příklad pro nalezený text	Příklad pro ignorovaný text
.	libovolný 1 znak	.	a, a, 9, #	abc
r*	žádnej nebo vice znaků r	ab*	a, ab, abb, abbb	b, aba
r+	jeden nebo vice znaků r	ab+	ab, abb, abbb	a, b, aba
r{m,n}	min. m, max. n znaků r	a{2,4}	aa, aaa, aaaa	a, aaaaa, aba
r{m}	právě m znaků r	a{3}	aaa	a, aa, aaaa, bbb
r?	žádnej nebo jeden znak r	ab?a	aa, aba	abba, aca
r|s	R nebo s	a+|b+	a, b, aa, bb, aaa	ab, aabb
( )	Podskupiny	a(b|c)a	aba, aca	aa, abca
^ $	Začátek / konec řádku	^a	a	b a
\< \>	Začátek / konec slova	\<abc	abc	xyzabc

Tab. 2 : Příklad použití operátorů

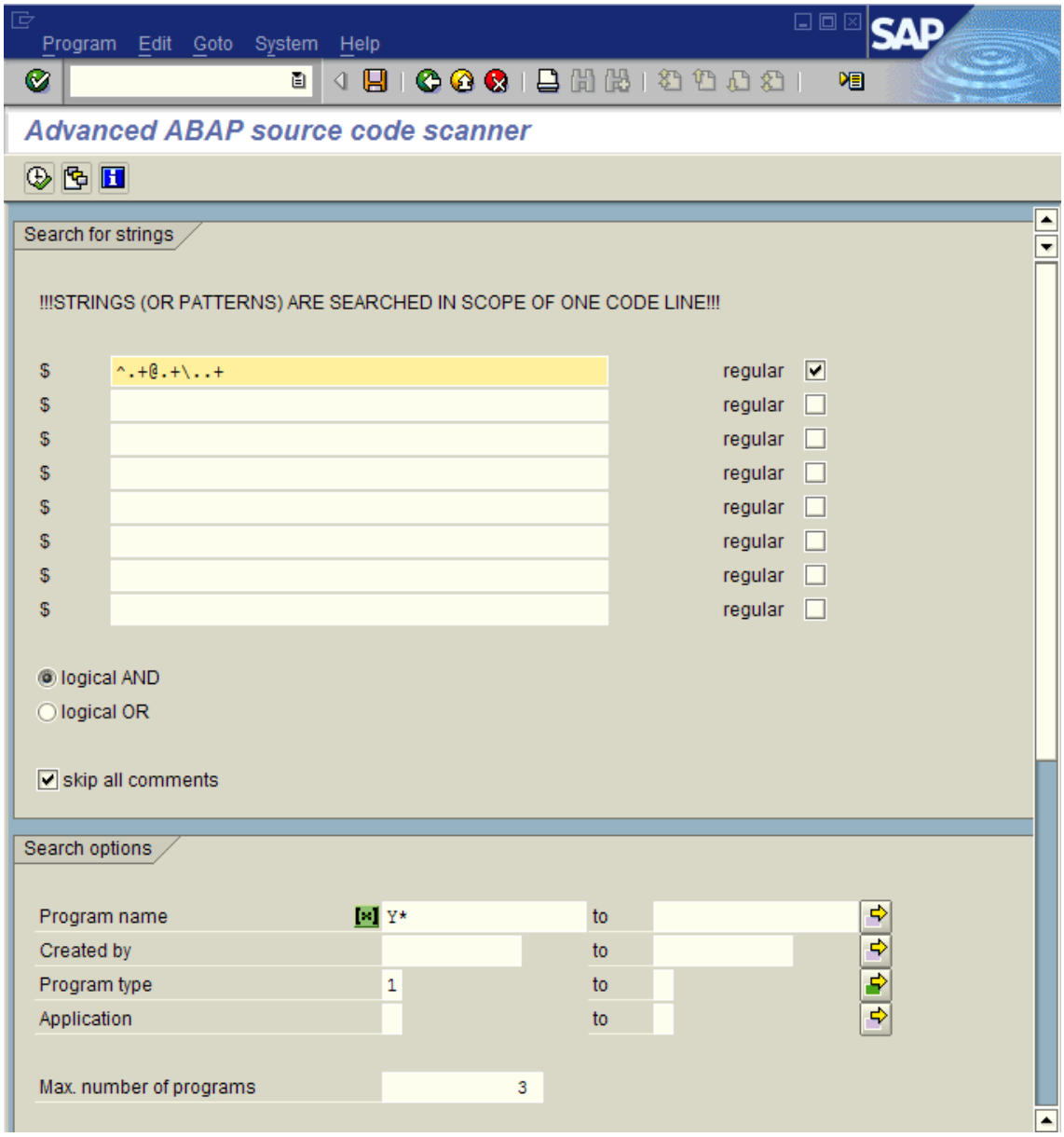
Pokud je potřeba použít nějaký operátor jako znak, píše se před něj operátor „\“.

4 Příklad použití regulárních výrazů

Na příkladu vyhledávání textů v programech je ukázáno použití regulárních výrazů. Regulární výrazy se v předchozích verzích vůbec nevyskytovaly. Díky přidání do nové verze se otevírají nové možnosti prohledávání textů.

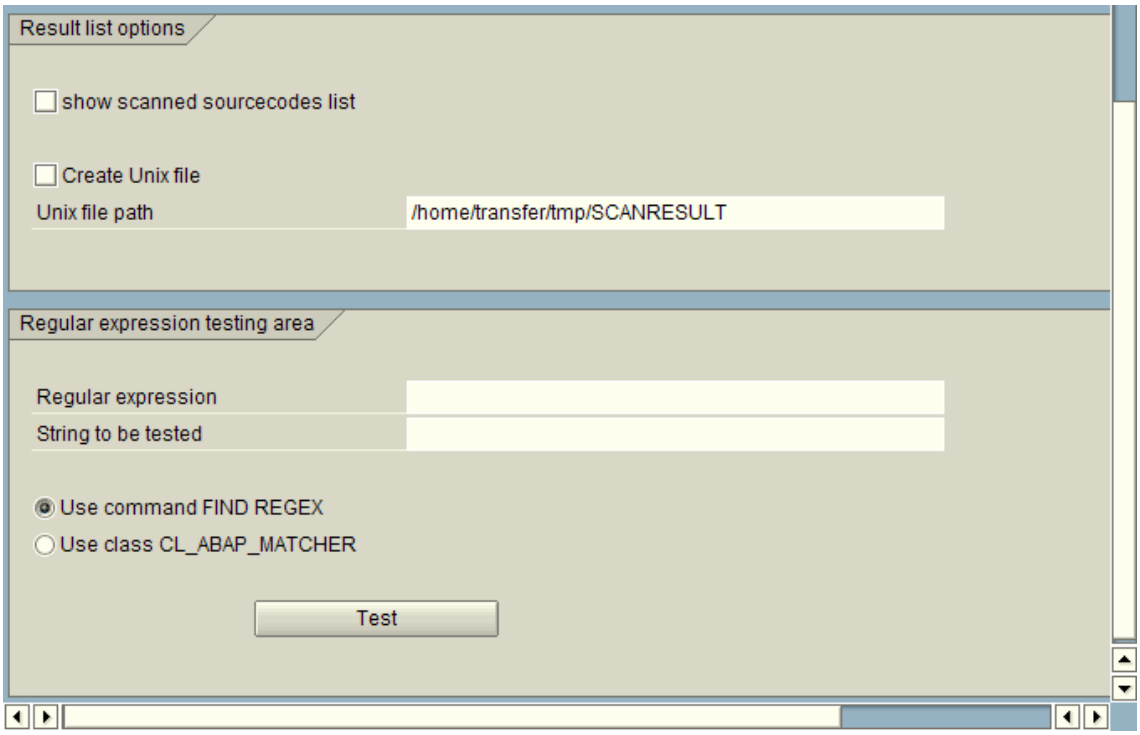
Je definována výběrová obrazovka kde se nastavují parametry pro hledání a hledaný řetězec. Je možno zadat až 8 různých textů P1 až P8 pro vyhledávání a u každého si vybrat zda má být regulární nebo obyčejný.

Prohledávané programy lze vybírat podle názvu, autora nebo typu.



Obr.3: První část výběrové obrazovky

Poslední část výběrové obrazovky slouží k otestování regulárních výrazů zda jsou správně napsány. Výsledky hledání lze uložit do souboru na server.



Obr.4: Druhá část výběrové obrazovky

Testování je možné pomocí příkazu FIND REGEX nebo pomocí standardní třídy CL_ABAP_MATCHER.

```
    IF FIND EQ 'X'.
* use command FIND REGEX for the test
      FIND FIRST OCCURRENCE OF REGEX REGXSTR IN TESTSTR.
      IF SY-SUBRC EQ 0.
        MESSAGE 'Match' TYPE 'S'.
      ELSE.
        MESSAGE 'Don''t match' TYPE 'S'.
      ENDIF.
      EXIT.
    ENDIF.

    IF MATCHER EQ 'X'.
* use class CL_ABAP_MATCHER for the test
      DATA:
        MATCHER TYPE REF TO CL_ABAP_MATCHER,
        MATCH   TYPE C LENGTH 1.
      TRY.
          MATCHER = CL_ABAP_MATCHER=>CREATE( PATTERN = REGXSTR
                                             IGNORE_CASE = 'X'
                                             TEXT        = TESTSTR ).
```

```
      CATCH CX_SY_INVALID_REGEX.
        MESSAGE 'Invalid syntax!' TYPE 'S'.
        EXIT.
    ENDTRY.
    MATCH = MATCHER->MATCH( ).
    IF MATCH EQ 'X'.
      MESSAGE 'Match' TYPE 'S'.
    ELSE.
      MESSAGE 'Don''t match' TYPE 'S'.
    ENDIF.
    EXIT.
  ENDIF.
```

Po spuštění programu se vybírají v transparentní tabulce TRDIR názvy programů které se budou prohledávat podle zadání na výběrové obrazovce. Tento výběr se vloží do interní tabulky NAMES.

```
SELECT NAME INTO TABLE NAMES FROM TRDIR
WHERE NAME IN NAME_OP
AND   CNAM IN CNAM_OP
AND   SUBC IN SUBC_OP
AND   APPL IN APPL_OP.
```

Postupně se procházejí nalezené programy a načítá se kód programu do interní tabulky SOURCE pomocí příkazu READ REPORT.

```
READ REPORT NAMES-NAME INTO SOURCE.
```

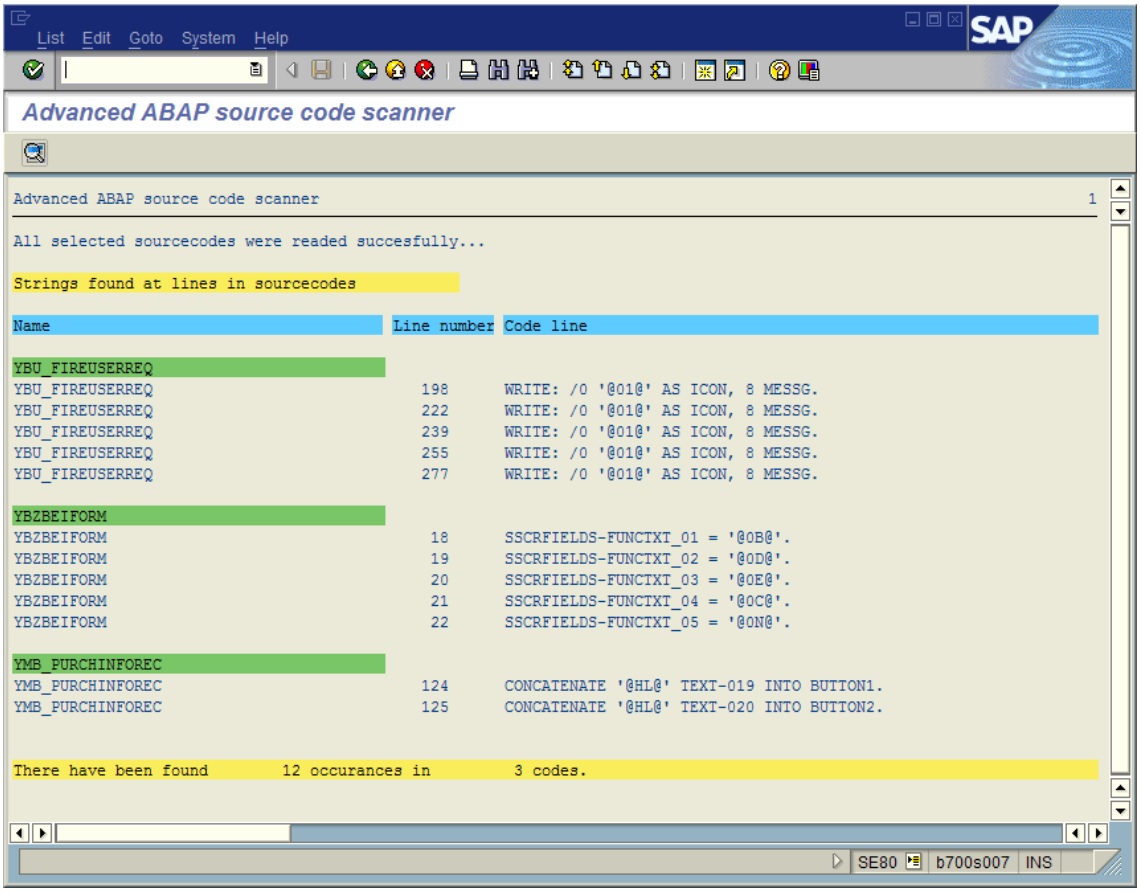
Načtený kód se prohledává buď pomocí klasického prohledávání textu nebo pomocí regulárních výrazů. Způsob prohledávání se zadává na výběrové obrazovce. Pro klasické hledání textu je použit příkaz SEARCH.

```
SEARCH C1 FOR P_STRING.
```

Pokud ze vyhledává pomocí regulárních výrazů, je použit příkaz REGEX

```
FIND FIRST OCCURENCE OF REGEX P_STRING IN C1.
```

Výsledek prohledávání programů se ukládá do interní tabulky OUTPUT_A a zobrazí se na výstupní obrazovce. Při poklikání myší na název programu na výstupní obrazovce se zavolá funkční modul EDITOR_PROGRAM který spustí editor programu a otevře v něm nalezený program.



Obr.5: Výstup programu

Závěr

Informační systémy a technologie se neustále vyvíjejí a pro podnik je důležité držet krok s tímto vývojem. Starší verze přestávají být postupně podporovány a vyvíjeny. Nasazení nové verze systému SAP v podniku je důležitý pro udržení s tempem vývoje a používáním nové technologie kvůli jednoduššímu udržení běhu podniku s konkurencí.

Nové verze SAPu pomáhají s prací jak uvnitř podniku, tak ve službách pro zákazníky. Jako příklad může být použitý velký rozmach webových služeb, jako jsou internetové obchody a zjednodušená propagace nových a stávajících výrobků.

S nástupem nových verzí se také zjednodušuje uživatelský vývoj systému. V nové verzi je plně podporováno objektové programovaní a přidány či upraveny další techniky programování . Velký pokrok je vidět také na editoru programů a v ladění programů. Zrychluje se tím práce na nových projektech v rámci rozvoje firmy.

V této práci jsou popsány hlavní rozdíly v programovaní uživatelských transakcí mezi verzemi systému SAP a na závěr je popsané řešení problému vyhledávání kódu pomocí regulárních výrazů, což dříve nebylo možné.

Program vyhledává texty jak pomocí klasické metody, tak pomocí regulárních výrazů. Výsledný program se povedl a dá se využívat jako pomocný nástroj pro programátory. Rychlost programu použitím regulárních výrazů se téměř nezměnila a i přes velký počet programů v celém systému SAP nalezne výsledek relativně rychle.

Literatura

[1] Firemní materiály SAP.

[2] Vrana I., Richta K.: Zásady a postupy zavádění podnikových informačních sytémů, Grada 2004, ISBN 80-247-1103-6.

[3] Řepa V. : Podnikové procesy, Grada, 2007, ISBN 978-80-247-2252-8.

[4] http://www.sap.com

[5] http://www.sdn.sap.com

Příloha A

Příloha obsahuje příklad použití regulárních výrazů v programu. Tento program slouží k prohledávání textů ve všech programech.

```
REPORT YBRSCN00 LINE-SIZE 500.
TABLES: TRDIR, SSCRFIELDS.

*---------------------------------------------------------------*
* Definition of selection screen
*---------------------------------------------------- ------*
SELECTION-SCREEN: BEGIN OF BLOCK STRS WITH FRAME TITLE TEXT-001.
SELECTION-SCREEN: SKIP 2.
SELECTION-SCREEN COMMENT 1(70) TEXT-002.
SELECTION-SCREEN: SKIP 1.

SELECTION-SCREEN: BEGIN OF LINE.
SELECTION-SCREEN COMMENT (6) TEXT-020.
PARAMETERS: STRING1(128) TYPE C.
SELECTION-SCREEN COMMENT 64(7) TEXT-019.
PARAMETERS: STR1REX AS CHECKBOX.
SELECTION-SCREEN: END OF LINE.

SELECTION-SCREEN: BEGIN OF LINE.
SELECTION-SCREEN COMMENT (6) TEXT-020.
PARAMETERS: STRING2(128) TYPE C.
SELECTION-SCREEN COMMENT 64(7) TEXT-019.
PARAMETERS: STR2REX AS CHECKBOX.
SELECTION-SCREEN: END OF LINE.

SELECTION-SCREEN: BEGIN OF LINE.
SELECTION-SCREEN COMMENT (6) TEXT-020.
PARAMETERS: STRING3(128) TYPE C.
SELECTION-SCREEN COMMENT 64(7) TEXT-019.
PARAMETERS: STR3REX AS CHECKBOX.
SELECTION-SCREEN: END OF LINE.

SELECTION-SCREEN: BEGIN OF LINE.
SELECTION-SCREEN COMMENT (6) TEXT-020.
PARAMETERS: STRING4(128) TYPE C.
```

```
SELECTION-SCREEN COMMENT 64(7) TEXT-019.
PARAMETERS: STR4REX AS CHECKBOX.
SELECTION-SCREEN: END OF LINE.

SELECTION-SCREEN: BEGIN OF LINE.
SELECTION-SCREEN COMMENT (6) TEXT-020.
PARAMETERS: STRING5(128) TYPE C.
SELECTION-SCREEN COMMENT 64(7) TEXT-019.
PARAMETERS: STR5REX AS CHECKBOX.
SELECTION-SCREEN: END OF LINE.

SELECTION-SCREEN: BEGIN OF LINE.
SELECTION-SCREEN COMMENT (6) TEXT-020.
PARAMETERS: STRING6(128) TYPE C.
SELECTION-SCREEN COMMENT 64(7) TEXT-019.
PARAMETERS: STR6REX AS CHECKBOX.
SELECTION-SCREEN: END OF LINE.

SELECTION-SCREEN: BEGIN OF LINE.
SELECTION-SCREEN COMMENT (6) TEXT-020.
PARAMETERS: STRING7(128) TYPE C.
SELECTION-SCREEN COMMENT 64(7) TEXT-019.
PARAMETERS: STR7REX AS CHECKBOX.
SELECTION-SCREEN: END OF LINE.

SELECTION-SCREEN: BEGIN OF LINE.
SELECTION-SCREEN COMMENT (6) TEXT-020.
PARAMETERS: STRING8(128) TYPE C.
SELECTION-SCREEN COMMENT 64(7) TEXT-019.
PARAMETERS: STR8REX AS CHECKBOX.
SELECTION-SCREEN: END OF LINE.

SELECTION-SCREEN: SKIP.

* logical connection between strings
PARAMETERS: ANDS RADIOBUTTON GROUP RAD0 DEFAULT 'X',
            ORS  RADIOBUTTON GROUP RAD0.

SELECTION-SCREEN: SKIP.
```

```
** all strings within one ABAP command
*PARAMETERS: ONECOM AS CHECKBOX DEFAULT 'X'.

* skip comments
PARAMETERS: SKIPC AS CHECKBOX DEFAULT 'X'.

SELECTION-SCREEN: END OF BLOCK STRS.

SELECTION-SCREEN: BEGIN OF BLOCK OPTS WITH FRAME TITLE TEXT-003.
SELECTION-SCREEN: SKIP.
* program name(s)
SELECT-OPTIONS: NAME_OP FOR TRDIR-NAME.
* author(s)
SELECT-OPTIONS: CNAM_OP FOR TRDIR-CNAM DEFAULT SY-UNAME.
* program type(s)
SELECT-OPTIONS: SUBC_OP FOR TRDIR-SUBC.
* application(s)
SELECT-OPTIONS: APPL_OP FOR TRDIR-APPL.
SELECTION-SCREEN: SKIP.
* limitation number of programs
PARAMETERS: LIMIT_P TYPE I.
SELECTION-SCREEN: SKIP.
SELECTION-SCREEN: END OF BLOCK OPTS.

SELECTION-SCREEN: BEGIN OF BLOCK LIST WITH FRAME TITLE TEXT-011.
SELECTION-SCREEN: SKIP.
PARAMETERS: REPA AS CHECKBOX.
SELECTION-SCREEN: SKIP.
PARAMETERS:
CREATF AS CHECKBOX,
FPATH(128) DEFAULT '/home/transfer/tmp/SCANRESULT' LOWER CASE.
SELECTION-SCREEN: SKIP.
SELECTION-SCREEN: END OF BLOCK LIST.

SELECTION-SCREEN: BEGIN OF BLOCK REGEXTEST WITH FRAME TITLE
TEXT-022.
SELECTION-SCREEN: SKIP.
PARAMETERS: REGXSTR(128) TYPE C,
            TESTSTR(128) TYPE C.
SELECTION-SCREEN: SKIP.
PARAMETERS:  FIND    RADIOBUTTON GROUP RAD1 DEFAULT 'X',
```

```
             MATCHER RADIOBUTTON GROUP RAD1.
SELECTION-SCREEN: SKIP.
SELECTION-SCREEN: BEGIN OF LINE,
      PUSHBUTTON 20(20) TEXT-023  USER-COMMAND TEST,
END OF LINE.
SELECTION-SCREEN: SKIP.
SELECTION-SCREEN: END OF BLOCK REGEXTEST.


*---------------------------------------------------------------*
* INITIALIZATION
* initialization of selection screen options
*---------------------------------------------------------------*
INITIALIZATION.
* in default: executables and includes will be scanned
  SUBC_OP-SIGN = 'I'.
  SUBC_OP-OPTION = 'EQ'.
  SUBC_OP-LOW = '1'.
  APPEND SUBC_OP.
  SUBC_OP-LOW = 'I'.
  APPEND SUBC_OP.

* in defalut: only Y* program will be scanned
  NAME_OP-SIGN = 'I'.
  NAME_OP-OPTION = 'CP'.
  NAME_OP-LOW = 'Y*'.
  APPEND NAME_OP.

*---------------------------------------------------------------*
* Global data definitions
*---------------------------------------------------------------*
* definition of internal table for program names
  TYPES: BEGIN OF NAMES_LINE,
         NAME LIKE TRDIR-NAME,
         END OF NAMES_LINE.
  DATA: NAMES TYPE TABLE OF NAMES_LINE WITH HEADER LINE.

* definition of internal table for source codes
  TYPES: BEGIN OF SOURCE_LINE,
           LINE(256),
```

```
         END OF SOURCE_LINE.
  DATA: SOURCE TYPE TABLE OF SOURCE_LINE WITH HEADER LINE.


* definition of internal table for output report, part A
* i.e. name of program, message
  TYPES: BEGIN OF OUTPUT_A_LINE,
          NAME LIKE TRDIR-NAME,
          MESSAGE(128) TYPE C,
          CRITICAL(1) TYPE C,
         END OF OUTPUT_A_LINE.
  DATA: OUTPUT_A TYPE TABLE OF OUTPUT_A_LINE WITH HEADER LINE
        WITH KEY NAME.

* definition of internal table for output report, part B
* i.e. name of program, lines of code
  TYPES: BEGIN OF OUTPUT_B_LINE,
            NAME LIKE TRDIR-NAME,
             STRING(128) TYPE C,
             CODE_LINE(256) TYPE C,
            LINNUM TYPE I,
         END OF OUTPUT_B_LINE.
  DATA: OUTPUT_B TYPE TABLE OF OUTPUT_B_LINE WITH HEADER LINE.

* definition of internal table for positions of strings
  TYPES: BEGIN OF OCCURANCE_LINE,
           STRING(128) TYPE C,
           CODE_LINE(128) TYPE C,
          LINNUM TYPE I,
          OFFSET TYPE I,
         END OF OCCURANCE_LINE.
  DATA: OCCURANCE TYPE TABLE OF OCCURANCE_LINE WITH HEADER LINE.


  DATA:
* logical flangs
  STRING1_EXISTS(1) TYPE C,
  STRING2_EXISTS(1) TYPE C,
  STRING3_EXISTS(1) TYPE C,
  STRING4_EXISTS(1) TYPE C,
```

```
  STRING5_EXISTS(1) TYPE C,
  STRING6_EXISTS(1) TYPE C,
  STRING7_EXISTS(1) TYPE C,
  STRING8_EXISTS(1) TYPE C,

* number of program scanned
  SOURCECNT TYPE I,

* increase of percentage after one program proccesed
  PRCINC TYPE F,
  PRCINC_I TYPE F, "I

* current status of progress indicator
  STATUS TYPE F, "I
  LAST_DISPLAYED_STATUS TYPE F VALUE 0,
  DIFFERENCE TYPE F,

* two parts of line of code for spliting the line to command and
inline
* comment
  C1(256) TYPE C,
  C2(256) TYPE C.

  DATA: AT_LEAST_ONE_CRITICAL(1) TYPE C.

* number of occurances (integer and string variant)
  DATA: TOTALOC TYPE I, TOTALOC_C(10) TYPE C,
* number of programs (integer and string variant)
  TOTALPR TYPE I, TOTALPR_C(10) TYPE C,
* message for short statistic
  MSG(128) TYPE C,

* variables for command HIDE
 HIDDEN_NAME LIKE OUTPUT_B-NAME,
 HIDDEN_LINNUM LIKE OUTPUT_B-LINNUM,
 HIDDEN_VALID_LINE TYPE C.

* num of programs found
  DATA: PROGCNT TYPE I.

*---------------------------------------------------------------*
```

```
* AT SELECTION-SCREEN
* description of behavior
*---------------------------------------------------------------*
AT SELECTION-SCREEN.

  IF SSCRFIELDS-UCOMM EQ 'TEST'.
    TRANSLATE TESTSTR TO UPPER CASE.
    IF FIND EQ 'X'.
* use command FIND REGEX for the test
      FIND FIRST OCCURRENCE OF REGEX REGXSTR IN TESTSTR.
      IF SY-SUBRC EQ 0.
        MESSAGE 'Match' TYPE 'S'.
      ELSE.
        MESSAGE 'Don''t match' TYPE 'S'.
      ENDIF.

      EXIT.
    ENDIF.

    IF MATCHER EQ 'X'.
* use class CL_ABAP_MATCHER for the test
      DATA:
        MATCHER TYPE REF TO CL_ABAP_MATCHER,
        MATCH   TYPE C LENGTH 1.

      TRY.
          MATCHER = CL_ABAP_MATCHER=>CREATE( PATTERN = REGXSTR
                                             IGNORE_CASE = 'X'
                                             TEXT        = TESTSTR ).
        CATCH CX_SY_INVALID_REGEX.
          MESSAGE 'Invalid syntax!' TYPE 'S'.
          EXIT.
      ENDTRY.

      MATCH = MATCHER->MATCH( ).

      IF MATCH EQ 'X'.
        MESSAGE 'Match' TYPE 'S'.
      ELSE.
        MESSAGE 'Don''t match' TYPE 'S'.
```

```
      ENDIF.
      EXIT.
    ENDIF.

  ENDIF.
*---------------------------------------------------------------*
* START-OF-SELECTION
* description of behavior
*---------------------------------------------------------------*
START-OF-SELECTION.
  PERFORM STAT(YSTAT). "Statistik fortschreiben! Achtung: COMMIT
WORK!

* regular expression syntax check!
  DATA: REG_ERROR TYPE C VALUE SPACE.
  PERFORM REGULAR_EXPRESSION_CHECK.
  IF REG_ERROR EQ 'X'.
    MESSAGE TEXT-021 TYPE 'S'.
    EXIT.
  ENDIF.

* fill the internal table NAMES with names of selected programs
* according to selection-options
  PERFORM FILL_NAMES.

* in frontend processing only count the increase of percentage
proccesed
* after one of programs was processed
  IF ( ( SY-BATCH EQ SPACE ) AND ( SOURCECNT NE 0 ) ).
    PRCINC = 100 / SOURCECNT.
    PRCINC_I = PRCINC.
  ENDIF.

* read sources of programs and analyze them
* (fills the internal table OUTPUT_A and OUTPUT_B)
  CLEAR PROGCNT.
  PERFORM READ_SOURCES.

  IF ( SY-BATCH EQ SPACE ).
* sorting result tables - stand by
    CALL FUNCTION 'SAPGUI_PROGRESS_INDICATOR'
```

```
      EXPORTING
        PERCENTAGE = STATUS
        TEXT       = TEXT-014.
  ENDIF.

* write the result list
  PERFORM WRITE_OUTPUT.

* creating an unix file
  IF CREATF EQ 'X'.
    DATA: FILE_LINE(500) TYPE C.
    DATA: LINTXT(10) TYPE C.

    OPEN DATASET FPATH FOR OUTPUT IN TEXT MODE ENCODING
    NON-UNICODE.

    IF SY-SUBRC NE 0.
* An error occured during the UNIX file creation!
      MESSAGE E040(YMM).
    ENDIF.

    LOOP AT OUTPUT_B.

      CLEAR: FILE_LINE, LINTXT.
      LINTXT = OUTPUT_B-LINNUM.

      CONCATENATE OUTPUT_B-NAME OUTPUT_B-CODE_LINE OUTPUT_B-
STRING LINTXT
      INTO FILE_LINE SEPARATED BY ';'.
      TRANSFER FILE_LINE TO FPATH.
    ENDLOOP.
    CLOSE DATASET FPATH.
  ENDIF.

*** END OF PROGRAM ***

*---------------------------------------------------------------*
* FORM FILL_NAMES
* Filling the internal table NAMES with names of programs from
* table
* TRDIR which will be scanned.
```

```
*---------------------------------------------------------------------*
FORM FILL_NAMES.
* select names of programs from table TRDIR
  SELECT NAME INTO TABLE NAMES FROM TRDIR
  WHERE NAME IN NAME_OP
  AND   CNAM IN CNAM_OP
  AND   SUBC IN SUBC_OP
  AND   APPL IN APPL_OP.

* save the number of found program names
  SOURCECNT = SY-DBCNT.

ENDFORM.                    "FILL_NAMES

*---------------------------------------------------------------------*
* FORM READ_SOURCES
* This routine reads each of programs(their names are in
* table NAMES)
* and fills the internal table OUTPUT_A with messages.
* It also calls the routine for analyzing readed source codes.
*---------------------------------------------------------------------*
FORM READ_SOURCES.

* clear the current progress indicator status
  CLEAR: STATUS, LAST_DISPLAYED_STATUS.

  LOOP AT NAMES.

* initialization of internal tables for work
    REFRESH SOURCE.
    REFRESH OCCURANCE.

* initialization of logical flags
    STRING1_EXISTS = SPACE.
    STRING2_EXISTS = SPACE.
    STRING3_EXISTS = SPACE.
    STRING4_EXISTS = SPACE.
    STRING5_EXISTS = SPACE.
    STRING6_EXISTS = SPACE.
    STRING7_EXISTS = SPACE.
```

```
    STRING8_EXISTS = SPACE.

* read source of given program into table SOURCE
    READ REPORT NAMES-NAME INTO SOURCE.


    IF SY-SUBRC NE 0.
* program could not be read!!!
      MOVE NAMES-NAME TO OUTPUT_A-NAME.
      MOVE TEXT-004 TO OUTPUT_A-MESSAGE.
      MOVE 'X' TO OUTPUT_A-CRITICAL.
      APPEND OUTPUT_A.
* go to the next program
      CONTINUE.
    ELSE.
* program succesfully scanned
      MOVE NAMES-NAME TO OUTPUT_A-NAME.
      MOVE TEXT-005 TO OUTPUT_A-MESSAGE.
      MOVE SPACE TO OUTPUT_A-CRITICAL.
      APPEND OUTPUT_A.
    ENDIF.

* fill the internal table OCCURANCE
    PERFORM CHECK_OCCURANCE.

* set logical flags according to the content of internal table
* OCCURANCE
    LOOP AT OCCURANCE WHERE STRING EQ STRING1.
    ENDLOOP.
    IF SY-SUBRC EQ 0.
      STRING1_EXISTS = 'X'.
    ENDIF.
    LOOP AT OCCURANCE WHERE STRING EQ STRING2.
    ENDLOOP.
    IF SY-SUBRC EQ 0.
      STRING2_EXISTS = 'X'.
    ENDIF.
    LOOP AT OCCURANCE WHERE STRING EQ STRING3.
    ENDLOOP.
    IF SY-SUBRC EQ 0.
```

```
      STRING3_EXISTS = 'X'.
    ENDIF.
    LOOP AT OCCURANCE WHERE STRING EQ STRING4.
    ENDLOOP.
    IF SY-SUBRC EQ 0.
      STRING4_EXISTS = 'X'.
    ENDIF.
    LOOP AT OCCURANCE WHERE STRING EQ STRING5.
    ENDLOOP.
    IF SY-SUBRC EQ 0.
      STRING5_EXISTS = 'X'.
    ENDIF.
    LOOP AT OCCURANCE WHERE STRING EQ STRING6.
    ENDLOOP.
    IF SY-SUBRC EQ 0.
      STRING6_EXISTS = 'X'.
    ENDIF.
    LOOP AT OCCURANCE WHERE STRING EQ STRING7.
    ENDLOOP.
    IF SY-SUBRC EQ 0.
      STRING7_EXISTS = 'X'.
    ENDIF.
    LOOP AT OCCURANCE WHERE STRING EQ STRING8.
    ENDLOOP.
    IF SY-SUBRC EQ 0.
      STRING8_EXISTS = 'X'.
    ENDIF.

    IF ANDS EQ 'X'.
* operators between strings are ANDs

* in AND case take initial strings as existing for logical
* evaluation
      IF STRING1 IS INITIAL.
        STRING1_EXISTS = 'X'.
      ENDIF.
      IF STRING2 IS INITIAL.
        STRING2_EXISTS = 'X'.
      ENDIF.
      IF STRING3 IS INITIAL.
```

```
        STRING3_EXISTS = 'X'.
      ENDIF.
      IF STRING4 IS INITIAL.
        STRING4_EXISTS = 'X'.
      ENDIF.
      IF STRING5 IS INITIAL.
        STRING5_EXISTS = 'X'.
      ENDIF.
      IF STRING6 IS INITIAL.
        STRING6_EXISTS = 'X'.
      ENDIF.
      IF STRING7 IS INITIAL.
        STRING7_EXISTS = 'X'.
      ENDIF.
      IF STRING8 IS INITIAL.
        STRING8_EXISTS = 'X'.
      ENDIF.

* if all strings are existing in given sourcecode, append
* output_B
      IF  ( STRING1_EXISTS EQ 'X' )
      AND ( STRING2_EXISTS EQ 'X' )
      AND ( STRING3_EXISTS EQ 'X' )
      AND ( STRING4_EXISTS EQ 'X' )
      AND ( STRING5_EXISTS EQ 'X' )
      AND ( STRING6_EXISTS EQ 'X' )
      AND ( STRING7_EXISTS EQ 'X' )
      AND ( STRING8_EXISTS EQ 'X' ).

        LOOP AT OCCURANCE.

          OUTPUT_B-NAME = NAMES-NAME.
          OUTPUT_B-STRING = OCCURANCE-STRING.
          OUTPUT_B-CODE_LINE = OCCURANCE-CODE_LINE.
          OUTPUT_B-LINNUM = OCCURANCE-LINNUM.
          APPEND OUTPUT_B.

        ENDLOOP.

* increase number of programs found
```

```
        ADD 1 TO PROGCNT.
      ENDIF.
    ELSE.
* operators between strings are ORs

* if at least one strings are existing in given sourcecode,
append
* output_B
      IF  ( STRING1_EXISTS EQ 'X' )
      OR ( STRING2_EXISTS EQ 'X' )
      OR ( STRING3_EXISTS EQ 'X' )
      OR ( STRING4_EXISTS EQ 'X' )
      OR ( STRING5_EXISTS EQ 'X' )
      OR ( STRING6_EXISTS EQ 'X' )
      OR ( STRING7_EXISTS EQ 'X' )
      OR ( STRING8_EXISTS EQ 'X' ).
        LOOP AT OCCURANCE.
          OUTPUT_B-NAME = NAMES-NAME.
          OUTPUT_B-STRING = OCCURANCE-STRING.
          OUTPUT_B-CODE_LINE = OCCURANCE-CODE_LINE.
          OUTPUT_B-LINNUM = OCCURANCE-LINNUM.
          APPEND OUTPUT_B.
        ENDLOOP.

* increase number of programs found
        ADD 1 TO PROGCNT.
      ENDIF.
    ENDIF.

* for frontend processing increase the progress indicator
    IF ( ( SY-BATCH EQ SPACE ) AND ( SOURCECNT NE 0 ) ).
      ADD PRCINC_I TO STATUS.
* status saturation
      IF STATUS GT 100.
        STATUS = 100.
      ENDIF.

* refresh the icon of indicator only if STATUS -
LAST_DISPLAYED_STATUS >= 1
      DIFFERENCE = STATUS - LAST_DISPLAYED_STATUS.
      IF DIFFERENCE GE 1 .
```

```
        CALL FUNCTION 'SAPGUI_PROGRESS_INDICATOR'
          EXPORTING
            PERCENTAGE = STATUS
            TEXT       = TEXT-012.
        LAST_DISPLAYED_STATUS = STATUS.
      ENDIF.
    ENDIF.

* check the number of programs found if limit is given
    IF ( NOT LIMIT_P IS INITIAL ) AND ( PROGCNT GE LIMIT_P ).
      EXIT.
    ENDIF.
  ENDLOOP. "LOOP AT NAMES.
ENDFORM.                    "READ_SOURCES

*---------------------------------------------------------------------*
* FORM CHECK_ OCCURANCE
* The form reads source code. It also fills up the internal
* table
* OCCURANCE with string and their positions
*---------------------------------------------------------------------*
FORM CHECK_OCCURANCE.

  DATA: CURR_SY_TABIX TYPE I.

* reading source code line by line
  LOOP AT SOURCE.

* skip comments (lines with first character eq '*')
    IF SKIPC EQ 'X'.
      IF SOURCE+0(1) EQ '*'.
        CONTINUE.
      ENDIF.
    ENDIF.

* searching for string1, string2, ... ,string8 at current line
    CURR_SY_TABIX = SY-TABIX.

    PERFORM SEARCH_STRING USING STRING1 SOURCE CURR_SY_TABIX
STR1REX.
```

```
    PERFORM SEARCH_STRING USING STRING2 SOURCE CURR_SY_TABIX
STR2REX.
    PERFORM SEARCH_STRING USING STRING3 SOURCE CURR_SY_TABIX
STR3REX.
    PERFORM SEARCH_STRING USING STRING4 SOURCE CURR_SY_TABIX
STR4REX.
    PERFORM SEARCH_STRING USING STRING5 SOURCE CURR_SY_TABIX
STR5REX.
    PERFORM SEARCH_STRING USING STRING6 SOURCE CURR_SY_TABIX
STR6REX.
    PERFORM SEARCH_STRING USING STRING7 SOURCE CURR_SY_TABIX
STR7REX.
    PERFORM SEARCH_STRING USING STRING8 SOURCE CURR_SY_TABIX
STR8REX.
  ENDLOOP.
ENDFORM.                    "CHECK_OCCURANCE

*---------------------------------------------------------------------*
* FORM SEARCH_STRING USING P_STRING P_SOURCE P_SY_TABIX
* The routine tries to find string P_STRING in string P_SOURCE
* if P_STRING is not empty. If found, new entry in internal
* table
* OCURRANCE will be created.
* Now it supports regulax expression searching as well.
*---------------------------------------------------------------------*
FORM SEARCH_STRING USING P_STRING P_SOURCE P_SY_TABIX P_REGULAR.
* the string was not entered by user, don't waste time
  IF P_STRING IS INITIAL.
    EXIT.
  ENDIF.

* if the parameter 'skip all comments' is checked then don't
consider
* the part of P_SOURCE which starts by '"'
  IF SKIPC EQ 'X'.
    SPLIT P_SOURCE AT '"' INTO C1 C2.
  ELSE.
    C1 = P_SOURCE.
  ENDIF.

* For non-regular strings use the original searching
  IF P_REGULAR NE 'X'.
    SEARCH C1 FOR P_STRING.
```

```
    IF SY-SUBRC EQ 0.
* string found at current line
      OCCURANCE-STRING = P_STRING.
      OCCURANCE-CODE_LINE = C1.
* code line without leading spaces
      SHIFT OCCURANCE-CODE_LINE LEFT DELETING LEADING SPACE.
      OCCURANCE-LINNUM = P_SY_TABIX.
      OCCURANCE-OFFSET = SY-FDPOS + 1.
      APPEND OCCURANCE.
    ENDIF.
* For regular expressions use this new version of searching
  ELSE.
    DATA: MOFF TYPE I.

    TRANSLATE C1 TO UPPER CASE. "code hase to be upper cased
first
    FIND FIRST OCCURRENCE OF REGEX P_STRING IN C1
    MATCH OFFSET MOFF.

    IF SY-SUBRC EQ 0.
* regular string found at current line
      OCCURANCE-STRING = P_STRING.
      OCCURANCE-CODE_LINE = C1.
* code line without leading spaces
      SHIFT OCCURANCE-CODE_LINE LEFT DELETING LEADING SPACE.
      OCCURANCE-LINNUM = P_SY_TABIX.
      OCCURANCE-OFFSET = MOFF.
      APPEND OCCURANCE.
    ENDIF.
  ENDIF.
ENDFORM.                    "SEARCH_STRING

*---------------------------------------------------------------*
* FORM WRITE_OUTPUT
* Writes out the internal tables OUTPUT_A and OUTPUT_B.
*---------------------------------------------------------------*
FORM WRITE_OUTPUT.

* the user wants to have 'scanned programs list'
  IF REPA EQ 'X'.
```

```
* part A of report
    WRITE: / TEXT-006 COLOR 3 INTENSIFIED ON.
    SKIP 1.
    SORT OUTPUT_A BY NAME.

    LOOP AT OUTPUT_A.
      IF OUTPUT_A-CRITICAL EQ 'X'.
* write out critical message
        WRITE: / OUTPUT_A-NAME COLOR COL_NEGATIVE INTENSIFIED
ON,
                 OUTPUT_A-MESSAGE COLOR COL_NEGATIVE INTENSIFIED
ON.
      ELSE.
* write out not critical message
        WRITE: / OUTPUT_A-NAME,
                 OUTPUT_A-MESSAGE.
      ENDIF.
    ENDLOOP.
    ULINE.
  ELSE.
* but even if the user doesn't want to have 'scanned programs
list'
* inform him about critical messages

    CLEAR AT_LEAST_ONE_CRITICAL.
    LOOP AT OUTPUT_A WHERE CRITICAL EQ 'X'.
      WRITE: / OUTPUT_A-NAME COLOR COL_NEGATIVE INTENSIFIED ON,
               OUTPUT_A-MESSAGE COLOR COL_NEGATIVE INTENSIFIED
ON.
      AT_LEAST_ONE_CRITICAL = 'X'.
    ENDLOOP.
    IF AT_LEAST_ONE_CRITICAL EQ SPACE.
      WRITE: / TEXT-013.
      SKIP 1.
    ENDIF.
  ENDIF.

* part B of report
  WRITE: / TEXT-007 COLOR 3 INTENSIFIED ON.
  SKIP 1.

* header for report B
```

```
  WRITE: /(40) TEXT-008 COLOR COL_HEADING INTENSIFIED ON,
          (11) TEXT-010 COLOR COL_HEADING INTENSIFIED ON,
          (256) TEXT-024 COLOR COL_HEADING INTENSIFIED ON,
          (128) TEXT-009 COLOR COL_HEADING INTENSIFIED ON.

  SORT OUTPUT_B BY NAME STRING LINNUM.

  CLEAR: TOTALPR, TOTALOC.

  LOOP AT OUTPUT_B.

* for new name of program write the semi-heading
    AT NEW NAME.
      SKIP 1.
      WRITE: / OUTPUT_B-NAME UNDER TEXT-008
      COLOR COL_POSITIVE INTENSIFIED ON.
      ADD 1 TO TOTALPR. "1 more program
    ENDAT.

    WRITE: /  OUTPUT_B-NAME UNDER TEXT-008,
           OUTPUT_B-LINNUM UNDER TEXT-010 CENTERED,
           OUTPUT_B-CODE_LINE UNDER TEXT-024,
           OUTPUT_B-STRING UNDER TEXT-009.

* hide the variables NAME and LINNUM for calling the editor
* for showing the code directly
    HIDDEN_NAME = OUTPUT_B-NAME.
    HIDDEN_LINNUM = OUTPUT_B-LINNUM.
    HIDDEN_VALID_LINE = 'X'.

    HIDE HIDDEN_NAME.
    HIDE HIDDEN_LINNUM.
    HIDE HIDDEN_VALID_LINE.

    ADD 1 TO TOTALOC. "1 more occurance

  ENDLOOP.

* write the counted numbers
  TOTALOC_C = TOTALOC.
```

```
  TOTALPR_C = TOTALPR.
  SKIP 2.

* concatenating the message
  CONCATENATE TEXT-015 TOTALOC_C TEXT-016 TOTALPR_C TEXT-017
  INTO MSG SEPARATED BY SPACE.
  WRITE: / MSG COLOR COL_TOTAL.
  CLEAR HIDDEN_VALID_LINE.
ENDFORM.                    "WRITE_OUTPUT

*---------------------------------------------------------------*
* AT LINE-SELECTION
* Code in this event calls editor for showing the code at the
*
* place where occurrence exists
*---------------------------------------------------------------*
AT LINE-SELECTION.

* go on only if valid line was selected
  IF HIDDEN_VALID_LINE EQ 'X'.

* call editor in display mode
    CALL FUNCTION 'EDITOR_PROGRAM'
      EXPORTING
*   APPID             = '  '
       DISPLAY           = 'X'
*   FBNAME            = ' '
       LINE              = HIDDEN_LINNUM
*   MESSAGE           = ' '
*   OFFSET            = '00'
        PROGRAM           = HIDDEN_NAME
*   TOPLINE           = '000000'
*   VARIED            = ' '
*   TRDIR_INF         = ' '
*   STATUS            = ' '
* IMPORTING
*   DYNPRO            =
*   EVENT             =
*   FCODE             =
*   MODULE            =
*   SUBRC             =
```

```
     EXCEPTIONS
       APPLICATION       = 1
       OTHERS            = 2
              .
    IF SY-SUBRC <> 0.
      MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
              WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
    ENDIF.
    CLEAR HIDDEN_VALID_LINE.
  ENDIF.

*---------------------------------------------------------------*
* REGULAR_EXPRESSION_CHECK
* This routine checks all entered regular expressions regarding
* syntax errors. If some error occurs it returns sy-subrc ne 0.
*---------------------------------------------------------------*
FORM REGULAR_EXPRESSION_CHECK.
  DATA: AUX_TEXT TYPE STRING VALUE 'abcdefgh'.
  IF ( STRING1 IS NOT INITIAL ) AND ( STR1REX EQ 'X' ).
    TRY.
        FIND FIRST OCCURRENCE OF REGEX STRING1 IN AUX_TEXT.
      CATCH CX_ROOT.
        REG_ERROR = 'X'.
        EXIT.
    ENDTRY.
  ENDIF.

  IF ( STRING2 IS NOT INITIAL ) AND ( STR2REX EQ 'X' ).
    TRY.
        FIND FIRST OCCURRENCE OF REGEX STRING2 IN AUX_TEXT.
      CATCH CX_ROOT.
        REG_ERROR = 'X'.
        EXIT.
    ENDTRY.
  ENDIF.

  IF ( STRING3 IS NOT INITIAL ) AND ( STR3REX EQ 'X' ).
    TRY.
        FIND FIRST OCCURRENCE OF REGEX STRING3 IN AUX_TEXT.
      CATCH CX_ROOT.
```

```
      REG_ERROR = 'X'.
      EXIT.
  ENDTRY.
ENDIF.

IF ( STRING4 IS NOT INITIAL ) AND ( STR4REX EQ 'X' ).
  TRY.
      FIND FIRST OCCURRENCE OF REGEX STRING4 IN AUX_TEXT.
    CATCH CX_ROOT.
      REG_ERROR = 'X'.
      EXIT.
  ENDTRY.
ENDIF.

IF ( STRING5 IS NOT INITIAL ) AND ( STR5REX EQ 'X' ).
  TRY.
      FIND FIRST OCCURRENCE OF REGEX STRING5 IN AUX_TEXT.
    CATCH CX_ROOT.
      REG_ERROR = 'X'.
      EXIT.
  ENDTRY.
ENDIF.

IF ( STRING6 IS NOT INITIAL ) AND ( STR6REX EQ 'X' ).
  TRY.
      FIND FIRST OCCURRENCE OF REGEX STRING6 IN AUX_TEXT.
    CATCH CX_ROOT.
      REG_ERROR = 'X'.
      EXIT.
  ENDTRY.
ENDIF.

IF ( STRING7 IS NOT INITIAL ) AND ( STR7REX EQ 'X' ).
  TRY.
      FIND FIRST OCCURRENCE OF REGEX STRING7 IN AUX_TEXT.
    CATCH CX_ROOT.
      REG_ERROR = 'X'.
      EXIT.
  ENDTRY.
ENDIF.
```

```
  IF ( STRING8 IS NOT INITIAL ) AND ( STR8REX EQ 'X' ).
    TRY.
        FIND FIRST OCCURRENCE OF REGEX STRING8 IN AUX_TEXT.
      CATCH CX_ROOT.
        REG_ERROR = 'X'.
        EXIT.
    ENDTRY.
  ENDIF.
ENDFORM.                    "regular_expression_check
```