HAAGA-HELIA
ammattikorkeakoulu

CART – Continuos Automated Regression Testing tool for QPR

Juho Vuorio

Opinnäytetyö

Tietojenkäsittely

2010

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

Opinnäytetyön mahdollisen esityksen päivämäärä

Koulutusohjelma

Tekijät Juho Vuorio.	**Ryhmä tai aloitusvuosi** 2004
Opinnäytetyön nimi CART – Continuos Automated Regression Tool for QPR	**Sivu- ja liitesivumäärä** 27 + 238
Ohjaaja tai ohjaajat Jyri Partanen	

Työssä toteutettiin automaattinen regressiotestaus työkalu QPR Software Oyj:lle. CART on pääsääntöisesti tarkoitettu sovelluskehittäjien käyttöön, jotta he pystyisivät tehokkaammin testaamaan koodimuutosten mahdollisia regressiovaikutuksia. Varsin laajan koodipohjan takia, regression aiheuttamat ongelmat ovat varteenotettava ongelma. Samoja koodinosia saatetaan käyttää todella monessa eri osassa ohjelmistoa. Kehittäjän on lähes mahdotonta varmistaa ilman testejä, ettei koodiin tehty muutos ole aihettanut ei-haluttua toiminnallisuutta ohjelmiston toiseen osaan.

CART on tehty regressiomuutosten löytämiseksi. CART sisältää QPR ProcessGuide (PG) ja QPR Scorecard (SC) testejä. CART käyttää hyväkseen kyseisten ohjelmien API- rajapintaa. Muut QPR:n tuotteen on jätetty CART:n ulkopuolelle, koska niiden API – rajapinta ei ole tarpeeksi monipuolinen, jotta niiden avulla voitaisiin tehdä kattavia testejä.

Käyttäjä määrittää CART:lle, mitkä testit halutaan ajaa. Tämän jälkeen CART siirtää testikannan käyttäjän omalle koneelle käyttäen avuksi QPR TransferToolia. Kun kanta on siirretty, CART käynnistää tarvittavat serverit. Tämän jälkeen CART käynnistää SC:n tai PG:n suorittaakseen testit. Testien jälkeen tulokset luetaan CART logeista ja tulokset kerrotaan käyttäjälle.

Työ on toteutettu Visual Basicilla ja testiskriptit vbscriptilla.

Ohjelman tavoitteita ei saavutettu lähinnä huonon ohjeistuksen ja epäonnistuneen käyttöönoton vuoksi

Asiasanat
Regressiotestaus:
Testausautomaatio
QPR Scorecard
QPR Processguide

HAAGA-HELIA
ammattikorkeakoulu

Abstract

Date of presentation

Degree programme

Authors Juho Vuorio	**Group or year of entry** 2004
The title of thesis CART – Continuos Automated Regression Tool for QPR	**Number of pages and appendices** 27+238
Supervisors Jyri Partanen	

The purpose of this job is to create a regression testing tool for QPR Software Plc. Because of the quite vast and old codebase, regression issues are a bit of a problem. The plan is that this regression testing tool would give an opportunity for the developers to perform regression tests quite easily and fast before they commit their code changes to the common code base.

If the developers would find the problems before the commit, the overall quality would probably improve. For this purpose, CART was implemented.

CART includes testscripts for QPR Scorecard and QPR ProcessGuide. CART uses QPR API for the tests and the other QPR applications do not have API or the API is too simple. This is the reason why the other applications are excluded from CART.

User decides what tests are executed with CART. After this, CART automatically transfers the test database to the users computer with QPR TransferTool. After the transfer, CART starts the servers and executes the chosen tests with QPR Scorecard Client or with QPR Processguide Client. After all the test are executed, CART returns the tests results to the user.

The program is developed with Visual Basic and the test scripts are developed with vbscript

The goals set for CART were not met. The support material was not detailed enough and the deployment was not successful.

Key words
Regression testing
Test automation
QPR Scorecard
QPR ProcessGuide

Sisällys

1 Johdanto

1.1 Yleisjohdanto

QPR Software Oyj on yritys, jonka tavoitteena on parantaa asiakkaidensa prosesseja sekä parantaa suorituskykyä. QPR Software Oyj:n ratkaisuja käyttää noin 1500 yritystä ympäri maailman ja tuotteen on käännetty yli 20:lle kielelle.

QPR Software Oyj tarjoaa seuraavanlaisia ratkaisuja: Toimintajärjestelmä, Prosessijohtamisen ratkaisut, Tuloskorttiratkaisut, Liiketoiminnan ohjausjärjestelmä, Laatujärjestelmät, Riskienhallintajärjestelmät, QPR Julkishallinnon Johtamisjärjestelmä sekä Työkulkujen automatisointi. Kyseiset ratkaisut pohjautuvat QPR:n ohjelmistoille. QPR Scorecard strategisiin tavoitteisiin, QPR Processguide prosessien suunnitteluun, QPR Portal tiedon jakamiseen ja seurantaan, QPR Factview toteuman analysointiin ja QPR Workflow toimintojen automatisointiin.

QPR:n eri ohjelmistot ovat suunniteltu toimimaan yhdessä ja erillisinä ohjelmina. Tämän sekä laajan ja osittain vanhan koodipohjan vuoksi, uusia muutoksia tehdessä tiettyyn ohjelmiston osaan, on vaikea huomioida mahdolliset vaikutukset toisiin osiin. Kehittäjä pystyy päättelemään mahdollisia vaikutuksia, mutta niiden manuaaliseen testaamiseen saattaisi kulua erittäin paljon aikaa.

Tämän työn tavoitteena oli toteuttaa automaattinen testaustyökalu, jonka avulla voidaan testata nopeasti ja varsin kattavasti ohjelmien perustoiminnallisuuksia. Työkalun tavoitteena on parantaa ohjelmien laatua havaitsemalla virheet mahdollisimman aikaisessa vaiheessa.

Continuous Automated Regression Testing tool (CART) on suunniteltu helppokäyttöiseksi testaustyökaluiksi pääsääntöisesti kehittäjille, mutta myös muillekkin. Työkalun avulla kehittäjät saavat testattua tekemänsä muutoksen vaikutukset nopeasti ja varsin laajasti. CART ei pysty kattamaan kaikkia mahdollisia käyttötapauksia rajallisten API – rajapinnan toiminnallisuuksien vuoksi, mutta peruskäyttötapaukset API - rajapinnan kautta voidaan käsitellään varsin kattavasti. CART testeihin kuuluu ainoastaan QPR Scorecardin ja QPR Processguiden testaaminen. Tämä johtuu siitä, että muiden tuotteiden API – rajapinta ei ole tarpeeksi kehittynyt, että sen avulla voitaisiin testata tuotteita tehokkaasti.

CART testit on tarkoitettu suoritettavaksi ennen muutosten lisäämistä yhteiseen koodiin. Tämän avulla kehittäjä voi havaita ja korjata ainakin osan mahdollisista virheistä ennen uuden koodin lisäämistä yhteiseen koodiin. CART testien suorittaminen myös vaatii sen, että kehittäjä on myös kääntänyt koodin, joten tämä myös estää virheet, jotka estävät testausbuildien kääntymisen. Vaikka kääntämisen estäviä virheitä sattuu varsin harvoin, niitä tapahtuu silloin tällöin ja kyseisten virheiden korjaamiseen kuluu aikaa turhaan.

1.2 Tavoitteet

Työn tavoitteena on parantaa QPR Scoracardin ja QPR Processguiden laatua. Tavoite pyritään saavuttamaan saamalla virheet kiinni ennenkuin ne lisätään varsinaiseen yhteiseen koodiin. Tämä pyritään saavuttamaan parantamalla ohjelmistokehittäjien testaamista.

Tavoitteen saavuttamiseksi, luodaan uusi testausohjelma Continuos Automated Regression Tool ”CART”. CART on helppokäyttöinen testiohjelma, jolla kehittäjät voivat testata omien muutoksien regressiovaikutuksia ohjelmien muihin osiin.

1.3 Tehtävänasettelu

Työssä tutkitaan automaatisen testauksen vaikutusta mahdolista vaikutusta QPR Processguiden ja QPR Scorecardin laatuun. Tavoitteena on saada parannettua QPR Scorecard sekä QPR Processguide ohjelmien laatua kehittämällä automaattinen testaustyökalu.

Saavutettu lopputulos päätellään siitä, kuinka monta virhettä on CART:n avulla löydetty verrattuna muilla menetelmillä löydettyihin virheisiin.

QPR on pystynyt aikaisemmin parantamaan tuotteidensa laatua automaattisten testaustyökalujen avulla. Hyvänä esimerkkinä toimii suorituskyvyn parantaminen QPR Portalissa. Tämä toteutettiin simuloimalla useita QPR Portal käyttäjiä samanaikaisesti työkalun avulla. Testeissä löydettiin ongelmakohdat ja ne korjattiin. Tähän samaan perustuu olettamus, että uusi testaustyökalu voisi parantaa QPR tuotteiden laatua.

1.4 Käsitteet

QPR Scorecard. QPR Scorecard on tuloskortti – ohjelma. Tämä ohjelma on toinen QPR:n tuotteista, jota testataan CART:n avulla. Testauksen kohteena on mm. uusien tuloskorttien luominen, mittarien luominen, arvojen syöttäminen ja käyttöoikeudet.

QPR Processguide. QPR Processguide on mallintamistyökalu. Pääsääntöisesti se on tarkoitettu erilaisten prosessien mallintamiseen. Tämä on toinen QPR:n tuotteista, jota testataan CART:n avulla. Testauksen kohteena on mm. mallien luominen ja poistaminen, elementtien luominen sekä käyttöoikeudet.

Regressiotestaus. Testaustyyppi, jossa testataan ohjelmiston osia, joihin muutokset eivät ole varsinaisesti kohdistuneet. Tarkoituksena on varmistaa, että tehdyt muutokset eivät ole aiheuttaneet ei-toivottuja vaikutuksia muihin ohjelmiston osioihin. (Perry, W. 2000a)

Testausautomaatio. Testaus suoritetaan työkalujen avulla automaattisesti. Tarkoitus on säästää resursseja sekä suorittaa testejä, joita ei ole mahdollista tehdä manuaalisesti, kuten suorituskykytestit. (Tian, J. 2005d)

Mustalaatikkotestaus. Testaa toiminnallisuutta. Tarkoituksena simuloida ohjelman varsinaista käyttämistä. (Perry, W.E. 2000b)

Lasilaatikkotestaus. Staattista testausta. Varsinaista koodia ei suoriteta. Tarkoituksena yrittää käydä läpi ohjelman osioita, joita ei välttämättä voida mustalaatikkotestauksella käydä läpi. Koodikatselmoinnit. (Perry, W.E. 2000b)

Harmaalaatikkotestaus. Mustalaatikko ja lasilaatikkotestauksen yhdistämistä. Testejä suoritettaessa ja suunniteltaessa käytetään koodikatselmointia avuksi, jotta saadaan mahdollisimman laaja kattavuus testeille.

Laadunhallinta. Pitää sisällään kaiken mitä laatuun liittyy. Määrittelee testauksen toimintaohjeet ja periaatteet. (Van Veenendal, E. 2007a)

2 Viitekehys

2.1 Laadun määritteleminen

Laadulla voidaan tarkoittaa montaa eri asiaa. Se voi tarkoittaa mm. luotettavuutta, käytettävyyttä tai ylläpidettävyyttä. Osa laadun määritelmistä voidaan mitata tarkkaan, kuten tehokkuus, mutta esimerkiksi käytettävyyttä on varsin hankala mitata, koska suuri osa on yksilön omaa arviota.

ISO-9126 (ISO, 2001) määrittelee hierarkisen kehyksen laadulle. Laatu on siinä jaettu kuuten eri osioon: Toiminnallisuuteen, luotettavuuteen tehokkuuteen, käytettävyyteen, ylläpidettävyyteen sekä siirrettävyyteen.

Toiminnallisuuden tarkoitus on täyttää joko suorasti tai epäsuorasti toiminnallisuudelle asetetut tavoitteet. Toiminnallisuuden tuntomerkkeinä ovat soveltuvuus (suitability), tarkkuus (accuracy), yhteentoimivuus (interoperability) ja turvallisuus (security).

Luotettavuudella tarkoitetaan sitä, että ohjelma pitää tietyissä olosuhteissa suorituskykynsä samalla tasolla määritetyn ajan. Luotettavuuden tuntomerkkeinä ovat kypsyys (maturity), virheensietokyky (fault tolerance) sekä palautettavuus (recoverability)

Tehokkuus määritetään niin, että se on suorituskyvyn taso tietyillä annetuilla resursseilla, tietyissä olosuhteissa. Tehokkuuden tuntomerkkejä ovat ohjelman käyttäytyminen tietyn ajan sisällä (time behaviour) sekä resurssien käyttö (resource behaviour).

Käytettävyydellä tarkoitetaan sitä, kuinka paljon resursseja/koulutusta ohjelman käyttäminen vaatii. Käytettävyyttä tosin on varsin mahdoton mitata, koska siihen kuuluu myös yksilön oma mielipide. Mikä joidenkin mielestä on helppo käyttää, on toiselle varsin vaikeaselkoinen käyttää. Käytettävyyden tuntomerkkejä ovat ymmärrettävyys (understandability), käyttöliittymän hallinnan opettelun haastavuus (learnability) sekä operoitavuus (operability).

Ylläpidettävyydellä tarkoitetaan sitä, kuinka vaikeaa on tehdä muutoksia ohjelmaan. Näitä muutoksia ovat mm. päivitykset. Tuntomerkkeinä ovat analysoitavuus (analyzability), vaihtuvuus (changeability), ohjelman vakaus (stability) sekä testattavuus (testability).

Siirretävyydellä tarkoitetaan sitä, kuinka helppoa ohjelma on siirtää ympäristöstä toiseen. Tuntomerkkeinä ovat muokkautuvuus (adabtability), asennuttavuus (installability), noudattaminen (conformance) sekä korvattavuus (replaceability). (Tian, J. 2005b)

2.2 Laadunhallinta

Laadunhallinta eli Quality management pitää sisällään kaikki toiminnot organisaation sisällä, jotka liittyvät laatuun. Laadunhallinta ei siis pelkästään keskity ohjelmiston laatuun vaan myös mm. työntekijöiden pätevyyteen ja prosesseihin. Lähtökohtana on usein quality policy, jossa määritellään laatuun liittyvät toimintaperiaatteet sekä toimintaohjeet. Quality managementtiin kuuluu myös laatutason suunnittelu (quality objectives ja quality planning), laadunohjaus (quality control, QC), laadunvarmistus (quality assurance, QA) sekä laadun parantaminen (quality improvement). Myös QPR on tehnyt oman quality policyn, jossa määritellään mm. julkaisujen vaadittu laatutaso, tarvittavat testit sekä testausautomaation osa testauksessa. (Van Veenendal, E. 2007a)

Laadunohjauksessa hallinnoidaan varsinaista prosessia ja tehdään päätöksiä perustuen toimintaperiaatteisiin sekä toimintaohjeisiin. Laadunohjauksessa tehdään päätöksiä mm. mitä tehdään ohjelmistoon jääneille virheille. Laadunohjaus pohjautuu pitkälti laadunvarmistuksesta saatuihin tietoihin. QPR:n quality policy ottaa kantaa laadunohjaukseen varsin tarkasti. Julkaisuissa ei saa olla mukana tiedettyjä suuren riskin tai vaikutusalueen avoimia virheitä. Tämän periaatteen mukaan virheet korjataan eli käytännössä ohjataan laatua. (Tian, J. 2005c)

Laadunvarmistuksen tavoitteena on luoda varmuutta laatutasosta ja varmistaa, että laadulle asetetut kriteerit täyttyvät. Laadunvarmistuksen avulla saadaan myös tietoa, mitä mahdollisia virheitä ohjelmisto vielä sisältää ja minkälaisia vaikutuksia virheestä seuraa. QPR suorittaa jokaiselle ohjelmistoon tehdylle muutokselle laadunvarmistuksen eli käytännössä testaa muutetun toiminnallisuuden. (Van Veenendal, E. 2007a; Tian,J 2005c)

Laadunvarmistus on varsin prosessikeskeistä työskentelyä. Tohtori W. Edwards Demingin kehittämä Scewhart Cycle kuvaa hyvin laadunvarmistusta. Siihen kuuluu neljä eri askelta: suunnittele (plan), tee (do), tarkista (check) ja toimi (act). Suunnittelussa suunnitellaan testit ja tavoitteet. Seuraavaksi tehdään testit. Tarkistuksessa arvioidaan ja seurataan tuloksia ja arvioidaan niitä määritettyjä tavoitteita silmälläpitäen. Lopuksi toimitaan. QPR:llä tämä nämä askeleet ovat testien suunnittelu muutettujen osioiden perusteella, testien suorittaminen,

testitulosten arvioiminen ja testitulosten perusteella, hyväksytään muutos tai lähetetään muutos uudelleen korjattavaksi. (Wikipedia; American Society of Quality)

Esimerkkinä Shewhartin syklistä: Ohjelmistoon on tehty muutoksia. Testaajan on suunniteltava muutoksen perusteella testit, jotka pohjautuvat muutoksen haluttuun toiminnallisuuteen. Tämän jälkeen testaaja suorittaa suunnitellut testit. Testien tulokset analysoidaan ja tehdään päätös, onko toiminnallisuus oikein, pitääkö jotain korjata vai pitääkö jotain vielä testata.

2.3 Testaustekniikoita

Testaustekniikat voidaan karkeasti jakaa kolmeen ryhmään: Lasilaatikkotestaus, mustalaatikkotestaus sekä harmaalaatikkotestaus, joka on yhdistelmä kahdesta aikaisemmasta testaustekniikasta.

2.3.1 Lasilaatikkotestaus

Lasilaatikkotestaus (White-box testing) joka tunnetaan myös nimellä rakenteellinen testaus (structural testing), on testausta, jossa testitapaukset perustuvat ohjelman rakenteeseen. Lasilaatikkotestauksella testataan ohjelmasta osioita, joita ei välttämättä päädyttäisi suorittamaan pelkällä mustalaatikkotestauksella. Lasilaatikkotestauksessa ei välttämättä suoriteta yhtään riviä koodia, vaan testaus voidaan tehdä pelkästään katselmoimalla koodia eli staattisella testaamisella. (Perry, W.E. 2000b)

Lasilaatikkotestauksen hyvinä puolina on se, että sen avulla voidaan suorittaa ohjelmasta sellaisia polkuja tai lauseita, mitä ei normaalikäytössä välttämättä suoritettaisi. Ohjelman toiminnallisuuden loogisuutta voidaan myös testata eli suoritetaanko ohjelman polut oikeassa järjesteyksessä ja suoritetaanko kaikki tarvittavat polut. Haittoina voidaan pitää sitä, että välttämättä ei voida määritellä, onko asiakkaan määrittelemät tavoitteet saavutettu vai ei, koska välttämättä koodia ei varsinaisesti suoriteta. Myöskään käyttötapaukset eivät välttämättä ole kovin realistisia, joten varsinaisen käytön kannalta testaus ei välttämättä kerro yhtään mitään. (Perry, W.E. 2000b)

Lasilaatikkotestaukseen kuuluu esimerkiksi seuraavia testaustekniikoita. Lausekattavuuden testaaminen (statement coverage), haarakattavuuden testaaminen (branch coverage), ehtojen

kattavuuden testaaminen (condition coverage) sekä polkujen kattavuuden testaaminen (path coverage).

2.3.2 Mustalaatikkotestaus

Mustalaatikkotestaus (Black-box testing) eli toiminnallisuuden testaus (functional testing) testaa pelkästään ohjelman toiminnallisuutta eikä ota kantaa mitä koodin osia ohjelma suorittaa, jotta haluttu tulos saadaan esille. Mustalaatikkotestauksen avulla voidaan myös testata erilaisten dokumenttien paikkaansa pitävyyttä. Esimerkiksi koulutusmateriaalit on hyvä testata mustalaatikkotestausta hyväksikäyttäen. (Perry, W.E. 2000b)

Mustalaatikkotestauksen etuna on luonnollisesti se, että se simuloi normaalia käyttöä. Huonona puolena voidaan pitää sitä, että ei voida varmistaa, suoritetaanko kaikkia ohjelman osia ja polkuja. Tämä johtaa siihen, että käytännössä 100 % testauskattavuus on lähes mahdotonta saavuttaa mustalaatikkotestauksella. (Perry, W.E. 2000b)

Mustalaatikkotestauksen tekniikoita ovat mm. raja-arvojen testaaminen (boundary values), virheiden arvaaminen eli ad hoc testaaminen sekä tilojen testaaminen (state testing).

2.3.3 Harmaalaatikkotestaus

Harmaalaatikkotestaus eli Gray box testing yhdistelee tekniikoita sekä mustalaatikkotestauksesta että valkolaatikkotestauksesta. Tavoitteena on saada mahdollisimman kattavat testit. Yhdistelemällä tekniikoita, voidaan saada selville, mitä komentoja ei esimerkiksi saatu suoritettua testaamisella. Koodia analysoimalla voidaan luoda uusia testitapauksia, joiden avulla kyseiset komennot suoritetaan ja testien kattavuutta saadaan parannettua.

2.4 Regressiotestaus

Yksi yleinen ongelma sovelluskehityksessä on, kun ohjelman jonkin osan toiminnallisuutta tai koodia muutetaan, saattaa se aiheuttaa ongelmia johonkin toiseen ohjelman osaan. Voi olla että uusi koodi sisältää uusia parametreja tai muuta tietoa, mitä vanha osa ei pysty käyttämään tai uusi koodi on vain virheellistä. Näiden ongelmien huomaamiseksi on kehitetty regressiotestaus. QPR:n tapauksessa regressiotestauksella on varsin tärkeä osa testausta, koska

koodipohja on vanha ja laaja. Kehittäjien on erittäin vaikea ottaa huomioon muutosten aiheuttamia vaikutuksia muissa ohjelmiston osissa.

Regressiotestausessa testataan ei-muutettua osaa ohjelmasta, jotta varmistutaan, ettei uusi muutos ole rikkonut vanhaa toiminnallisuutta. Tämä toteutetaan normaalisti suorittamalla jo suoritettuja testitapauksia, jotta voidaan tarkistaa, että testitapausten tulokset pysyvät muuttumattomina. Regressiotestauksella voidaan myös varmistaa, että dokumentointi ja testitapaukset ovat edelleen paikkaansa pitäviä. (Perry, W. 2000a)

Regressiotestaus on yleensä varsin aikaa vievää, jos sitä ei ole automatisoitu. Tämän takia onkin varsin tarkkaan harkittava, mitä osioita testataan regressiotestauksella. Yleisesti ottaen regressiotestausta kannattaa käyttää niissä tapauksissa, jos on tiedossa että muutos aiheuttaa varsin suuren riskin toiminnallisuuteen. (Perry, W. 2000a)

2.5 Testausautomaatio

Testausautomaation tarkoitus on automatisoida joitain manuaalisia testejä testaustyökalujen avulla. Testausautomaation tarve on suuri, koska manuaalinen testaus alusta loppuun voi olla pitkäveteistä sekä taipuvainen virheisiin. Esimerkiksi QPR:llä on yli 2000 manuaalisen regressiotestitapauksen kokoelma, joiden kaikkien suorittamiseen kuluu arviolta noin 300 tuntia. Testausautomaation avulla voitaisiin vähentää testien suorittamiseen tarvittavaa aikaa ilman, että suoritettavia testitapauksia tarvitsisi vähentää. (Tian, J. 2005a)

Jotta testausautomaatiolla voitaisiin säästää ihmisiä varsin rankalta ja uuvuttavalta manuaaliselta testaamiselta ja samalla parantaa yleisesti laatua, on aluksi selvitettävä mitä voidaan automatisoida ja mitä kannattaa automatisoida. Tämän jälkeen on otettava selvää, minkälaisia testaustyökaluja on markkinoilla vai tarvitseeko tehdä oma testaustyökalu, jotta tavoitteet saavutetaan. Valintaan vaikuttaa myös olennaisesti kustannukset eli onko halvempaa ostaa työkalu vai kehittää oma testaustykalu. On hyvä myös ottaa huomioon, että pelkkä kehittäminen tai ostaminen ei riitä. Käyttäjien kouluttaminen on myös otettava huomioon.

Testausautomaatiolla voidaan myös toteuttaa testejä, joita ei olisi mahdollista toteuttaa manuaalisesti. Hyvänä esimerkkinä voidaan pitää suorituskykytestausta. Erilaisilla testaustyökaluilla voidaan simuloida monen sadan tai tuhannen käyttäjän kuormaa mm. verkkosovellukselle. Tämänkaltaisten testien suorittaminen manuaalisesti olisi erittäin työlästä ja resursseja kuluttavaa ja monissa tapauksissa ei ole mahdollista saada käyttöön vaadittavaa

määrää resursseja, jotta testaus olisi todenmukaista. On erittäin tärkeää tietää miten ohjelmistot suoriutuvat, kun käyttäjiä on erittäin monta ja ohjelmiston resurssit toimivat maksimikapasiteetillaan, jotta voidaan välttää tilanteita, joissa kuormitus aiheuttaa ongelmia ohjelman toiminnallisuuteen. (Tian, J. 2005d)

Kaikenkaikkiaan testausautomaatio voidaan jakaa kolmeen osioon: Testien suunnittelu ja valmistelut, testien suorittaminen ja tavoitteet sekä testitulosten analysointi ja jatkotoimenpiteet. (Tian, J. 2005a)

2.5.1 Testien suunnittelu ja valmistelut

Testejä suunniteltaessa on asetettava tavoitteet testeille eli mitä yritämme saavuttaa kyseisellä testauksella. Kun tavoitteet ovat määritelty, voidaan alkaa miettimään miten ja millä keinoilla tavoitteet saavutetaan. QPR:llä tavoitteet ovat käytännössä uuden muutoksen toiminnallisuuden varmistaminen ja varmistuminen siitä, ettei muutos ole aiheuttanut regressiovaikutuksia. (Tian, J. 2005a)

Kun suunnitelmat ovat valmiit, voidaan aloittaa varsinaisten testien valmistelut. Valmisteluihin kuuluvat erilaisten testausmallien rakentaminen, yksittäisten testitapausten suunnittelu sekä koko testisarjojen valmistelu. QPR:llä normaalisti testisarjoja tehdään vain isommille muutoksille tai eri julkaisujen hyväksymistestien yhteydessä. Quality policy määrittelee QPR:llä, mitkä testit on suoritettava eri julkaisujen yhteydessä. (Tian, J. 2005a)

2.5.2 Testien suorittaminen ja tavoitteet

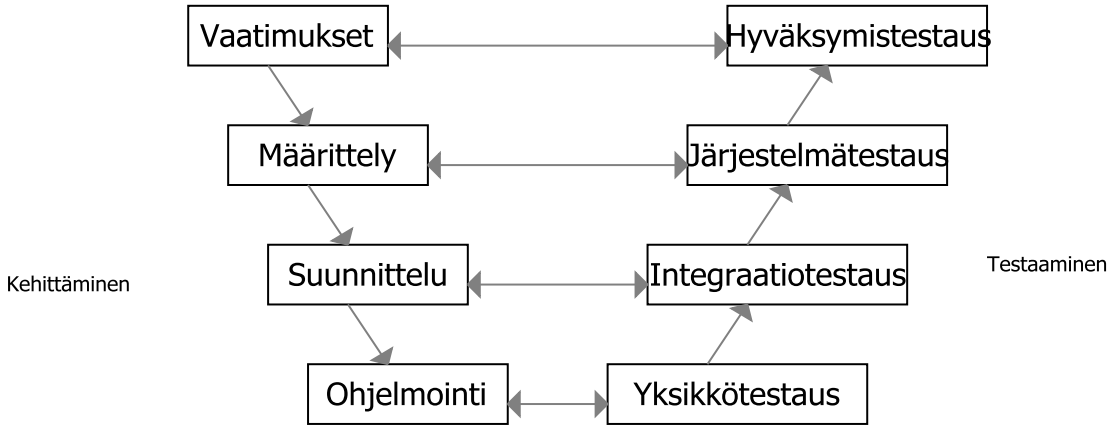
Testien suorittamiseen on arvioitava käytettävä aika ja resurssit. QPR:llä arviointi on myös erityisen tärkeää, koska ohjelma on laaja ja testaamiseen on rajallisesti resursseja käytettävissä. Valitaan resurssien mukaan suoritettavat testit, suoritetaan ne ja arvioidaan tulokset ja mahdolliset järjestelmän ongelmat. Järjestelmän ongelmat tässä tapauksessa voivat myös tarkoittaa testityökalun ongelmia eikä pelkästään tietokoneen järjestelmän ongelmia. (Tian, J. 2005a)

2.5.3 Testitulosten analysointi ja jatkotoimenpiteet

Kun testien tulokset on saatu, aloitetaan tulosten analysointi. Analysoinnissa tarkistetaan mahdollisia ongelmia sekä tehdään päätös mahdollisten virheiden korjaamisesta sekä päätös,

mitä testejä ajetaan mahdollisten korjausten jälkeen. Kuten kappaleessa 2.2 mainittiin, QPR:n quality policy vaikuttaa siihen, mitä virheitä korjataan sekä siihen, mitä testejä on suoritettava. (Tian, J. 2005a)

2.6 Testaaminen eri vaiheissa



Kuva .1 Testauksen V-malli

Oheisessa testauksen V – mallissa kuvataan testauksen eri vaiheita ohjelmistokehityksessä. Mallin vasemmalla puolella on käytännössä ohjelman varsinainen kehittäminen ja suunnittelu eli määritellään, mitä ohjelmalta halutaan. Samalla kun ohjelmaa suunnitellaan, on myös aloitettava testaamisen suunnittelu. Mallin oikealla puolella vain suoritetaan testit, jotka on määritelty suoritettavaksi kehittämisen aikana. Varsinkin hyväksymistestauksen suunnittelu on oltava kunnossa, koska muuten on erittäin hankalaa todistaa asiakkaalle, että kaikki halutut toiminnallisuudet löytyvät ohjelmasta. Testaamisen suunnittelu myös auttaa osaltaan kehittämisen suunnan pitämisessä eli varmistetaan, että tavoitteet saavutetaan.

Yksikkötestaus eli Unit testing tai Component testing, on yksittäisen ohjelman komponentin testaamisen. Käytännössä tämän testausvaiheen suorittaa aina kehittäjä. Yksikkötestauksessa voidaan käyttää sekä mustalaatikkotestausta että lasilaatikkotestausta, riippuen tapauksesta. (OAMK; Van Veenendal, E. 2007b)

Integraatiotestauksella, eli component integration testing, tarkoitetaan tässä kuviossa komponenttien välistä integraatiotestaamista. Tarkoituksena on löytää virheitä komponenttien rajapintojen välisestä kommunikaatiosta sekä varmistaa, että komponenttien välinen kommunikaatio toimii odotetulla tavalla.

Komponenttien integraatiotestausta voidaan lähestyä kahdella eri tavalla: ns. ”Big Bang” – integraatio tai inkrementaalinen integraatio, jossa käytetään avuksi erilaisia ajureita tai muita testikomponentteja. ”Big Bang” – integraatiossa, kaikki eri komponentit käytännössä testataan samalla kertaa eikä eri osissa. Tämän tekniikan ehdottomana hyötynä on nopeus, koska erillisiä ajureita ei tarvita. Haittana on se, että voi olla erittäin työlästä ja vaikeaa löytää mahdollisen virheen aiheuttanut komponentti.

Inkrementaalisessa integaatiossa on kaksi eri tekniikkaa: Bottom-Up tai Top – Down. Bottom – up tekniikassa lähdetään liikkeelle alemman tason komponenteista ja siirrytään sieltä ylöspäin ylemmän tason komponentteihin, kunnes kaikki komponentit ovat testattu. Top – Down tekniikassa tehdään juuri päinvastoin, eli päätason komponentit testataan ensin ja siirrytään alempien tasojen testaamiseen, kunnes kaikki tasot ovat testattu. (OAMK; Van Veenendal, E. 2007b)

Järjestelmätestauksessa eli system testingissa tarkoituksena on varmistaa, että ohjelma täyttää sille asetetut vaatimukset. Järjestelmätestauksessa testataan myös ei-toiminnallisia osia ohjelmasta, kuten käytettävyyttä ja kuormitusta. Järjestelmätestauksesta tulisi vastata joku muu kuin kehittäjä, koska usein kehittäjillä on taipumus pyrkiä todistamaan vain, että ohjelma oikeasti toimii. Suurin osa testausautomaatiosta tehdään yleensä tässä vaiheessa. Koska QPR:n ohjelmat ovat varsin laajoja, käytetään usein apuna myös kehittäjiä tässä testauksen vaiheessa. Kehittäjien tieto kyseisestä alueesta auttaa suunnittelemaan kattavampia testejä ja tietoa, mitkä osat on syytä testata tarkemmin. Varsinkin regressiotestauksen suunnittelussa, tästä on suurta hyötyä. (OAMK; Van Veenendal, E. 2007b)

Melko usein testauksen pääpaino on järjestelmätestauksessa, koska silloin ohjelman kaikki osat pitäisi olla toiminnassa. Tässä vaiheessa myös usein löydetään ja korjataan erinäisiä virheitä mahdollisesti varsin paljon ja siitä voi aiheutua ei-toivottuja muutoksia muihin ohjelman osa-alueisiin, joten regressiotestaus on varsin suuri myös järjestelmätestausta. (OAMK; Van Veenendal, E. 2007b)

Hyväksymistestauksessa eli acceptance testingissa, joko hyväksytään tai hylätään tuotos. Hyväksymistestauksessa käydään läpi, että ohjelma täyttää asiakkaan kanssa tehdyt vaatimukset. Usein tämän testausvaiheen suorittaa joko asiakas itse tai ainakin asiakkaan olisi hyvä olla itse mukana prosessissa varsin läheisesti. Usein kun asiakas on tilannut ohjelman, määritellään jo vaatimukset odotetusta lopputuloksesta. Näitä määrityksiä käytetään hyväksi

hyväksymistestauksessa ja pyritään todistamaan asiakkaalle, että vaatimukset ovat totetutettu. QPR:n tapauksessa hyväksymistestaus tehdään pitkälti samalla järjestelmätestauksen yhteydessä. QPR käyttää ”sisäistä asiakasta” eli tuotepäällikköjä, kun kehitetään uusia toiminnallisuuksia. Eräänlainen hyväksymistestaus suoritetaan jo ennen järjestelmätestausta, jolloin kehittäjä esittää toiminnallisuuden tuotepäälliköille.
(OAMK; Van Veenendal, E. 2007b)

Kirjassaan ”Effective methods for Software Testing”, William E. Perry lokeroi karkeasti eri testausvaiheiden suorittajat seuraavasti: Yksikkötestauksen suorittaa kehittäjä. Integraatiotestauksen voi suorittaa sekä kehittäjä että käyttäjä. Järjestelmätestauksen suorittaa kehittäjä käyttäjien avustuksella. Hyväksymistestauksen suorittaa käyttäjä. Kyseisessä lokeroinnissa tosin ongelman aiheuttaa se, ettei Perry varsinaisesti erottele kehittäjiä ja testaajia. Yleisesti ottaen, järjestelmätestauksen suorittaa lähinnä testaaja kehittäjän mahdollisella avustuksella.
(Perry, W.E. 2000b)

2.7 Balanced Scorecard

Balanced scorecardin eli tuloskorttien ideana on antaa yrityksille mahdollisuus seurata ja mitata taloudellista kehitystä kolmella eri osa-alueella. Osa-alueet ovat yrityksen suhteet omiin asiakkaisiinsa, yrityksen omat avainprosessit ja oppiminen ja kasvu.

Tarkoituksena on lisätä tuloskorttien avulla suorituskykymittareita, joita verrataan varsinaiseen tulokseen. Näiden avulla voidaan helposti havaita, miten mahdolliset muutokset ovat vaikuttaneet yrityksen suoritukseen eri osa-alueilla. Mittareiden avulla saadaan myös huomattavasti enemmän tietoa eri osa-alueiden kehityksestä, jolloin on helpompi tehdä ratkaisuja muutoksien suhteen ja varmistaa, että eri osa-alueet noudattavat yrityksen päättämää strategiaa. (Kaplan, R & Norton, D, 2007)

2.8 Prosessien mallintaminen

Prosessien mallintamisella on tarkoitus selkeyttää yrityksen sisäisiä toimintamalleja purkamalla ne graafiseksi esitykseksi. Yrityksen sisällä tämä selkeyttää huomattavasti toimintaa, koska tarvittavat prosessit ovat selkeästi mallinnettu. Myös QPR käyttää varsin kattavasti prosessien mallintamista. Esimerkiksi virheidenkäsittelyn prosessi on tuotekehityksessä määritelty varsin tarkasti. (White, S)

3 CART - ohjelma

3.1 Esittely

Continuous Automated Regression Testing tool eli CART, on testaustyökalu QPR Softwaren QPR Scorecard sekä QPR Processguide tuotteiden testaamiseen. CART on helppokäyttöinen ja varsin kattava testaustyökalu, jolla pyritään laadun parantamiseen. Helppokäyttöisyys oli alusta alkaen tärkeä lähtökohta, koska CART on tarkoitettu pääsääntöisesti kehittäjien käytettäväksi.

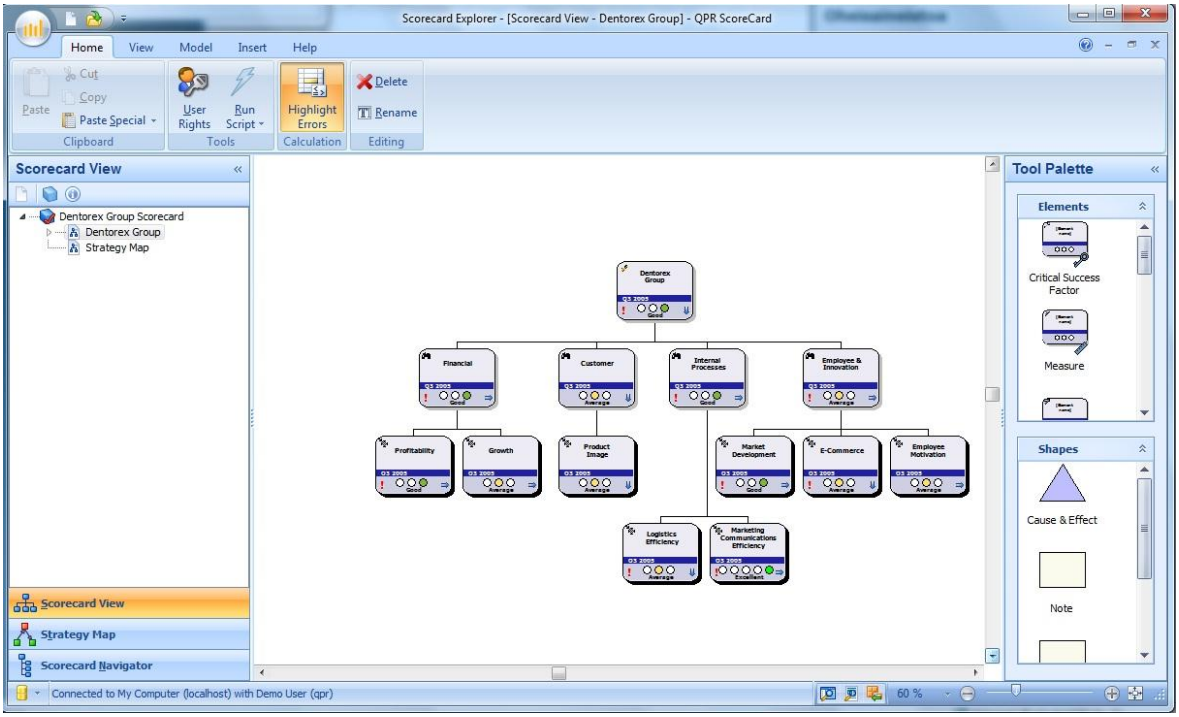
CART:n avulla pyritään suorittamaan kattavia regressiotestejä automaattisesti, jotta kehittäjät pystyisivät suorittamaan testejä ilman merkittävää resurssien käyttöä. Tämä nostaa suoritettavien testien kattavuutta jo aikaisessa vaiheessa, joka omalta osaltaan mahdollistaa ohjelmiston laadun parantumisen. CART testaa QPR:n ohjelmien toiminnallisuutta eli se keskittyy puhtaasti mustalaatikkotestaukseen. Testauksen ajankohtana toimii kehittäjillä pääsääntöisesti integraatiovaihe sekä järjestelmätestausvaihe. Testaajilla järjestelmätestausvaihe.

CART on suunniteltu toimimaan kaikilla QPR 8.1 tukemilla alustoilla.

3.2 Testattavat ohjelmat

Kuten johdannossa mainittiin, CART ei testaa kaikkia QPR:n ohjelmia. CART:n käyttämä API – rajapinta on tarpeeksi laaja vain QPR Scorecard sekä QPR Processguide – ohjelmissa, joten ne ovat ainoat testattavat ohjelmat.

3.2.1 QPR Scorecard

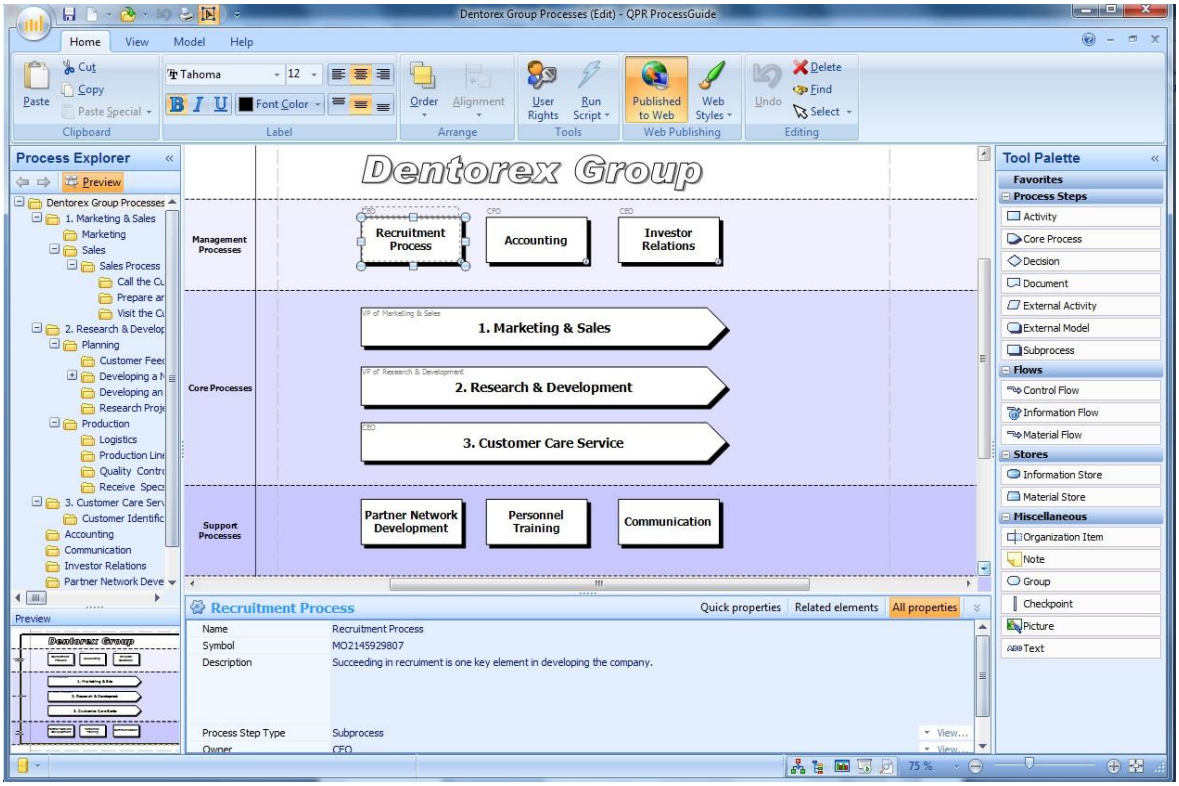


Kuva 2. QPR Scorecard – Scorecard View

QPR Scorecard on tuloskortti – ohjelma, jonka tavoitteena on auttaa yrityksiä mittaamaan suorituskykyään eri osa-alueilla. Kappaleessa 2.7 on tarkempaa tietoa tuloskorttien periaatteesta.

CART ei pysty testaamaan QPR Scorecardin graafisia ominaisuuksia. Testaus keskittyykin mallien luontiin ja poistamiseen, tuloskorttien luontiin, muokkaamiseen ja poistamiseen, mittareiden luontiin, muokkaukseen ja poistamiseen sekä käyttöoikeuksiin. Eri arvoasetusten, periodien sekä sarjojen luonti ei ole mahdollista QPR APIn kautta, joten CART ei pysty myöskään luomaan uusia asetuksia näiden suhteen. Tämän vuoksi CART käyttää pohjana valmiiksi rakennettuja malleja, joiden pohjalta se luo uuden mallin sekä vaihtelee näitä asetuksia eri mittareissa.

3.2.2 QPR Processguide



Kuva 3. QPR Processguide – Flowchart View

QPR Processguide on prosessien mallintamiseen tarkoitettu ohjelma. Prosessien mallintamisella pyritään selkeyttämään yrityksen sisäisiä prosesseja. Prosessien mallintamisen avulla pyritään myös parantamaan prosesseja. Kun prosessit avataan kaavioksi, niitä on helpompi tutkia, jolloin mahdolliset ongelmakohdat ovat helpommin löydettävissä.

Processguiden pääpaino on graafinen. API ei pysty varsinaisesti varmentamaan, että kaikki elementit luodaan oikeisiin paikkoihin kaavioihin, joten CART ei myöskään pysty tätä tekemään. Testaamisen pääpaino on elementtien luomisessa, muokkaamisessa, poistamisessa ja käyttöoikeuksissa. Processguidessa on myös mahdollista viedä tai tuoda dataa XML – muodossa ja CART testaa myös tätä toiminnallisuutta.

3.3 Suunnitelma

3.3.1 Määritys

Määrittely aloitettiin määrittelemällä käyttötapaukset, toimijat sekä toimijakuvaukset. Toimijoiksi määriteltiin kehittäjät sekä testaajat. Käyttötapauksiksi määriteltiin molemmille

ohjelmille kolme eri tasoista testiä, sekä Delphi Unit testing projectin suorittaminen CART kautta. Projektin määrityskuvasto löytyy liitteestä 9.

Tutkittaessa Delphi Unit testing projectia, kävi varsin nopeasti selväksi, että teknisesti tämä osio olisi huomattavan monimutkainen ja aikaa vaativa. Tämä olisi vaatinut huomattavaa perehtymistä sekä Delphi – koodiin, että käytettyyn kehitysympäristöön. Tätä ei ollut mahdollista toteuttaa tämän työn resursseilla, joten Delphi Unit testing project pudotettiin pois CART toteutuksesta.

3.3.2 Varsinaisen ohjelman suunnittelu

Suunnittelu aloitettiin rakentamalla perusmalli CART:in toiminnallisuudelle. Perusmalli oli varsin yksinkertainen, jossa oli otettava huomioon vain perustoiminnot. Perustoiminnot ovat servereiden käynnistys, skriptien suorittaminen oikeilla ohjelmilla sekä tulosten raportoiminen.

Kun toiminnallisuuden perusrakenne oli suunniteltu, valittiin kolme kehittäjää, joille suoritettiin kysely heidän käytössä olevasta kehitysympäristöstään. Näiden perusteella suunniteltiin ohjelman tarkempi toiminnallisuus, kuten käytettävät polut sekä kansiot ja mahdolliset eroavaisuudet kehitysympäristöissä, kuten onko kehitysympäristö 32 – vai 64 - bittinen.

Tämän jälkeen seuraavat ongelmat oli ratkaistava: Miten tietokanta säilyy koskemattomana, jotta testien aloitustila pysyy aina samana ja miten mahdolliset päivitykset tietokantaan hoidetaan? Miten estetään yhtäaikaisen käytön ongelmat ajettaesssa samaa testiskriptia kahdelta eri koneelta ja miten testiskriptien päivitys hoidetaan? 32 – ja 64 – bittisen ympäristöjen erot ja niiden aiheuttamat haasteet?

Tietokannan koskemattomuus ratkaistiin niin, että varsinainen testikanta sijaitsee tietyssä tietokannassa. CART käyttää hyödykseen QPR TransferToolia, jonka avulla kanta siirretään käyttäjän omalla koneella sijaitsevaan kantaan. Tällöin kanta säilyy muuttumattomana ja on kaikille sama testien alkaessa. Keskittämällä kannan yhteen paikkaan, myös kannan päivittäminen helpottuu huomattavasti. QPR TransferTool siirtää aina koko kannan, joten uudet päivitykset tulevat käyttöön välittömästi kaikille.

Skriptien päivitys ja niiden yhtäaikaisen käytön ongelmat ratkaistiin laittamalla ne samalle koneelle kuin kanta. CART kopioi suoritettavat skriptit käyttäjän koneelle, jolloin on vain

erittäin pieni todennäköisyys, että kyseinen skripti olisi varattu joka estäisi kopioinnin ja aiheuttaisi testien epäonnistumisen. CART myös poistaa käyttäjän koneelta vanhat skriptit. Tämän ansiosta CART käyttää aina uusimpia skripteja.

3.4 Toteutus

Toteutuksessa haasteita loi valittu kieli sekä määrittelyssä tehty perustavaa laatua oleva virhe. CART on totetutettu Visual Basicilla. Ennen toteutuksen alkua myös Java sekä C# testattiin lyhyesti. Kielen valintaan vaikutti eniten se, että QPR API tukee vbscriptia, joten oletuksena oli että tästä olisi hyötyä.

Käytännössä CART:sta toteutettiin työn aikana kaksi eri versiota. Ensimmäinen versio toteutettiin alkuperäisen määrittelykuvaston perusteella. Suurimpia haasteita tuli alusta alkaen valitusta kielestä. Visual Basicin muiden ohjelmien hallinta mahdollisuudet ovat varsin rajalliset tai sitten saadakseen tehtyä perusasioita, kuten prosessin lopettaminen, olisi vaatinut monta sataa riviä koodia. Haastavin osa kehittämisessä olikin juuri muiden suoritettavien tiedostojen sammuttaminen. Ongelma ratkaistiin lopulta kirjoittamalla erillinen skripti käyttäen apuna AutoHotkey:ta. CART suorittaa kyseisen skriptin tarvittaessa.

32 - ja 64 – bittisen ympäristön erot CART:in kannalta rajoittuvat vain polun nimiin. QPR Softwaren ohjelmistot asentuvat vakiona C:\Program Files – hakemistoon. Koska kyseessä on 32 – bittinen ohjelmisto, asentuu se 64 – bittisessä ympäristössä C:\Program Files(x86) – hakemistoon. CART tunnisti bittisyyden tarkistamalla, löytyykö C:\Windows – hakemistosta ”SysWOW64” – kansiota, joka löytyy ainoastaan 64 – bittisestä ympäristöstä. Tämän perusteella CART:n ensimmäinen versio valitsi oikein Program Files – polun jota käyttää.

Muita suurempia ongelmia ei ensimmäisen version kanssa oikeastaan ollut. Kielen valinta aiheutti pieniä ongelmia koko ajan jo mainituista syistä. Version testauksessa käytettiin kehitysympäristön lisäksi yhden kehittäjän ympäristöä. Testeissä ei löytynyt mitään suurempia ongelmia. Pieniä parannuksia tehtiin käyttöliittymään sekä testiskriptit muokattiin ottamalla huomioon se, että kehittäjien ympäristössä ei ole käytössä kielitiedostoja eli Dictionaryja. Tämä toteutettiin niin, että mittari ja elementti tyypit muutettiin kovakoodatuista tyypeistä muuttujiksi, jotka haetaan skriptia ajettaessa. Toteutuksen mahdollisti API – toiminnallisuus PGModel.Find, jonka avulla pystytään hakemaan elementtityyppien nimet. Tämä myös mahdollistaa CART käyttämisen kaikilla eri kielillä mitä QPR ohjelmat tukevat.

Näiden muutosten jälkeen CART lähetettiin hyväksymistestaukseen. Testauksessa löytyi erittäin vakava puute. CART oli suunniteltu määrittelyn perusteella käytettäväksi vain tietystä polusta, olettaen että kaikilla kehittäjillä on samanlainen kehitysympäristö käytössään. Tämä piti kaikkien muiden kohdalla paitsi yhden kehittäjän. Testeissä tuli myös esille, että kaikki kehittäjät eivät käytä samoja portteja. Nämä kaksi testeissä löydettyä ongelmaa aiheutti käytännössä sen, että ohjelma oli suurelta osalta tehtävä täysin uudestaan. Näiden lisäksi tuli myös palautetta siitä, että CART oli suunniteltu toimimaan vain yhdestä tietystä hakemistosta ajettaessa. Pieniä haasteita aiheutti myös se, että kaikilla kehittäjillä ei ollut käytössä vakio – asennuksessa tulevaa Access – tietokantaa ja sen mukana tulevaa ODBC – yhteyttä. Tämän vuoksi kehittäjien piti aluksi lisätä itse ODBC – yhteys, jotta kanta pystyttiin importtaamaan oikein.

CART:iin lisättiin oma ini – tiedosto sekä käyttöliittymään ”Settings” – painike, josta ini – tiedosto avataan. Käyttäjä voi itse määritellä käytettävien ohjelmien polut, portit sekä tietokannan. CART lukee nämä kaikki tiedot ja käyttää niitä hyväkseen. Liitteessä 2 on kuvattu CART.ini – tiedoston sisältö ja Liitteessä yksi, kohdassa ”Sub ReadSettings” löytyy toiminnallisuus, jolla arvot luetaan muuttujiin. Ohjelmien polkujen sekä tietokannan tietojen käyttäminen on toteutettu yksinkertaisesti lukemalla tiedot muuttujiin suoraan. Tämä ratkaisu korjasi ongelmat tietokantojen ja ei-vakio asennusten suhteen.

Kyseisestä ”Settings” – ratkaisusta oli myös ongelmien ratkaisun lisäksi aivan suunnaton hyöty. Alkuperäinen CART oli suunniteltu vain yhdelle QPR – versiolle, mutta ratkaisun ansiosta se mahdollisti myös muiden QPR versioiden testaamisen. Ainoana rajoitteena on tietokanta, joka on luotu 8.0 versiolla sekä testiskriptit, jotka sisältävät vain 8.1 ja uudemmissa versioissa tuettuja toiminnallisuuksia, joten tämänhetkinen tilanne mahdollistaa versioiden 8.1 ja 8.2 (kehitysversio) testaamisen.

Käytettyjen porttien tietojen käyttäminen ei ollut aivan niin yksinkertaista. Portin tietoja käytetään autentikoidessa testiskripteissä, joten muuttuja piti saada itse skriptiin. Tämä toteutettiin laittamalla skriptoihin tietty merkkijono portin tilalle. CART lukee skriptin kokonaisuudessaan muuttujaan, hakee ini – tiedostosta portin arvon, muokkaa tietyn merkkijonon arvon vastaamaan ini – tiedostossa olevaa arvoa ja luo skriptin koneelle, jolla testit ajetaan.

CART suorittaminen onnistuu mistä tahansa hakemistosta. Ongelma ratkaistiin vain tekemällä CART:iin toiminto, joka tarkistaa mistä hakemistosta CART.exe on suoritettu.

3.4.1 Testiskriptit

CART sisältää yhteensä kuusi erilaista testiskriptia. Kolme QPR Scorecardille sekä kolme QPR Processguidelle. Molemmilla tuotteilla on Smoke, Intermediate ja Advanced tason skriptit.

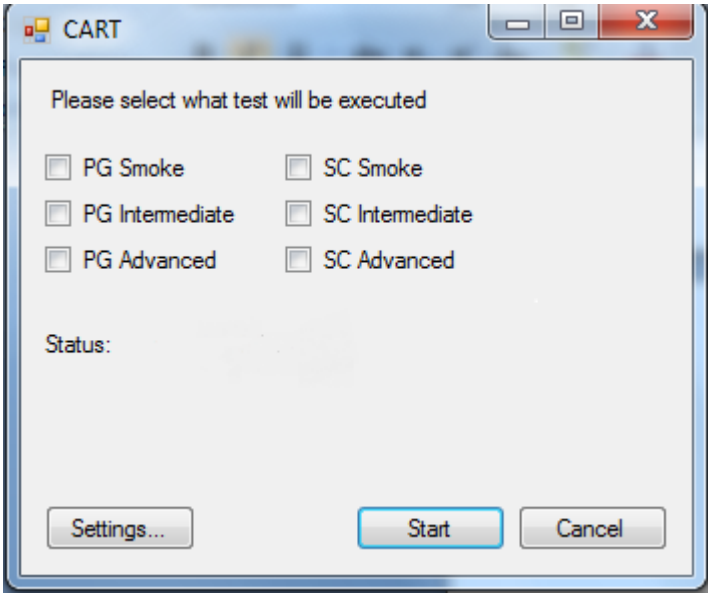
Smoke tasolla käydään läpi vain aivan perustoimintoja. Testeissä suoritetaan muutama perus API – komento, kuten serveriin yhteyden ottaminen ja autentikointi, mallin avaus, elementin luominen malliin sekä mallin tallennus. Tarkoituksena on vain varmistaa, että ohjelma käynnistyy oikein, autentikointi toimii sekä se, ettei mahdolliset muutokset ole rikkoneet mm. malliformaattia. Liitteissä 3 ja 4 löytyvät molemmat Smoke - testiskriptit

Intermediate tasolla suoritetaan jo hieman monimutkaisempia komentoja sekä testataan käyttöoikeuksia. Tämän tason testit yrittävät simuloida normaalikäyttöä, tosin ilman odotteluja. API komennot käydään läpi varsin kattavasti. Testikannassa löytyvillä käyttäjillä on erilaisia oikeuksia ja näillä käyttäjillä yritetään tehdä erilaisia muutoksia. Liitteissä 5 ja 6 löytyvät molemmat Intermediate - testiskriptit

Advanced tasolla suoritetaan kaikki mahdolliset API – komennot, jotka eivät vaadi käyttäjän toimia mennäkseen läpi. Myös käyttöoikeudet testataan tällä tasolla. Tämän tason testit varmistavat, että API komennot toimivat. Myöskin eri komentojen eri propertyt suoritetaan. Liitteissä 7 ja 8 löytyvät molemmat Advanced – testiskriptit

3.5 Toiminnallisuus

CART – ohjelman suorittamisessa on käytännössä kolme eri vaihetta, jotka käyttäjän on käytävä läpi. Suunnittelu ja valmistelut, testien suorittaminen sekä tulosten analysointi ja jatkotoimenpiteet.



Kuva 4. CART käyttöliittymä

Oheisessa kuvassa on kuva CART – käyttöliittymästä. Valintaruutujen avulla käyttäjä määrittelee, mitä testejä halutaan suorittaa. Käyttäjä voi valita joko yhden tai useampia testejä samalla kertaa suoritettavaksi. Settings – napista avataan CART.ini notepadiin, jolloin käyttäjä voi muokata asetuksia haluamallaan tavalla. Käyttäjän on tosin muistettava tallentaa muutokset tiedostoon notepadin kautta. Start – napista CART lähtee suorittamaan valittuja testejä. Cancel – nappi sulkee ohjelman.

3.5.1 Suunnittelu ja valmistelut

Ensimmäisellä kerralla on käyttäjän kopoitava CART omalle koneelleen määritellystä paikasta. CART on kopioitava kohteeseen, jossa ei vaadita järjestelmän ylläpitäjän oikeuksia, jotta ohjelma voidaan ajaa. Esimerkiksi Windows 7 – käyttöjärjestelmässä, jossa User Account Control (UAC) on aktivoitu, ei CART:ia voida asentaa suoraan C – juureen. Muita rajoituksia käytettävällä kansiolla ei ole.

Käyttäjän on suunniteltava, mitkä testit suoritetaan. Suunnittelussa on otettava huomioon, mihin ohjelman osioon muutokset ovat tehty ja mihin mahdollisesti osioon muutokset myös mahdollisesti vaikuttavat.

Koska CART:ssa on valmiina jo testimallit ja skriptit, on käyttäjän valmisteluna määritettävä oikeat tiedot cart.ini - tiedostoon. Määritettävät asiat ovat servereiden ja clienttien polut,

käytettävän tietokannan ODBC – yhteyden nimi, käyttäjätunnus sekä salasana. Myös client – ohjelmien käyttämät portit on määritettävä.

3.5.2 Testien suorittaminen

Suunnitellut testit suoritetaan valitsemalla käyttöliittymästä haluttujen testien checkboxit. Tämän jälkeen CART automaattisesti tekee tarvittavat alustukset, siirtää testikannan testattavaan ympäristöön ja suorittaa valitut testit. CART ilmoittaa testien tulokset sen jälkeen, kun kaikki valitut testit on suoritettu.

3.5.3 Tulosten analysointi ja jatkotoimenpiteet

Analysoinnissa otetaan huomioon, onko CART löytänyt virheitä vai ei. Jos virheitä ei ole löytynyt, ei ole syytä jatkotoimenpiteisiin. Jos virheitä on löytynyt, on kehittäjän silloin analysoitava, mistä virhe on johtunut ja korjattava virheen aiheuttanut osa. Virheen korjaamisen jälkeen on suoritettava testit uudelleen, jotta voidaan varmistua korjauksen onnistumisesta sekä siitä, ettei uusi korjaus ole aiheuttanut ei-toivottuja vaikutuksia ohjelman eri osiin.

3.6 Tulokset

Toistaiseksi CART:sta ei ole ollut kovin suurta hyötyä. Käytettyihin tunteihin verrattuna, CART ei ole löytänyt tarpeeksi virheitä tai estänyt virheiden päätymistä varsinaiseen koodiin. Suurin ongelma on ollut ohjelman käyttöönotossa, joka ei onnistunut halutulla tavalla. Syynä käyttöönoton epäonnistumiseen on todennäköisesti varsin puutteelliset ohjeet sekä epävarmuus, jonka ensimmäiset testit aiheuttivat. Toki osansa on myös sillä, että monilla kehittäjillä ovat omat työtapansa ja niiden tapojen muuttaminen, kuten uuden työkalun käyttöönotto, ei välttämättä käy niin helposti. Kehittäjille tehdyn kyselyn jälkeen kävi myös ilmi, että yksi ei edes tiennyt oikeastaan ohjelman käytöstä mitään, koska hän oli ollut pitkällä lomalla ja ohjeistukset ym. olivat menneet täysin ohi ja toisille aiheutti edelleen epäselvyyttä, mitä CART oikeastaan testaa. Kommentit tosin hieman herättävät myös ihmetystä, koska kyseiset kehittäjät olivat osallisena hyväksymistestauksessa.

Käyttöönoton epäonnistumisen takia, ei voida tehdä johtopäätöksiä sen suhteen, voiko testausautomaatio parantaa QPR tuotteiden laatua. CART on toimintansa aikana löytänyt 4

virhettä, jotka ovat päätyneet ohjelman kehitysversioon. Vertauksena, muu testaaminen on löytänyt 153 virhettä kehitysversiosta kuukauden aikana.

3.7 Arviointi

CART ohjelmassa ja työssä on molemmissa paljon hyvää ja huonoa. Prosessin johtamisessa ja ajankäytön hallinnassa olivat työn suurimmat puutteet. Projekti ei valmistunut määräaikaan mennessä ja projektinohjauskokous, jossa hyväksyttiin aikataulun muutokset, pidettiin aivan liian myöhään. Projektin myöhästymisen aiheutti pitkälti hyväksymistestauksessa löytyneet ongelmat, jotka oli pakko korjata, jotta CART voitaisiin ottaa käyttöön.

Hyväksymistestauksessa löydetyt ongelmat tosin olivat varsin hyödyllisiä CART tulevaisuuden käytön kannalta, koska tarvittavat muutokset mahdollistivat CART:n käyttämisen myös muilla versioilla. Ainoa rajoittava tekijä on enää testiskriptit, koska niissä saattaa olla toiminnallisuuksia, jotka eivät ole tuettuja tietyillä versioilla.

Itse ohjelman suurimmat puutteet ovat virheidenkäsittely sekä status – bar. Ohjelma käsittelee testauksessa sattuvat virheet oikein, mutta jos esimerkiksi ini – tiedosto sisältää vääriä tietoja, ei CART ota siihen kantaa mitenkään ja esiin tuleva virheilmoitus ei ole tarpeeksi informatiivinen.

Kokonaisuudessaan työn heikkous on dokumentointi ja raportointi. Suuret muutokset osittain vaikeuttivat asiaa. Kiireessä ohjelman varsinaista toiminnallisuutta tehdessä, dokumentoinnissa on otettu valitettavia oikoteitä, jotka vaikuttivat työn onnistumiseen ja käyttöönottoon valitettavan paljon.

Yhteenvetona voidaan sanoa, että kokonaisuudessaan CART epäonnistui tavoitteessaan parantaa QPR:n ohjelmistojen laatua. Löydettyjen virheiden määrä suhteessa muilla keinoilla löydettyihin virheiden määrään jäi erittäin pieneksi, jolloin voidaan sanoa tehdä johtopäätös, että vaikutus laatuun on erittäin vähäinen.

3.8 Resurssien käyttö

Yhteensä CART kehitys ja varsinaisen työn dokumentointi on vienyt 485 tuntia tekijältä. 400 tunnin tavoite ylittyi käytännössä sen takia, että CART vaati varsin massiivisia muutoksia

ensimmäisen hyväksymistestauksen jälkeen. Kehittäjillä testaukseen on mennyt yhteensä n.5 tuntia.

Hyväksymistestauksessa löydettyjen ongelmien lisäksi, ongelmia aiheutti myös uudet toiminnallisuudet QPR:n osalta. Syksyn aikana toteutetut toiminnallisuudet aiheuttivat muutoksia CART:n testiskripteihin, jolloin muutoksia täytyi tehdä tai odottaa, että toiminnallisuudet tulevat valmiiksi. Tämä odottelu sekä muutamat muut seikat aiheuttivat sen, että resurssien käyttö ei jakaantunut tasaisesti halutulla tavalla vaan työtä tehtiin välillä varsin verkkaisesti ja välillä huomattavia määriä viikossa.

4 Oppimiskokemukset

Oppimiskokemuksena CART toteuttaminen on ollut erittäin monipuolista ja haastavaa. Kyseisessä työssä varsinkin itse toteutuksen osa kävi erittäin haastavaksi käytetyn kielen takia ja puutteellisten määritysten vuoksi.

Määritystä tehtäessä, ei otettu kohderyhmää tarpeeksi hyvin huomioon. Kohderyhmän ollessa varsin pieni, olisi parempi kysyä kaikilta kohderyhmän jäseniltä tarvittavat tiedot. CART – tapauksessa ongelmaksi osoittautui kehitysympäristön rakenne. Yhtä kehittäjää lukuunottamatta, kaiki kehittäjät käyttävät samanlaista polkurakennetta. CART suunniteltiin kyseiselle polulle, jolloin yhden kehittäjän kohdalla CART ei toiminut laisinkaan. Tämän tilanteen olisi voinut välttää tekemällä kattava kysely.

Toinen suuri haaste oli käytetty kieli eli Visual Basic. Kieli oli melko rajallinen ja se aiheutti vaikeuksia toteutuksen kanssa. Tekemällä kattavan pohjustuksen eri kielivaihtoehdoista, kielen aiheuttamilta ongelmilta oltaisiin vältytty ainakin joltain osilta. Vääriin valintoihin toki vaikutti myös vahvasti se, että tekijän ohjelmointikokemus ei ole kovin laaja, joten paljon resursseja jouduttiin käyttämään kielen opiskeluun.

Ohjelmiston ja prosessin dokumentoimisen tärkeys tuli ilmi varsin selvästi työn loppuvaiheessa.

5 Ehdotukset jatkotoimenpiteksi

QPR:llä on käytössä myös toinen regressiotestausjärjestelmä Fully Automated Test Environment (FATE). CARTin skriptojen käsittely on tarkoituksella tehty melko samankaltaiseksi, jotta mahdollinen osittainen integraatio olisi mahdollinen. CARTin käyttöliittymä tosin ei tue toistaiseksi muiden kuin CART:in omien skriptojen suorittamista, joten tämä vaatisi myös käyttöliittymän muutosta. Tämä voitaisiin käytännössä toteuttaa niin, että käyttäjä saa vapaasti valita, minkä skriptin hän haluaa suorittaa. CART:n pitäisi myös osata muokata skriptia niin, että se olisi yhteensopiva CART:n kanssa.

QPR on myös hankkimassa käyttöliittymän testaamiseen tarkoitettua testaustyökalua, joten olisi varsin hyödyllistä integroida CART testit myös toisen työkalun testeihin. Tämän pystyy toteuttamaan todella pienillä muutoksilla, joten hyöty olisi varsin välitön pienellä resurssien käytöllä. Käytännössä tämä tarkoittaisi testiskriptien muokkausta, jotta niitä pystyttäisiin suorittamaan ilman CART tekemiä muokkauksia.

Koska CART ei itse sinäänsä ole versio-riippuvainen, mutta testiskriptit ovat, olisi hyvä jatkuvasti kehittää ja päivittää skriptoja eri versioille. Tämä vaatii myös toki muutoksia itse CARTiin, jotta käyttäjä pystyy valitsemaan, minkä version testejä ajetaan, mutta muutoksista olisi huomattavaa hyötyä. CART:iin pitäisi lisätä valinta, jolla käyttäjä voi määritellä, minkä version skriptit ajetaan.

Tällä hetkellä käyttäjä ei varsinaisesti pysty mitenkään vaikuttamaan ajettavien skriptojen sisältöön. Olisi hyödyllistä, että käyttäjä pystyisi valitsemaan esimerkiksi sen, mitä API komentoja ja niiden parametreja ajetaan. Tämän toteuttamiseksi pitäisi tehdä varsin suuria muutoksia sekä CART:iin, että skripteihin. Tämän toteuttaminen toisaalta tekisi eri versioiden skriptien luomisen melkein turhaksi, koska käyttäjä pystyisi itse määrittelemään käytetyt parametrit. Tästä olisi myös varsinkin hyötyä uusien kehittäessä uusia API komentoja, koska nämä uudet komennot eivät luonnollisesti voi olla mukana vielä skrirteissa. Tämä vaatisi sen, että CART pystyisi muokkaamaan skripteja huomattavasti monipuolisemmin kuin nyt.

CART koulutus, joka tukisi kehittäjien työkalun käyttöönottoa, jotta työkalun mahdollisuuksista saataisiin mahdollisimman suuri hyöty irti. Kehittäjät myös haluaisivat tarkemmat tiedot, mitä CART ja sen testit pitävät sisällään, jotta he saisivat hyödyn irti

mahdollisimman hyvin, joten ohjeiden sekä muiden dokumentaatioiden tasoa on tarkennettava huomattavasti.

6 Suositukset toimintatapojen muuttamikseksi

QPR:n sisällä toimintatapoja olisi muutettava niin, että kehittäjät alkaisivat todenteolla käyttämään CART:ia jokapäiväisessä työskentelyssään. Tällöin CART:sta saataisiin mahdollinen hyöty parhaiten irti. CART käyttö pitäisi määrittää normaaliin kehitysprosessiin, jolloin CART – käyttö olisi pakollista ennenkuin koodimuutokset lisätään yhteiseen koodipohjaan.

Vastaavien ympäristöjen toteutuksessa on tulevaisuudessa syytä suorittaa alustavat tutkimukset kaikille kehittäjille. Tämä on mahdollista, koska kehittäjiä on varsin rajallinen määrä. Tämän toimintatavan avulla vältyttäisiin yllätyksiltä esimerkiksi tarkastatteassa kehittäjien ympäristöjä ja kehittämällä joitain työkaluja kyseisiin ympäristöihin.

Kattavammat tutkimukset käytettävien ohjelmointikielien päätöksiin. Tämän toteuttaminen tosin vaatii parempaa tuntemusta ohjelmoinnista yleisesti. Eri ohjelmointikielillä tulisi toteuttaa pienimuotoinen ohjelma, jotta kielen käyttö tulisi tutummaksi.

Selkeä käyttöönotto ja ohjeistukset. Määritellään koska ohjelma otetaan käyttöön, luodaan selkeät ohjeet sekä määritellään, miten, kuka ja milloin ohjelmaa käyttää.

Työn dokumentoinnin parantaminen. Kaikki toiminnallisuudet sekä ongelmatilanteet pitäisi kirjata ylös. Tämä auttaisi käyttämisessä ja alentaisi kynnystä käyttöönotossa.

Lähteet

Tian, J. 2005a. Software Quality Engineering. Testing, Quality Assurance and Quantifiable Improvement. s. 97-101. John Wiley & Sons, Inc. Hoboken, NJ.

Tian, J. 2005b. Software Quality Engineering. Testing, Quality Assurance and Quantifiable Improvement. s.18-19 . John Wiley & Sons, Inc. Hoboken, NJ.

Tian, J. 2005c. Software Quality Engineering. Testing, Quality Assurance and Quantifiable Improvement. s.27-28 . John Wiley & Sons, Inc. Hoboken, NJ.

Tian, J. 2005d. Software Quality Engineering. Testing, Quality Assurance and Quantifiable Improvement. s.213 . John Wiley & Sons, Inc. Hoboken, NJ.

Perry, W.E. 2000a. Effective Methods for Software Testing. 2.Painos. s.114-116. John Wiley & Sons

Perry, W.E. 2000b. Effective Methods for Software Testing. 2.Painos. s.67-69. John Wiley & Sons

ISO 2001. ISO/IEC 9126-1:2001 Software Engineering – Product Quality – Part 1: Quality Model. ISO.

Van Veenendal, E.& all. 2007a. Standard glossary of terms used in Software testing. s.28. Luettavissa: http://www.istqb.org/downloads/glossary-current.pdf. Luettu: 31.3.2010

Van Veenendal, E. & all. 2007b Standard glossary of terms used in Software testing. s.9,12,39. Luettavissa: http://www.istqb.org/downloads/glossary-current.pdf. Luettu: 31.3.2010

Oulun seudun ammattikorkeakoulu OAMK. Software Business Competence, Testaustasot. Luettavissa: http://www.oamk.fi/sbc/testaus/testaustasot.htm
Luettu 5.4.2010

Wikipedia. W. Edwards Deming. Luettavissa:
http://en.wikipedia.org/wiki/W._Edwards_Deming Luettu 5.4.2010

American Society of Quality. Project Planning and Implementing Tools. Plan-Do-Check-Act Cycle. Luettavissa: http://asq.org/learn-about.quality/project-planning-tools/overview/pdca-cycle.html Luettu: 5.4.2010

Robert S. Kaplan & David P. Norton. Using the Balanced Scorecard as a Strategic Management System. Luettavissa: http://donwload.microsoft.com/documents/uk/peopleready/Using the Balanced Scorecard as a Strategic Management System.pdf. Luettu: 11.4.2010

Stephen A. White. Introduction to BPMN. Luettavissa: http://www.bpmn.org/Documents/Introduction_to_BPMN.pdf. Luettu: 11.4.2010

Liitteet

Liite1. Lähdekoodi

```
Imports System.Diagnostics
Imports System.IO

Public Class CART
    Dim SCSmoke As Boolean
    Dim SCInt As Boolean
    Dim SCAdv As Boolean
    Dim PGSmoke As Boolean
    Dim PGInt As Boolean
    Dim PGAdv As Boolean
    Dim PGExpImp As Boolean = False
    Dim SCExpImp As Boolean = False
    Dim Result As Boolean = True
    Dim ResultStr As String = ""
    Dim iniFile As String
    Dim SCS As String
    Dim UMS As String
    Dim PGS As String
    Dim WAS As String
    Dim SCD As String
    Dim PGD As String
    Dim SCPort As String
    Dim PGPort As String
    Dim TT As String
    Dim ODBC As String
    Dim DBUsenN As String
    Dim DBPass As String
    Dim PFolder As String
    Public Sub CreateSettings()
        'This sub checks if ini - file already exists and if not, it
creates it
        Dim CFile As Boolean
        Dim dTaksId As Double
        Dim NotePad As String = "C:\Windows\notepad.exe"

        Call GetCartFolder()

        CFile = My.Computer.FileSystem.FileExists(iniFile)

        If (CFile = True) Then
            dTaksId = Shell(NotePad & " " & iniFile,
AppWinStyle.NormalFocus)
        Else
            FileSystem.FileCopy("\\qpr-
118cart.qpr.com\CART\files\Cart.ini", iniFile)
            dTaksId = Shell(NotePad & " " & iniFile)
        End If
    End Sub

    Private Sub ExecuteProgram()
        'Check what scripts are selected and executes the selected ones
```

```
        Call DeleteOldLogs()

        If SCSmoke = True Then
            StatusText.Text = "Executing SCSmoke"
            Call ExportImportDB("SC")
            SCExpImp = True
            Call StartServers()
            Call ExecuteScipt("SC_Smoke.smf", "SC")
            Call ReadLog("SC_SMOKE_Log.txt", "SCSmoke")
            Call CloseServers()
        End If
        If SCInt = True Then
            StatusText.Text = "Executing SCIntermadiate"
            Call ExportImportDB("SC")
            SCExpImp = True
            Call StartServers()
            Call ExecuteScipt("SC_Intermediate.smf", "SC")
            Call ReadLog("SC_INT_Log.txt", "SCIntermadiate")
            Call CloseServers()
        End If
        If SCAdv = True Then
            StatusText.Text = "Executing SCAdvanced"
            Call ExportImportDB("SC")
            SCExpImp = True
            Call StartServers()
            Call ExecuteScipt("SC_Advanced.smf", "SC")
            Call ReadLog("SC_ADV_Log.txt", "SCAdvanced")
            Call CloseServers()
        End If
        If PGSmoke = True Then
            StatusText.Text = "Executing PGSmoke"
            Call ExportImportDB("PG")
            PGExpImp = True
            Call StartServers()
            Call ExecuteScipt("PG_Smoke.pmf", "PG")
            Call ReadLog("PG_SMOKE_Log.txt", "PGSmoke")
            Call CloseServers()
        End If
        If PGInt = True Then
            StatusText.Text = "Executing PGIntermediate"
            Call ExportImportDB("PG")
            PGExpImp = True
            Call StartServers()
            Call ExecuteScipt("PG_Intermediate.pmf", "PG")
            Call ReadLog("PG_INT_Log.txt", "PGIntermediate")
            Call CloseServers()
        End If
        If PGAdv = True Then
            StatusText.Text = "Executing PGAdvanced"
            Call ExportImportDB("PG")
            PGExpImp = True
            Call StartServers()
            Call ExecuteScipt("PG_Advanced.pmf", "PG")
            Call ReadLog("PG_ADV_Log.txt", "PGAdvanced")
            Call CloseServers()
        End If
        PGExpImp = False
        SCExpImp = False
        Call TestResult()
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
```

```
        Call ExecuteProgram()

    End Sub
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click

        Close()
    End Sub
    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click

        Call CreateSettings()

    End Sub
    Public Function DirExists(ByVal OrigFile As String)
        Dim fs
        fs = CreateObject("Scripting.FileSystemObject")
        DirExists = fs.folderexists(OrigFile)
    End Function

    Private Sub CheckBox1_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CheckBox1.CheckedChanged
        If CheckBox1.CheckState = CheckState.Checked Then
            PGSmoke = True
        Else
            PGSmoke = False
        End If
    End Sub

    Private Sub CheckBox2_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CheckBox2.CheckedChanged
        If CheckBox2.CheckState = CheckState.Checked Then
            PGInt = True
        Else
            PGInt = False
        End If
    End Sub

    Private Sub CheckBox3_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CheckBox3.CheckedChanged
        If CheckBox3.CheckState = CheckState.Checked Then
            PGAdv = True
        Else
            PGAdv = False
        End If
    End Sub

    Private Sub CheckBox4_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CheckBox4.CheckedChanged
        If CheckBox4.CheckState = CheckState.Checked Then
            SCAdv = True
        Else
            SCAdv = False
        End If
    End Sub

    Private Sub CheckBox5_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CheckBox5.CheckedChanged
        If CheckBox5.CheckState = CheckState.Checked Then
            SCInt = True
        Else
            SCInt = False
```

```
        End If
    End Sub

    Private Sub CheckBox6_CheckedChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CheckBox6.CheckedChanged
        If CheckBox6.CheckState = CheckState.Checked Then
            SCSmoke = True
        Else
            SCSmoke = False
        End If
    End Sub

    Private Sub ExportImportDB(ByVal Prod As String)
        'This sub executes the export/import from transfertool
        Dim ProcID As Integer
        Dim ExpFile As String

        Call CloseServers()
        Call ReadSettings()

        ExpFile = """File:" & PFolder & "\export.dat"""

        If (SCExpImp = False And PGExpImp = False) Then
            ProcID = Shell("" & TT & " /export_FILE /" & ExpFile & "
/ds:CART /ds_user:cart /ds_pwd:cart /LOG:" & PFolder &
"\Logs\exportLog.txt /SILENT", , True, )
            ProcID = Shell("" & TT & " /Import_FILE /" & ExpFile & "
/dt:" & ODBC & " /dt_user:" & DBUsenN & " /dt_pwd:" & DBPass & " /LOG:" &
PFolder & "\Logs\ImportLog.txt /SILENT", , True, )
        End If
        If (Prod = "SC" And SCExpImp = True) Then
            ProcID = Shell("" & TT & " /Import_FILE /" & ExpFile & "
/dt:" & ODBC & " /dt_user:" & DBUsenN & " /dt_pwd:" & DBPass & " /LOG:" &
PFolder & "\Logs\ImportLog.txt /SILENT", , True, )
        End If
        If (Prod = "PG" And PGExpImp = True) Then
            ProcID = Shell("" & TT & " /Import_FILE /" & ExpFile & "
/dt:" & ODBC & " /dt_user:" & DBUsenN & " /dt_pwd:" & DBPass & " /LOG:" &
PFolder & "\Logs\ImportLog.txt /SILENT", , True, )
        End If
    End Sub
    Private Sub StartServers()
        'Starts servers
        Dim SCSId As Boolean = False
        Dim UMSId As Boolean = False
        Dim PGSId As Boolean = False
        Dim WASId As Boolean = False

        Call ReadSettings()
        'MsgBox(UMS)
        'MsgBox(SCS)
        'MsgBox(PGS)
        'MsgBox(WAS)
        System.Diagnostics.Process.Start(UMS)
        System.Diagnostics.Process.Start(SCS)
        System.Diagnostics.Process.Start(PGS)
        System.Diagnostics.Process.Start(WAS)

        'Shell(UMS, , True, )
        'Shell(SCS, , True, )
        'Shell(PGS, , True, )
        'Shell(WAS, , True, )
    End Sub
```

```
    Private Sub CloseServers()
        'Calls KillEmAll.exe to kill all the servers and clients
        Dim KillEmAll As Integer

        KillEmAll = Shell("\\qpr-
118cart.qpr.com\CART\files\KillEmAll.exe", AppWinStyle.Hide, True, )

    End Sub

    Sub ExecuteScipt(ByVal ScriptName As String, ByVal DevC As String)


        Dim Script As String

        If (DevC = "SC") Then
            If File.Exists("" & PFolder & "\Scripts\" & ScriptName) Then
                File.Delete("" & PFolder & "\Scripts\" & ScriptName)
                My.Computer.FileSystem.CopyFile("\\qpr-
118cart.qpr.com\CART\scripts\" & ScriptName, "" & PFolder & "\Scripts\" &
ScriptName)
            Else
                My.Computer.FileSystem.CopyFile("\\qpr-
118cart.qpr.com\CART\scripts\" & ScriptName, "" & PFolder & "\Scripts\" &
ScriptName)
            End If
            'readScript = My.Computer.FileSystem.ReadAllText("" & PFolder
& "\Scripts\" & ScriptName)
            Script = ("" & PFolder & "\Scripts\" & ScriptName)
            Call WriteSettings("SC", Script)
            Shell("" & SCD & " """ & PFolder & "\Scripts\""" &
ScriptName, , True, )
        Else
            If File.Exists("" & PFolder & "\Scripts\" & ScriptName) Then
                File.Delete("" & PFolder & "\Scripts\" & ScriptName)
                My.Computer.FileSystem.CopyFile("\\qpr-
118cart.qpr.com\CART\scripts\" & ScriptName, "" & PFolder & "\Scripts\" &
ScriptName)
            Else
                My.Computer.FileSystem.CopyFile("\\qpr-
118cart.qpr.com\CART\scripts\" & ScriptName, "" & PFolder & "\Scripts\" &
ScriptName)
            End If
            'readScript = My.Computer.FileSystem.ReadAllText("" & PFolder
& "\Scripts\" & ScriptName)
            Script = ("" & PFolder & "\Scripts\" & ScriptName)
            Call WriteSettings("PG", Script)
            Shell("" & PGD & " """ & PFolder & "\Scripts\""" &
ScriptName, , True, )
        End If
    End Sub
    Sub ReadSettings()
        'Reads ini settings and places the values to variables
        Dim Setting As String

        If (iniFile = "") Then
            Call GetCartFolder()
        End If

        Dim ioFile As New StreamReader(iniFile)
        Dim ioLine As String
        ioLine = ioFile.ReadLine
        While Not ioLine = ""
            ioLine = ioFile.ReadLine
```

```
                Setting = LSet(ioLine, 7)
                If (Setting = "UMSPath") Then
                    UMS = Mid(ioLine, 9)
                ElseIf (Setting = "SCSPath") Then
                    SCS = Mid(ioLine, 9)
                ElseIf (Setting = "PGSPath") Then
                    PGS = Mid(ioLine, 9)
                ElseIf (Setting = "WASPath") Then
                    WAS = Mid(ioLine, 9)
                ElseIf (Setting = "SCDPath") Then
                    SCD = Mid(ioLine, 9)
                ElseIf (Setting = "PGDPath") Then
                    PGD = Mid(ioLine, 9)
                ElseIf (Setting = "SCDPort") Then
                    SCPort = Mid(ioLine, 9)
                ElseIf (Setting = "PGDPort") Then
                    PGPort = Mid(ioLine, 9)
                ElseIf (Setting = "TT_Path") Then
                    TT = Mid(ioLine, 9)
                ElseIf (Setting = "ODBCNam") Then
                    ODBC = Mid(ioLine, 9)
                ElseIf (Setting = "DBUserN") Then
                    DBUsenN = Mid(ioLine, 9)
                ElseIf (Setting = "DBUsrPW") Then
                    DBPass = Mid(ioLine, 9)
                End If

            End While


        End Sub
        Sub GetCartFolder()
            ' Checks where the CART is located in the system
            PFolder = My.Computer.FileSystem.CurrentDirectory
            iniFile = "" & PFolder & "\Cart.ini"
            Call CreateTestFolders()
        End Sub

        Sub ReadLog(ByVal sFile As String, ByVal sName As String)
            'Check the logs for errors
            Dim fileReader As String
            Dim FindError As String = "Error!"
            Dim FindOK As String = "All OK!"
            Dim ErrorFile As String
            Dim TimeStamp As String
            Dim FileNotFoundFile As System.IO.FileStream

            If My.Computer.FileSystem.FileExists("" & PFolder & "\Logs\" &
sFile) Then
                fileReader = My.Computer.FileSystem.ReadAllText("" & PFolder
& "\Logs\" & sFile)

                'If My.Computer.FileSystem.FileExists("C:\CART2\Logs\" &
sFile) Then
                'fileReader =
My.Computer.FileSystem.ReadAllText("C:\CART2\Logs\" & sFile)
                If fileReader.Contains(FindError) Then
                    Result = False
                    ResultStr = ResultStr + ", " & sName
                    My.Computer.FileSystem.CopyFile("" & PFolder & "\Logs\" &
sFile, "\\qpr-118cart.qpr.com\CART\Logs\" & Now & sFile)
                ElseIf fileReader.Contains(FindOK) Then
                    Call DeleteFiles(sFile)
```

```
            Else
                Result = False
                ResultStr = ResultStr + ", " & sName
                TimeStamp = Now
                ErrorFile = "\\qpr-118cart.qpr.com\CART\Logs\" & sFile &
Now & ".txt"
                ErrorFile = Replace(ErrorFile, Chr(58), Chr(46))
                'My.Computer.FileSystem.CopyFile("C:\CART2\" & sFile,
ErrorFile)
                My.Computer.FileSystem.CopyFile("" & PFolder & "\Logs\" &
sFile, ErrorFile)
            End If
        Else
            Result = False
            ResultStr = ResultStr + ", " & sName
            TimeStamp = Now
            ErrorFile = "\\qpr-118cart.qpr.com\CART\Logs\NOT FOUND" &
sFile & Now & ".txt"
            ErrorFile = Replace(ErrorFile, Chr(58), Chr(46))
            FileNotFoundFile = System.IO.File.Create(ErrorFile)
        End If
    End Sub

    Sub DeleteFiles(ByVal sFile As String)
        My.Computer.FileSystem.DeleteFile("" & PFolder & "\Logs\" &
sFile)
        'My.Computer.FileSystem.DeleteFile("C:\CART2\Logs\" & sFile)
    End Sub

    Sub TestResult()
        If (Result = True) Then
            MsgBox("All test passed without any errors")
        Else
            MsgBox("The following test found errors: " & ResultStr)
        End If
        ResultStr = ""
    End Sub
    Sub CreateTestFolders()
        ' Creates the folder hierarchy to directory, where CART is
located
        Dim Logs As String
        Dim Scripts As String
        Dim SBool As Boolean
        Dim LBool As Boolean

        Logs = "" & PFolder & "\Logs"
        Scripts = "" & PFolder & "\Scripts"

        LBool = DirExists(Logs)
        SBool = DirExists(Scripts)

        If (LBool = False) Then
            Directory.CreateDirectory(Logs)
        End If

        If (SBool = False) Then
            Directory.CreateDirectory(Scripts)
        End If
    End Sub
    Sub WriteSettings(ByVal sProd As String, ByVal Script As String)
        ' Modifies the tests scripts to correct values
        'Dim file As System.IO.StreamWriter
        Dim File1 As String
```

```
        Dim File2 As String
        Dim File3 As String
        Dim reader As String

        If (sProd = "SC") Then
            'file = My.Computer.FileSystem.OpenTextFileWriter(ExecScript,
True)
            'file.WriteLine("")
            'file.WriteLine("Sub Settings")
            'file.WriteLine("PortN = " & SCPort)
            'file.WriteLine("End Sub")
            'file.Close()

            reader = My.Computer.FileSystem.ReadAllText(Script)

            File1 = Replace(reader, "<PortN>", SCPort)
            File2 = Replace(File1, "<PFolder>", PFolder)
            File3 = Replace(File2, "<LogFolder>", Script)

            'MsgBox(File3)

            My.Computer.FileSystem.WriteAllText(Script, File3, False)
        Else
            'file = My.Computer.FileSystem.OpenTextFileWriter(ExecScript,
True)
            'file.WriteLine("")
            'file.WriteLine("Sub Settings")
            'file.WriteLine("PortN = " & PGPort)
            'file.WriteLine("End Sub")
            'file.Close()

            reader = My.Computer.FileSystem.ReadAllText(Script)

            File1 = Replace(reader, "<PortN>", PGPort)
            File2 = Replace(File1, "<PFolder>", PFolder)
            File3 = Replace(File2, "<LogFolder>", Script)

            My.Computer.FileSystem.WriteAllText(Script, File3, False)
        End If
    End Sub
    Sub IniExists()
        Dim IniEx As Boolean

        Call GetCartFolder()

        IniEx = My.Computer.FileSystem.FileExists(iniFile)

        If IniEx = False Then
            FileSystem.FileCopy("\\qpr-
118cart.qpr.com\CART\files\Cart.ini", iniFile)
        End If
    End Sub
    Sub DeleteOldLogs()

        If PFolder = "" Then
            Call GetCartFolder()
        End If

        For Each foundFile As String In
My.Computer.FileSystem.GetFiles(PFolder & "\Logs\")

            My.Computer.FileSystem.DeleteFile(foundFile,
FileIO.UIOption.OnlyErrorDialogs, FileIO.RecycleOption.DeletePermanently)
```

```
        Next
    End Sub
End Class
```

Ohessa on CART.ini – tiedoston vakioasetukset.

```
[CART settings: UMSPath, SCSPath, PGSPath, WASPath, SCDPath, PGDPath, SCDPort,
PGDPort, TT_Path, ODBCNam, DBUserN, DBUsrPw]
UMSPath="C:\Program Files\QPR 8.1\User Management System\QPR.UMS.Server.exe"
SCSPath="C:\Program Files\QPR 8.1\Scorecard Application Server\QPR.SC.Server.exe"
PGSPath="C:\Program Files\QPR 8.1\ProcessGuide Application Serv-
er\QPR.PG.Server.exe"
WASPath="C:\Program Files\QPR 8.1\Web Application Server\QPR.WAS.Server.exe"
SCDPath="C:\Program Files\QPR 8.1\Scorecard\QPR.SC.Client.exe"
PGDPath="C:\Program Files\QPR 8.1\ProcessGuide\QPR.PG.Client.exe"
SCDPort=20761
PGDPort=4521
TT_Path=C:\Program Files\QPR 8.1\Utilities\QPR.TransferTool.exe
ODBCNam=QPR_81_ACCESS_DEMO
DBUserN=
DBUsrPW=
```

```
<QPR_SCRIPT_FILE Language=vbscript Timeout=0>
Option Explicit

Dim iResult
Dim aServerModels
Dim PortN
'*************************************************
Sub Main

  Call Report("Starting to execute", "", 0)

  iResult = PGApplication.SetUIMode(0)

  Call ConnectToServerAndAuthenticate

  Call OpenServerModels

  Call CreateModel

  Call Report("PG_Smoke executed", "", 0)

  iResult = PGApplication.Quit(0)
End Sub
'*************************************************
Sub ConnectToServerAndAuthenticate
'This sub connect to server and authenticates with qpr/demo

  iResult = PGApplication.ConnectToServer("localhost", 4251)
  if (iResult <> 0) Then
    Call Report("ConnectToServerAndAuthenticate", "Could not connect to server", -1)
  End if
```

```
  iResult = PGApplication.Authenticate("qpr", "demo", "")
  if (iResult <> 0) Then
    Call Report("ConnectToServerAndAuthenticate", "Could not authenticate:", -1)
  End if
End Sub
'*****************************************************
Sub OpenServerModels
'This sub gets all the servermodels to an array and opens the models

  Dim i
  Dim ModelPath
  Dim oModel

  iResult = PGApplication.GetServerModels(aServerModels)
  if (iResult <> 0) Then
    Call Report("OpenServerModels", "", -1)
  End if

  for i = lbound(aServerModels) to uBound(aServerModels)
    if(aServerModels(i,3) <> 1) Then
      ModelPath = ""&aServerModels(i,9)

      iResult = PGApplication.OpenServerModel("\\Dentorex Group Processes", oModel)
      if (iResult <> 0) Then
        Call Report("OpenServerModels", "Could not open model "&ModelPath&": " , -1)
      End if

      iResult = PGModel.CloseModel(1)
    End if
  Next
End Sub
'*****************************************************
Sub CreateModel
' In this sub a new model is created. The model is based on Dentorex model.
```

```
' Few elements will be created to flowchart. After that, the model will be saved as a server model and deleted.

  Dim oModel
  Dim ElemId1
  Dim ElemId2
  Dim ElemId3
  Dim FlowId
  Dim Params
  Dim Id
  Dim i
  Dim lSubProcess
  Dim lActivities
  Dim SubType
  Dim ActType
  Dim lFlows
  Dim FlowType

  iResult = PGApplication.CreateModel(True, "\\Dentorex Group Processes", oModel)
  if (iResult <> 0) Then
    Call Report("CreateModel", "Could not create model: ", -1)
  End if

  iResult = PGModel.Find("ELEMENTTYPES=ALLSUBPROCESSES", "", "", lSubProcess)
  iResult = PGModel.Find("ELEMENTTYPES=ALLACTIVITIES", "", "", lActivities)
  iResult = PGModel.Find("ELEMENTTYPES=ALLFLOWS", "", "", lFlows)

  SubType = lSubProcess(0)
  ActType = lActivities(0)
  FlowType = lFlows(0)

  iResult = PGModel.CreateElement(""&SubType, "Sub1", "", ElemId1)
  if (iResult <> 0) Then
    Call Report("CreateModel", "Could not create element1: ", -1)
```

```
End if

Params = "ParentElement=ElemId1; InstanceGraphicalProperties=100,100,100,100"

iResult = PGModel.CreateElement(""&ActType, "Activity1", Params, ElemId2)
if (iResult <> 0) Then
  Call Report("CreateModel", "Could not create element2: ", -1)
End if

Params = "ParentElement=ElemId1; InstanceGraphicalProperties=300,300,100,100"

iResult = PGModel.CreateElement(""&ActType, "Activity2", Params, ElemId3)
if (iResult <> 0) Then
  Call Report("CreateModel", "Could not create element3: ", -1)
End if

Params = "From=ElemId2;To=ElemId3"

iResult = PGModel.CreateElement(""&FlowType, "Flow 1", Params, FlowId)
if (iResult <> 0) Then
  Call Report("CreateModel", "Could not create flow:", -1)
End if

iResult = PGModel.SaveServerModelAs("\\SmokeTestModel")
if (iResult <> 0) Then
  Call Report("CreateModel", "Could not save model: ", -1)
End if

for i = 0 to 3
  if (i = 0) Then
    Id = FlowId
  elseif(i = 1) Then
    Id = ElemId3
  elseif(i = 2) Then
    Id = ElemId2
```

```
    else
      Id = ElemId1
    End if

    iResult = PGModel.DeleteElement(Id,"")
    if (iResult <> 0) Then
      Call Report("CreateModel", "Could not delete element: ", -1)
    End if
  Next

  iResult = PGModel.DeleteModel
End Sub
'*************************************************
Sub Report(log1, log2, error)
Dim objFSO, objFolder, objShell, objTextFile
  Dim dDate
  Dim dTime
  Dim strFile
  Dim objFSO2, objFolder2, objShell2, strDirectory2

  strDirectory2 = "<PFolder>"

  if (error = -1 OR error = "") Then

  End if

  strFile = "<PFolder>\Logs\PG_SMOKE_Log.txt"

  dTime = Time
  dDate = Date

  Set objFSO = CreateObject("Scripting.FileSystemObject")

  If objFSO.FileExists(""&strFile) = 0 Then
    Set objTextFile = objFSO.CreateTextFile(""&strFile)
```

```
    objTextFile.WriteLine(""&dTime &"                    "&dDate &"  "&log1 &" "&log2 +
PGApplication.GetErrorMessage(iResult))
    if (completed = False) Then
      objTextFile.WriteLine(""&dTime &"                  "&dDate &"  "&log1 &" Er-
ror:"&log2 + PGApplication.GetErrorMessage(iResult))
    PGApplication.Quit
   'End if
    objTextFile.Close
  Else
    Set objTextFile = objFSO.OpenTextFile(""&strFile, 8, True)
    objTextFile.WriteLine(""&dTime &"                    "&dDate &"  "&log1 &" "&log2 +
PGApplication.GetErrorMessage(iResult))
    if (completed = False) Then
     objTextFile.WriteLine(""&dTime &"                  "&dDate &"  "&log1 &" Er-
ror:"&log2 + PGApplication.GetErrorMessage(iResult))
      PGApplication.Quit
    End if
    objTextFile.Close
  End If

  if (log1 = "PG_Smoke executed") Then
    'if (completed = False) Then
    '  Set objTextFile = objFSO.OpenTextFile(""&strFile, 8, True)
    '  objTextFile.WriteLine(""&dTime &"                "&dDate &"  Error: Errors occurred
while executing script")
    '  objTextFile.Close
    'else
      Set objTextFile = objFSO.OpenTextFile(""&strFile, 8, True)
      objTextFile.WriteLine(""&dTime &"                "&dDate &"  All OK!")
      objTextFile.Close
    'End if
  End if
End Sub
'*****************************************************
```

```
<QPR_SCRIPT_FILE Language=vbscript Timeout=0>
Option Explicit

Dim iResult

Sub Main

  Call Report("Starting to execute SC_Smoke","" ,0)

  iResult = SCApplication.SetUIMode(0)

  Call ConnectToServerAndAuthenticate()

  Call OpenServerModels()

  Call CreateAnDeleteModel

  Call Report("SC_Smoke Executed","" ,0)
  SCApplication.Quit
End Sub
'**************************************
Sub ConnectToServerAndAuthenticate

  iResult = SCApplication.ConnectToServer("localhost", <PortN>)
  if (iResult <> 0) Then
    Call Report("ConnectToServerAndAuthenticate", "Cannot connect to server: " + SCAppli-
cation.GetErrorMessage(iResult), -1)
  End if

  iResult = SCApplication.Authenticate("qpr", "demo", "")
  if (iResult <> 0) Then
    Call Report("ConnectToServerAndAuthenticate", "Cannot authenticate: " + SCApplica-
tion.GetErrorMessage(iResult), -1)
  End If
```

```
End Sub
'**************************************
Sub OpenServerModels

Dim aModels
Dim oModel
Dim i

  iResult = SCApplication.GetModels(aModels)
  if (iResult <> 0) Then
    Call Report("OpenServerModels", "Could not get models: " + SCApplica-
tion.GetErrorMessage(iResult), -1)
  End if

  for i = lBound(aModels) to uBound(aModels)
    iResult = SCApplication.OpenModel(""&aModels(i,0), oModel)
    if (iResult <> 0) Then
      Call Report("OpenServerModels", "Could not open model '"&aModels(i,0)&"' : " +
SCApplication.GetErrorMessage(iResult), -1)
    End if

    iResult = SCModel.CloseModel
  Next

End Sub
'**************************************
Sub CreateAnDeleteModel

Dim oModel
Dim SCId1
Dim SCId2
Dim ElemId1
Dim ElemId2

  iResult = SCApplication.CreateModel("TestModel1", "Government", oModel)
```

```
  IF (iResult <> 0) Then
    Call Report("CreateAndDeleteModel", "Could not create model: " + SCApplica-
tion.GetErrorMessage(iResult), -1)
  End if

  iResult = SCModel.CreateScorecard("TestSC", "TSC1", 0, SCId1)

End Sub
'**************************************
Sub Report(log1, log2, error)
Dim objFSO, objFolder, objShell, objTextFile
  Dim dDate
  Dim dTime
  Dim strFile
  Dim objFSO2, objFolder2, objShell2, strDirectory2

  strDirectory2 = "<PFolder>"

  if (error = -1 OR error = "") Then

  End if

  strFile = "<PFolder>\Logs\SC_SMOKE_Log.txt"

  dTime = Time
  dDate = Date

  Set objFSO = CreateObject("Scripting.FileSystemObject")

  If objFSO.FileExists(""&strFile) = 0 Then
    Set objTextFile = objFSO.CreateTextFile(""&strFile)
    objTextFile.WriteLine(""&dTime &"                    "&dDate &"  "&log1 &" "&log2 +
SCApplication.GetErrorMessage(iResult))
    'if (completed = False) Then
```

```
    '  objTextFile.WriteLine(""&dTime &"                    "&dDate &"  "&log1 &" Er-
ror:"&log2 + SCApplication.GetErrorMessage(iResult))
    '  SCApplication.Quit
    'End if
    objTextFile.Close
  Else
    Set objTextFile = objFSO.OpenTextFile(""&strFile, 8, True)
    objTextFile.WriteLine(""&dTime &"                    "&dDate &"  "&log1 &" "&log2 +
SCApplication.GetErrorMessage(iResult))
    'if (completed = False) Then
    '  objTextFile.WriteLine(""&dTime &"                    "&dDate &"  "&log1 &" Er-
ror:"&log2 + SCApplication.GetErrorMessage(iResult))
    '  SCApplication.Quit
    'End if
    objTextFile.Close
  End If

  if (log1 = "SC_Smoke Executed") Then
    'if (completed = False) Then
    '  Set objTextFile = objFSO.OpenTextFile(""&strFile, 8, True)
    '  objTextFile.WriteLine(""&dTime &"                    "&dDate &"  Error: Errors occurred
while executing script")
    '  objTextFile.Close
    'else
      Set objTextFile = objFSO.OpenTextFile(""&strFile, 8, True)
      objTextFile.WriteLine(""&dTime &"                    "&dDate &"  All OK!")
      objTextFile.Close
    'End if
  End if
End Sub
'*****************************************************
```

```
<QPR_SCRIPT_FILE language = vbscript TimeOut=0>
Option Explicit

Private oModel
Private stUser
private stDeleteUser
Private stUserPw
Private n
Private iId
Private iId2
Private aId
Private zId
Private xId
Private i
Private BaseModel
Private stPath
Private ReplacedModel
Private ApiResult

  stUserPw = ""
  BaseModel = "UserRightsTest"
  ReplacedModel = "ReplaceModel"
  stPath = "<PFolder>\Logs"
  n = 0
'*****************************************************************************
'*****************************************************************************
Sub Main()

  Dim iResult
  Dim iActionId
  Dim iCopyActionId
  Dim objFSO, objFolder, objShell
```

```
Call APILog("Main", "Starting to run PG_Intermediate", 0, iResult)

Set objFSO = CreateObject("Scripting.FileSystemObject")
if objFSO.FolderExists(stPath) Then
  Set objFolder = objFSO.GetFolder(stPath)
  Call APILog("Main", "Folder already created", 0, 0)
else
  Set objFolder = objFSO.CreateFolder(stPath)
  Call APILog("Main", "Created folder", 0, 0)
End if

Call APILog("Main", "Setting UI mode (0)", 0, iResult)
PGApplication.SetUIMode(0)

Call ConnectToServer()

Call GetGraphsAndVersion()

Call CreateBaseModelAndReplaceModel(oModel)

do while n < 9
  if (n = 0 OR n = 1 OR n = 2) then
    stUser = "ViewUser"
  elseif (n = 3 OR n = 4 OR n = 5) then
    stUser = "UpdateUser"
  elseif (n = 6 OR n = 7 OR n = 8) then
    stUser = "AdminUser"
  End if

  Call Authenticate()

  Call CreateModel(oModel)

  if(n = 0 OR n = 3 OR n = 6) then
    stDeleteUser = "ViewUser"
```

```
    Call AuthenticateDeleteUser()
  elseif(n = 1 OR n = 4 OR n = 7) then
    stDeleteUser = "UpdateUser"
    Call AuthenticateDeleteUser()
  elseif(n = 2 OR n = 5 OR n = 8) then
    stDeleteUser = "AdminUser"
    Call AuthenticateDeleteUser()
  End if

  Call ExportImportXML()

  'Call ReplaceModel()

  Call DeleteElement(oModel)

  n = n + 1
  Call APILog("Main", ""&BaseModel &" loop number "&n , 0, iResult)
loop

'Call DeleteModels(ReplacedModel)

Call DeleteModels(BaseModel)

'if objFSO.FolderExists(stPath) Then
'  objFSO.DeleteFolder(stPath)
'  Call APILog("Main", "Deleted directory", 0, 0)
'else
'  Call APILog("Main", "Directory not found", -1, 0)
'End if

Call APILog("Main", "Completed PG_Intermediate", 0, iResult)

PGApplication.Quit(1)

End Sub
```

```
'*****************************************************************************
'*****************************************************************************
Sub ConnectToServer()
  Dim iResult

  iResult = PGApplication.ConnectToServer("localhost", <PortN>)
  If (iResult <> 0) Then
    Call APILog("ConnectToServer", "Could not connect to server: ", -1, iResult)
  End If

End Sub
'*****************************************************************************
'*****************************************************************************
Sub Authenticate()
  Dim iResult

  Call APILog("Authenticating user: "&stUser ,
"++++++++++++++++++++++++++++++++++++++++++++++++", 0, iResult)

  iResult = PGApplication.Authenticate(stUser, ""&stUserPw, "")
  If (iResult <> 0) Then
    Call APILog("Authenticate", "Could not authenticate user: ", -1, iResult)
  End If
End Sub
'*****************************************************************************
'*****************************************************************************
Sub AuthenticateDeleteUser()
  Dim iResult

  Call APILog("Authenticating DeleteUser: "&stDeleteUser,
"++++++++++++++++++++++++++++++++++++++++++++++++", 0, iResult)

  iResult = PGApplication.Authenticate(""&stDeleteUser, ""&stUserPw, "")
  If (iResult <> 0) Then
```

```
  Call APILog("ConnectToServerAndAuthenticate", "Could not authenticate user: ", -1, iResult)
 End If
End Sub
'*****************************************************************************
'*****************************************************************************
Sub GetGraphsAndVersion()
 Dim iResult
 Dim ViewId
 Dim TestModel
 Dim ModelList
 Dim i
 Dim ModelName
 Dim BranchId

 iResult = PGApplication.Authenticate("AdminUser", "", "")
 if (iResult <> 0) Then
  Call APILog("GetGraphs", "Could not auteticate AdminUser: ", -1, iResult)
 End if

 iResult = PGApplication.GetServerModels(ModelList)
 if (iResult <> 0) Then
  Call APILog("GetGraphs", "Could not get server models: ", -1, iResult)
 End if

 for i = lBound(ModelList,1) to uBound(ModelList,1)
  if (""&ModelList(i,3) = "7") Then
   ModelName = ""&ModelList(i,9)
  End if
 Next

 iResult = PGApplication.OpenServerModel(""&ModelName, TestModel)
 if (iResult <> 0) Then
  Call APILog("GetGraphs", "Could not open model: ", -1, iResult)
 End if
```

```
  iResult = PGModel.CreateBranch("TestBranch", "", BranchId)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not create branch: ", -1, iResult)
  End if

  iResult = PGModel.GetActive("VIEWSETTINGS", ViewId)
  if (iResult <> 0) Then
    Call APILog("GetGraphs", "Could not get view setting id: ", -1, iResult)
  End if

  iResult = PGModel.GetGraph(0, "C:\Regression\TestGraph.jpg",
"TYPE=JPG;VIEWSETTINGS="&ViewId)
  if (iResult <> 0) Then
     Call APILog("GetGraphs", "Could not get graph: ", -1, iResult)
  End if

  iResult = PGModel.CloseModel(0)
End Sub
'*****************************************************************************
'*****************************************************************************
Sub CreateBaseModelAndReplaceModel(oModel)
  Dim iResult
  Dim ElemId
  Dim ModelId
  Dim arrRights
  Dim i


  Call APILog("CreateBaseModelAndReplaceModel", "Starting to create BaseModel and ReplaceModel", 0, iResult)

   'Movid this to GetGraphsAndVersion
  'iResult = PGApplication.Authenticate("AdminUser", "", "")
```

```
'if (iResult <> 0) Then
'  Call APILog("CreateBaseModelAndReplaceModel", "Could not auteticate AdminUser: ", -1, iResult)
'End if

iResult = PGApplication.CreateModel(True, "", ModelId)
if (iResult <> 0) Then
  Call APILog("CreateBaseModelAndReplaceModel", "Could not create a new model: ", -1, iResult)
End if

iResult = PGModel.SetModelProperty("ModelVersionManagement", 1)
if (iResult <> 0) Then
  Call APILog("CreateBaseModelAndReplaceModel", "Could not set model property: ", -1, iResult)
End if

iResult = PGModel.SaveServerModelAs("\\"&ReplacedModel)
if (iResult <> 0) Then
  Call APILog("CreateBaseModelAndReplaceModel", "Could not save replace model: ", -1, iResult)
End if

'iResult = PGModel.CreateBranch("TestBranch", "", BranchId)
'if (iResult <> 0) Then
'  Call APILog("CreateBaseModelAndReplaceModel", "Could not create branch: ", -1, iResult)
'End if

iResult = PGModel.CloseModel(0)
if (iResult <> 0) Then
  Call APILog("CreateBaseModelAndReplaceModel", "Could not close model: ", -1, iResult)
End if
```

```
  Call APILog("CreateBaseModelAndReplaceModel", "Created ReplaceModel. Starting to
create BaseModel", 0, 0)

  iResult = PGApplication.CreateModel(True, "", oModel)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not create a new model: ", -1,
iResult)
  End if

  iResult = PGModel.CreateElement("Subprocess", "Test SubProcess", "", ElemId)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not createElement: ", -1, iRe-
sult)
  End if

  Call SetObjRights(0, "ViewUser", "ApplyToChildLevels=true", 1)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not createElement: ", -1, iRe-
sult)
  End if

  Call SetObjRights(ElemId, "ViewUser", "ApplyToChildLevels=false", 0)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not set object rights 1: ", -1,
iResult)
  End if

  Call SetObjRights(0, "UpdateUser", "ApplyToChildLevels=true", 3)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not set object rights 2: ", -1,
iResult)
  End if

  Call SetObjRights(0, "AdminUser", "ApplyToChildLevels=true", 0)
  if (iResult <> 0) Then
```

```
    Call APILog("CreateBaseModelAndReplaceModel", "Could not set object rights 3: ", -1,
iResult)
  End if

  Call GetObjRights(ElemId)

  Call GetObjRights(0)

  Dim outValueVersion

  iResult = PGModel.SetModelProperty("MODELVERSIONMANAGEMENT", 0)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not set version management: ",
-1, iResult)
  End if

  iResult = PGModel.GetModelProperty("MODELVERSIONMANAGEMENT", outValu-
eVersion)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not get version management: ",
-1, iResult)
  End if

  if (""&outValueVersion <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Wrong version management: ", -1,
iResult)
  End if

  iResult = PGModel.SaveServerModelAs("\\"&BaseModel)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not save base model: ", -1,
iResult)
  End if

  Call GetSetModelProperty()
```

```
  iResult = oModel.CloseModel(0)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not close/save BaseModel: ", -
1, iResult)
  End if
End Sub
'*****************************************************************************
'*****************************************************************************
Sub CreateModel(oModel)
  Dim iResult
  Dim x
  Dim y
  Dim pass
  Dim fail
  Dim OrgId
  Dim lSubProcess
  Dim lActivities
  Dim SubType
  Dim ActType

  iResult = PGApplication.OpenServerModel("\\"&BaseModel, oModel)
  If (iResult <> 0) Then
    Call APILog("CreateModel", "Could not open server model: ", -1, iResult)
  else
    Call APILog("CreateModel", "User: "&stUser&" opened server model: ", 0, iResult)
  End If

  iResult = PGModel.Find("ELEMENTTYPES=ALLSUBPROCESSES", "", "", lSubPro-
cess)
  iResult = PGModel.Find("ELEMENTTYPES=ALLACTIVITIES", "", "", lActivities)

  SubType = lSubProcess(0)
  ActType = lActivities(0)
```

```
  iResult = PGModel.CreateElement(SubType, "Main 1", "InstanceGraphicalProper-
ties=100,100,100,100;", iId)
  If (iResult <> 0) Then
    if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
      fail = -1
    elseif(stUser = "ViewUser") Then
      fail = 0
    End if
    Call APILog("CreateModel", "Could not create element 'Main 1': ", fail, iResult)
  else
    if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
      pass = 0
    elseif(stUser = "ViewUser")Then
      pass = -1
    End if
    Call APILog("CreateModel", "User: "&stUser&" created 1st Subprocess", pass, iResult)
  End If
  for y = 0 To 1
    iResult = PGModel.CreateElement(SubType, "Sub"& y, "ParentEle-
ment=iId;InstanceGraphicalProperties=150,150,150,150;", aId)
    If (iResult <> 0) Then
      if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
        fail = -1
      elseif(stUser = "ViewUser") Then
        fail = 0
      End if
      Call APILog("CreateModel", "Could not create element 'Sub"&y &"': ", fail, iResult)
    End If
    for x = 0 To 1
      iResult = PGModel.CreateElement(ActType, "Activity 1", "ParentEle-
ment=aId;InstanceGraphicalProperties=170,170,170,170;", zId)
      If (iResult <> 0) Then
        if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
          fail = -1
        elseif(stUser = "ViewUser") Then
```

```
        fail = 0
       End if
       Call APILog("CreateModel", "Could not create activity 'Activity 1': ", fail, iResult)
      End If

      iResult = PGModel.CreateElement(ActType, "Activity 2", "ParentElement="&aId&";",
xId)
      If (iResult <> 0) Then
       if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
        fail = -1
       elseif(stUser = "ViewUser") Then
        fail = 0
       End if
       Call APILog("CreateModel", "Could not create activity 'Activity 2': ", fail, iResult)
      End If
    Next
  Next

  iResult = PGModel.CreateElement(SubType, "Main 2", "InstanceGraphicalProper-
ties=70,70,70,70;", iId2)
  If (iResult <> 0) Then
    if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
     fail = -1
    elseif(stUser = "ViewUser") Then
     fail = 0
    End if
    Call APILog("CreateModel", "Could not create element 'Main 2': ", fail, iResult)
  else
    if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
     pass = 0
    elseif(seUser = "ViewUser")Then
     pass = -1
    End if
    Call APILog("CreateModel", "User: "&stUser&" created 2nd Subprocess", pass, iResult)
  End If
```

```
  for y = 0 To 1
    iResult = PGModel.CreateElement(SubType, "Sub"& y, "ParentEle-
ment=iId2;InstanceGraphicalProperties=70,70,70,70;", aId)
    If (iResult <> 0) Then
      if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
        fail = -1
      elseif(stUser = "ViewUser") Then
        fail = 0
      End if
      Call APILog("CreateModel", "Could not create element 'Sub"&y &"': ", fail, iResult)
    End If
    for x = 0 To 1
      iResult = PGModel.CreateElement(ActType, "Activity 1", "ParentEle-
ment=aId;InstanceGraphicalProperties=170,170,170,170;", zId)
      If (iResult <> 0) Then
        if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
          fail = -1
        elseif(stUser = "ViewUser") Then
          fail = 0
        End if
        Call APILog("CreateModel", "Could not create activity 'Activity 1': ", fail, iResult)
      End If

      iResult = PGModel.CreateElement(ActType, "Activity 2", "ParentEle-
ment=aId;InstanceGraphicalProperties=50,50,50,50;", xId)
      If (iResult <> 0) Then
        if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
          fail = -1
        elseif(stUser = "ViewUser") Then
          fail = 0
        End if
        Call APILog("CreateModel", "Could not create activity 'Activity 2': ", fail, iResult)
      End If
    Next
  Next
```

```
iResult = PGModel.SaveModel()
if (iResult <> 0) Then
  Call APILog("CreateModel", "Could not save server model: ", -1, iResult)
else
  Call APILog("CreateModel", "User: "&stUser&" saved server model", 0, iResult)
End if

iResult = PGModel.SaveFileModelAs(""&stPath&"\"&stUser&n&".pmf")
if (iResult <> 0) Then
  if(stUser = "AdminUser") Then
    fail = -1
  elseif(stUser = "ViewUser" OR stUser = "UpdateUser") Then
    fail = 0
  End if
  Call APILog("CreateModel", "Could not save file model: ", fail, iResult)
else
  if(stUser = "AdminUser") Then
    pass = 0
  else
    pass = -1
  End if
  Call APILog("CreateModel", "User: "&stUser&" saved file model", pass, iResult)
End if

iResult = PGModel.CloseModel(0)
If (iResult <> 0) Then
  Call APILog("CreateModel", "Could not close server model: ", -1, iResult)
else
  Call APILog("CreateModel", "Closed model", 0, iResult)
End If

End Sub
'*******************************************************************************
'*******************************************************************************
```

```
Sub ExportImportXML()
  Dim ExportModel
  Dim iResult
  Dim pass
  Dim fail

  Call APILog("ExportImportXML", "Exporting model to XML", 0, iResult)

  iResult = PGApplication.OpenServerModel("\\"&BaseModel, oModel)
  If (iResult <> 0) Then
    Call APILog("ExportImportXML", "Could not open server model: ", -1, iResult)
  else
    Call APILog("ExportImportXML", "User: "&stDeleteUser&" opened server model: ", 0,
iResult)
  End If

  iResult = PGModel.ExecuteXMLTask("Export All.PGT",
""&stPath&"\"&stUser&n&".xml", "", "")
  if (iResult <> 0) Then
    if(stDeleteUser = "ViewUser") Then
      fail = 0
    elseif(stDeleteUser = "AdminUser"  OR stDeleteUser = "UpdateUser")Then
      fail = -1
    End if
    Call APILog("ExportImportXML", ""&stDeleteUser&" Could not export model to XML:
", fail, iResult)
  else
    if(stDeleteUser = "ViewUser")Then
      pass = -1
    else
      pass = 0
    End if
    Call APILog("ExportImportXML", "User: "&stDeleteUser&" exported model to XML",
pass, iResult)
  End if
```

```
 iResult = PGModel.CloseModel(1)
 if (iResult <> 0) Then
  Call APILog("ExportImportXML", "Could not close model: ", -1, iResult)
 else
  Call APILog("ExportImportXML", "Closed model", 0, iResult)
 End if

 Call APILog("ExportImportXML", "Importing model from XML", 0, iResult)

 iResult = PGApplication.OpenServerModel("\\"&BaseModel, oModel)
 If (iResult <> 0) Then
  Call APILog("ExportImportXML", "Could not open server model: ", -1, iResult)
 else
  Call APILog("ExportImportXML", "User: "&stDeleteUser&" opened server model: ", 0,
iResult)
 End If

 iResult = PGModel.ExecuteXMLTask("Import All.PGT",
""&stPath&"\"&stUser&n&".xml", "", "ReplaceCreate")
 if (iResult <> 0) Then
  if(stDeleteUser = "ViewUser" OR stDeleteUser = "UpdateUser") Then
   fail = 0
  elseif (stDeleteUser = "AdminUser") Then
   fail = -1
  End if
  Call APILog("ExportImportXML", "Could not import model: ", fail, iResult)
 else
  if(stDeleteUser = "ViewUser" OR stDeleteUser = "UpdateUser") Then
   pass = -1
  elseif(stDeleteUser = "AdminUser")Then
   pass = 0
  End if
  Call APILog("ExportImportXML", "User: "&stDeleteUser&" imported model from
XML", pass, iResult)
```

```
  End if

  iResult = PGModel.CloseModel(1)
  if (iResult <> 0) Then
    Call APILog("ExportImportXML", "Could not close model: ", -1, iResult)
  End if

End sub
'*****************************************************************************
'*****************************************************************************
Sub ReplaceModel()
  Dim iResult
  Dim pass
  Dim fail

  Call APILog("ReplaceModel", "Starting to replace model", 0, iResult)

  iResult = PGApplication.OpenServerModel("\\"&BaseModel, oModel)
  If (iResult <> 0) Then
    Call APILog("ReplaceModel", "Could not open server model: ", -1, iResult)
  else
    Call APILog("ReplaceModel", "User: "&stDeleteUser&" opened server model", 0, iResult)
  End If

  iResult = PGModel.ReplaceModel("\\"&ReplacedModel, "ReplaceUserRights=TRUE")
  if (iResult <> 0)Then
    if (stDeleteUser = "ViewUser" OR stDeleteUser = "UpdateUser") Then
      fail = 0
    elseif (stDeleteUser = "AdminUser") Then
      fail = -1
    End if
    Call APILog("ReplaceModel", "Could not replace model: ", fail, iResult)
  else
    if (stDeleteUser = "ViewUser" OR stDeleteUser = "UpdateUser") Then
      pass = -1
```

```
  elseif(stDeleteUser = "AdminUser")Then
    pass = 0
  End if
  Call APILog("ReplaceModel", "User: "&stDeleteUser&" replaced model", pass, iResult)
 End if

 iResult = PGModel.CloseModel(1)
 if (iResult <> 0) Then
  Call APILog("ReplaceModel", "Could not close model: ", -1, iResult)
 End if

End sub
'*****************************************************************************
'*****************************************************************************
Sub DeleteElement(oModel)
 Dim iResult
 Dim pass
 Dim fail

 iResult = PGApplication.OpenServerModel("\\"&BaseModel, oModel)
 if (iResult <> 0) Then
  Call APILog("DeleteElement", "Could not open server model: ", -1, iResult)
 else
  Call APILog("DeleteElement", "User: "&stDeleteUser&" opened server model", 0, iResult)
 End If

 iResult = PGModel.DeleteElement(iId, "")
 If (iResult <> 0) Then
  if (stDeleteUser = "ViewUser") Then
    pass = -1
    fail = 0
  elseif (stDeleteUser = "UpdateUser" OR stDeleteUser = "AdminUser") Then
    if (stUser = "AdminUser" OR stUser = "UpdateUser") Then
      fail = -1
    elseif (stUser = "ViewUser") Then
```

```
      fail = 0
    End if
  End if
  Call APILog("DeleteElement", "Could not delete 1st Subprocess: ", fail, iResult)
 else
  if (stDeleteUser = "ViewUser") Then
    pass = -1
  else
    pass = 0
  End if
  Call APILog("DeleteElement", "User: "&stDeleteUser&" deleted 1st Subprocess", pass,
iResult)
 End If

 iResult = PGModel.DeleteElement(iId2, "")
 If (iResult <> 0) Then
  if (stDeleteUser = "ViewUser") Then
    fail = 0
  elseif (stDeleteUser = "UpdateUser" OR stDeleteUser = "AdminUser") Then
    if (stUser = "AdminUser" OR stUser = "UpdateUser") Then
      fail = -1
    elseif (stUser = "ViewUser") Then
      fail = 0
    End if
  End if
  Call APILog("DeleteElement", "Could not delete 2nd Subprocess: ", fail, iResult)
 else
  if (stDeleteUser = "ViewUser") Then
    pass = -1
  elseif(stDeleteUser = "UpdateUser" OR stDeleteUser = "AdminUser") Then
    if (stUser = "AdminUser" OR stUser = "UpdateUser")Then
      pass = 0
    elseif(stUser = "ViewUser")Then
      pass -1
    End if
```

```
    End if
    Call APILog("DeleteElement", "User: "&stDeleteUser&" deleted 2nd Subprocess", pass,
iResult)
  End If

  iResult = PGModel.CloseModel(1)
  If (iResult <> 0) Then
    Call APILog("DeleteElement", "Could not close server model: ", -1, iResult)
  End If
End Sub
'*****************************************************************************
'*****************************************************************************
Sub DeleteModels(Model)
  Dim iResult
  Dim i
  Dim x
  Dim sModel
  Dim Test

  for i = 0 to 2
    if (i = 0) Then
      stDeleteUser = "ViewUser"
      Call AuthenticateDeleteUser()
    elseif (i = 1) Then
      stDeleteUser = "UpdateUser"
      Call AuthenticateDeleteUser()
    else
      stDeleteUser = "AdminUser"
      Call AuthenticateDeleteUser()
    End if

    iResult = PGApplication.OpenServerModel("\\"&Model, oModel)
    if (iResult <> 0) Then
      Call APILog("DeleteModels", "Could not open "&Model&": ", -1, iResult)
    End if
```

```
    iResult = PGModel.DeleteModel
    if (iResult <> 0) Then
      if (stDeleteUser = "ViewUser" OR stDeleteUser = "UpdateUser") Then
        Call APILog("DeleteModels", ""&stDeleteUser&" could not delete model", 0, 0)
      else
        Call APILog("DeleteModels", "AdminUser could not delete model: ", -1, iResult)
      End if
    else
      if (stDeleteUser = "ViewUser" OR stDeleteUser = "UpdateUser") Then
        Call APILog("DeleteModels", ""&stDeleteUser&" could delete model "&Model, -1, iRe-
sult)
      else
        Call APILog("DeleteModels", "AdminUser deleted model "&sModel, 0, 0)
      End if
    End if
  Next
End Sub
'*****************************************************************************
'*****************************************************************************
Sub SetObjRights(ElemId, User, Param, Rights)
  Dim iResult

  iResult = PGModel.SetObjectRights(ElemId, User, Param, Rights)
  if (iResult <> 0) Then
    if (""&User = "AdminUser") Then
      Call APILog("SetObjRights", "Could not set rights for AdminUser", 0, 0)
    else
      Call APILog("SetObjRights", "Could not set object rights to "&User&": ", -1, iResult)
    End if
  elseif (iResult = 0 AND ""&User = "AdminUser") Then
    Call APILog("SetObjRights", "Object rights set for AdminUser!!!", -1, 0)
  End if
End Sub
'*****************************************************************************
```

```
'*****************************************************************************
Sub GetObjRights(ElemId)
  Dim iResult
  Dim i
  Dim arrRights

  iResult = PGModel.GetObjectRights(ElemId, "IncludeInheritedRights=false", arrRights)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not get object rights: ", -1, iResult)
  End if

  for i = lbound(arrRights, 1) to uBound(arrRights, 1)
    if (""&arrRights(i,1) = "ViewUser") Then
      if (ElemId = 0) Then
        if (""&arrRights(i,3) = 1) Then
          Call APILog("GetObjRights", "ViewUser has correct rights to main process level", 0, 0)
        else
          Call APILog("GetObjRights", "ViewUser has incorrect rights to main process level. AccessLevel: "&arrRights(i,3), -1, 0)
        End if
      else
        if (""&arrRights(i,3) = 0) Then
          Call APILog("CreateBaseModelAndReplaceModel", "ViewUser has correct rights to subprocess", 0, 0)
        else
          Call APILog("CreateBaseModelAndReplaceModel", "ViewUser has incorrect rights to subprocess. AccessLevel: "&arrRights(i,3), -1, 0)
        End if
      End if
    elseif(""&arrRights(i,1) = "UpdateUser") Then
      if (""&arrRights(i,3) = 3) Then
        Call APILog("CreateBaseModelAndReplaceModel", "UpdateUser has correct rights to subprocess", 0, 0)
      else
```

```
      Call APILog("CreateBaseModelAndReplaceModel", "UpdateUser has incorrect rights to
subprocess. AccessLevel: "&arrRights(i,3), -1, 0)
     end if
   elseif(""&arrRights(i,1) = "AdminUser") Then
     if (""&arrRights(i,3) = 4) Then
      Call APILog("CreateBaseModelAndReplaceModel", "AdminUser has correct rights to
subprocess", 0, 0)
     else
      Call APILog("CreateBaseModelAndReplaceModel", "AdminUser has incorrect rights to
subprocess. AccessLevel: "&arrRights(i,3), -1, 0)
     End if
   End if
  Next
End Sub
'*****************************************************************************
'*****************************************************************************
Sub GetSetModelProperty()
  Dim iResult
  Dim number
  Dim z
  Dim PropertyName(6)
  Dim PropertyValue(3)
  Dim outValue

  Call APILog("GetSetModelProperty", "Starting to set/get model properties", 0, 0)

  PropertyName(0) = "VersionNumber"
  PropertyName(1) = "Status"
  PropertyName(2) = "Author"
  PropertyName(3) = "Comment"
  PropertyName(4) = "LastChanged"
  PropertyName(5) = "ModelCreated"
  PropertyName(6) = "LastSaved"
  PropertyValue(0) = "1.123"
  PropertyValue(1) = "0"
```

```
  PropertyValue(2) = "qpr"
  PropertyValue(3) = "This comment is created via API"

  for z = 0 to 3
    iResult = PGModel.SetModelProperty(""&PropertyName(z), ""&PropertyValue(z))
    if (iResult <> 0) Then
      Call APILog("GetSetModelProperty", "Could not set model property
"&PropertyName(z)&": ", -1, iResult)
    End if

    iResult = PGModel.GetModelProperty(""&PropertyName(z), outValue)
    if (iResult <> 0) Then
      Call APILog("GetSetModelProperty", "Could not get model property
"&PropertyName(z)&": ", -1, iResult)
    End if

    if (""&PropertyValue(z) = ""&outValue) Then
      Call APILog("GetSetModelProperty", "Get/Set had same value", 0, 0)
    else
      Call APILog("GetSetModelProperty", "Get/Set had different value!", -1, 0)
    End if
  next

  for z = 4 to 6
    iResult = PGModel.GetModelProperty(""&PropertyName(z), outValue)
    if (iResult <> 0) Then
      Call APILog("GetSetModelProperty", "Could not get model property
"&PropertyName(z)&": ", -1, iResult)
    End if
  next
End Sub
'*****************************************************************************
'*****************************************************************************
Sub APILog(log1, log2, error, iResult)
  Dim objFSO, objFolder, objShell, objTextFile
```

```
Dim dDate
Dim dTime
Dim strFile
Dim objFSO2, objFolder2, objShell2, strDirectory2

strDirectory2 = "<PFolder>"

if (""&log2 = "Starting to run 'PG_UserRights.pmf'") Then
  Set objFSO2 = CreateObject("Scripting.FileSystemObject")
  if objFSO2.FolderExists(strDirectory2) Then
    Set objFolder2 = objFSO2.GetFolder(strDirectory2)
    Call APILog("APILog", "Folder has already been created", 0, 0)
  else
    Set objFolder2 = objFSO2.CreateFolder(strDirectory2)
    Call APILog("APILog", "Created folder : "&strDirectory2, 0, 0)
  End if
End if

Dim i

strFile = "<PFolder>\Logs\PG_INT_Log.txt"

dTime = Time
dDate = Date

if (error = -1 OR error = "") Then
  ApiResult = ApiResult + 1

  'MsgBox ""&log1&", "&log2&": " + PGApplication.GetErrorMessage(iResult)
End If

Set objFSO = CreateObject("Scripting.FileSystemObject")

If objFSO.FileExists(""&strFile) = 0 Then
  Set objTextFile = objFSO.CreateTextFile(""&strFile)
```

```
    objTextFile.WriteLine(""&dTime &"                    "&dDate &"  "&log1 &"      "&log2 +
PGApplication.GetErrorMessage(iResult))
    if (APIResult <> 0) Then
      objTextFile.WriteLine(""&dTime &"                  "&dDate &"  Error:"&log1 &"
                  "&log2 + PGApplication.GetErrorMessage(iResult))
      'PGApplication.Quit(0)
    End if
    objTextFile.Close
  Else
    Set objTextFile = objFSO.OpenTextFile(""&strFile, 8, True)
    objTextFile.WriteLine(""&dTime &"                    "&dDate &"  "&log1 &"      "&log2 +
PGApplication.GetErrorMessage(iResult))
    if (APIResult <> 0) Then
      objTextFile.WriteLine(""&dTime &"                  "&dDate &"  Error:"&log1 &"
                   "&log2 + PGApplication.GetErrorMessage(iResult))
                  'TEST
                  APIResult = 0
                  'End Testi
      'PGApplication.Quit(0)
    End if
    objTextFile.Close
  End If

  if (""&log2 = "Completed PG_Intermediate") Then
    Set objFSO = CreateObject("Scripting.FileSystemObject")

    Set objTextFile = objFSO.OpenTextFile(""&strFile, 8, True)
    if (ApiResult = 0) Then
      objTextFile.WriteLine(""&dTime &"                  "&dDate &"                  All OK!")
    else
      objTextFile.WriteLine(""&dTime &"                  "&dDate &"                  Error:
Errors occurred while executing script!")
    End if
    objTextFile.Close
  End if
```

```
End Sub
'*************************************************************************
'*************************************************************************
```

```
<QPR_SCRIPT_FILE language = vbscript TimeOut=0>
Option Explicit

Private stUser                          'Username
Private stUserPw            'Users password
Private y
Private iResult
Private ViewUser
Public completed
Private objFSO, objFolder, objShell, strDirectory
Private ExternalDirectory

  stUser = "qpr"
  stUserPw = "demo"
  y = 0
  completed = True
  ViewUser = "ViewUser"
  strDirectory = "<PFolder>"

Private SCName              'Helps identificate Scorecards
'*********************************************************
'*********************************************************
Sub Main()

  Dim n
  Dim i
  Dim oModel
  Dim sModelName
  Dim bCalculation

  i = 0
  n = 0
  bCalculation = False
```

```
Call APILog("Main", "Starting to run SC_Intermediate", 0)

iResult = SCApplication.SetUIMode(0)

'Set objFSO = CreateObject("Scripting.FileSystemObject")
'if objFSO.FolderExists(strDirectory) Then
'  Set objFolder = objFSO.GetFolder(strDirectory)
'  Call APILog("Main", "Folder has already been created", 0)
'else
'  Set objFolder = objFSO.CreateFolder(strDirectory)
'  Call APILog("Main", "Created folder : "&strDirectory, 0)
'End if

Call ConnectToServerAndAuthenticate()

Call Applications()

Call VarialTest()

'*************************
'SC26398B
'Call SaveAndMerge(2)
'*************************

Call MiscTest()

Call OpenModel(oModel, sModelName)

Call Calculations(bCalculation)

i = 0
do while i < 4
  if (i=0) Then
    SCName = "NONE"
```

```
  elseif (i=1) Then
    SCName = "VIEW"
  elseif (i=2) Then
    SCName = "UPDATE"
  elseif (i=3) Then
    SCName = "ADMIN"
  End if
  Call CreateSCElements()
  i = i + 1
loop

Call Calculations(bCalculation)

Call Authenticate()

Call OpenModel(oModel, sModelName)

Call CopyScorecards()

Call Modify()

Call DeleteScorecards(ViewUser)

Call DeleteModel()

Call ConnectToServerAndAuthenticate

Call OpenModel(oModel, sModelName)

Call DeleteScorecards(stUser)

'*************************
'SC26398B
'Call SaveAndMerge(1)
'*************************
```

```
  '*************************
  'SC26398B
  'Call DeleteModels()
  '*************************

  'if objFSO.FolderExists(strDirectory) Then
  '  objFSO.DeleteFolder(strDirectory)
  '  Call APILog("Main", "Deleted directory", 0)
  'else
  '  Call APILog("Main", "Directory not found", -1)
  'End if

  Call APILog("Main", "SC_Intermediate completed", 0)

  SCApplication.Quit

End Sub
'*******************************************************
'*******************************************************
Sub ConnectToServerAndAuthenticate()

  iResult = SCApplication.ConnectToServer("localhost", <PortN>)
  if (iResult <> 0) Then
    Call APILog("ConnectToServerAndAuthenticate", "Could not connect to server: ", -1)
  End if

  iResult = SCApplication.Authenticate(""&stUser, ""&stUserPw, "")
  if (iResult <> 0) Then
    iResult = SCApplication.WriteLog("API: ConnectToServerAndAuthenticate", "Authentica-
tion failed", -1)
    Call APILog("ConnectToServerAndAuthenticate", "Authentication failed: ", -1)
  End if

End Sub
```

```
'*****************************************************
'*****************************************************
Sub Authenticate()

  Call APILog("Authenticate", "Autenticating 'ViewUser'", 0)

  iResult = SCApplication.Authenticate(""&ViewUser, "", "")
  if (iResult <> 0) Then
    iResult = SCApplication.WriteLog("API: ConnectToServerAndAuthenticate", "Authentica-
tion failed", -1)
    Call APILog("Authenticate", "Authentication failed: ", -1)
  End if

End Sub
'*****************************************************
'*****************************************************
Sub Applications()
  Dim AppProperty
  Dim PropertyName
  Dim i
  Dim Models()
  Dim ModelName
  Dim NeedModel
  Dim UserGroupArray()
  Dim UserId
  Dim UserName
  Dim UserInfoArray(0,3)

  NeedModel = FALSE

  for i = 0 to 2
    if (i = 0) Then
      PropertyName = "NAME"
    elseif (i = 1) Then
      PropertyName = "VERSION"
```

```
  else
    PropertyName = "EXEPATH"
  End if

  iResult = SCApplication.GetProperty(PropertyName, AppProperty)
  if (iResult <> 0) Then
    Call APILog("Applications", "Could not get property: ", -1)
  End if

  Call APILog("Applications", ""&PropertyName &": "&AppProperty, 0)
Next

i = 0

iResult = SCApplication.GetModels(Models)
if (iResult <> 0) Then
  Call APILog("Applications", "Could not get models: ", -1)
End if

for I = lbound(Models, 1) to ubound(Models, 1)
  ModelName = Models(i, 0)
  if (ModelName = "Government") Then
    NeedModel = TRUE
  End if
Next

if (NeedModel = FALSE) Then
  Call APILog("Applications", "Government - model not found. Test will fail.", -1)
else
  Call APILog("Applications", "Government - model found", 0)
End if

iResult = SCApplication.GetGroupInfo(-1, UserGroupArray)
if (iResult <> 0) Then
  Call APILog("Applications", "Could not get group info: ", -1)
```

```
  End if

  iResult = SCApplication.GetuserId(stUser, UserId)
  if (iResult <> 0) Then
    Call APILog("Applications", "Could not get User Id: ", -1)
  End if

  iResult = SCApplication.GetuserName(UserId, UserName)
  if (iResult <> 0) Then
    Call APILog("Applications", "Could not get user name: ", -1)
  End if

  if(stUser = UserName) Then
    Call APILog("Applications", "User is correct", 0)
  else
    Call APILog("Applications", "Wrong user in use. Some error in script: ", -1)
  End if

  iResult = SCApplication.GetUserInfo(userId, UserInfoArray)
  if (iResult <> 0) Then
    Call APILog("Applications", "Could not get user info", -1)
  End if

  if (userInfoArray(0,0) = UserId AND UserInfoArray(0,1) = stUser) Then
    Call APILog("Applications", "User name and id matches", 0)
  else
    Call APILog("Applications", "User name and Id doesn't match: ", -1)
  End if
End sub
'*****************************************************
'*****************************************************
Sub SaveAndMerge(number)
  Dim ModelId
  Dim ImportedId
```

```
Call APILog("SaveAndMerge", "Saving and merging models", 0)

iResult = SCApplication.CreateModel("Merge1", "Government", ModelId)
if (iResult <> 0) Then
  Call APILog("SaveAndMerge", "Could not create a new model", -1)
End if

iResult = SCModel.SaveAs("Merge2")
if (iResult <> 0) Then
  Call APILog("SaveAndMerge", "Could not save model: ", -1)
End if

iResult = SCApplication.OpenModel("Merge2", ModelId)
if (iResult <> 0) Then
  Call APILog("SaveAndMerge", "Could not open model: ", -1)
End if

'SCApplication.sleep(10000)

iResult = SCModel.ExportModel(""&strDirectory &"\Merge.smf")
if (iResult <> 0) Then
  Call APILog("SaveAndMerge", "Could not export model: ", -1)
End if

iResult = SCApplication.OpenModel("Merge1", ModelId)
if (iResult <> 0) Then
  Call APILog("SaveAndMerge", "Could not open model: ", -1)
End if

iResult = SCModel.MergeModel(""&strDirectory &"\Merge.smf")
if (iResult <> 0) Then
  Call APILog("SaveAndMerge", "Could not merge models: ", -1)
End if

if (number = 2) Then
```

```
    iResult = SCApplication.OpenModel("Merge1", ModelId)
    if (iResult <> 0) Then
      Call APILog("DeleteModels", "Could not open model: ", -1)
    End if

    iResult = SCModel.DeleteModel
    if (iResult <> 0) Then
      Call APILog("DeleteModels", "Could not delete model", -1)
    End if

    iResult = SCApplication.OpenModel("Merge2", ModelId)
    if (iResult <> 0) Then
      Call APILog("DeleteModels", "Could not open model: ", -1)
    End if

    iResult = SCModel.DeleteModel
    if (iResult <> 0) Then
      Call APILog("DeleteModels", "Could not delete model", -1)
    End if
  End if
End Sub
'*****************************************************
'*****************************************************
Sub DeleteModels()
  Dim iResult
  Dim ModelId

  iResult = SCApplication.OpenModel("Merge1", ModelId)
  if (iResult <> 0) Then
    Call APILog("DeleteModels", "Could not open model: ", -1)
  End if

  iResult = SCModel.DeleteModel
  if (iResult <> 0) Then
    Call APILog("DeleteModels", "Could not delete model", -1)
```

```
  End if

  iResult = SCApplication.OpenModel("Merge2", ModelId)
  if (iResult <> 0) Then
    Call APILog("DeleteModels", "Could not open model: ", -1)
  End if

  iResult = SCModel.DeleteModel
  if (iResult <> 0) Then
    Call APILog("DeleteModels", "Could not delete model", -1)
  End if

End Sub
'*****************************************************
'*****************************************************
Sub OpenModel(voModel, vsModelName)
  Dim iResult
  Dim oModel
  Dim Model

  Model = "Government"

  iResult = SCApplication.OpenModel("Government", oModel)
  if (iResult <> 0) Then
    Call APILog("OpenModel", "Could not open model using name ", -1)
  End if

  iResult = SCModel.CloseModel
  if (iResult <> 0) Then
    Call APILog("OpenModel", "Could not close model", -1)
  End if

  iResult = SCApplication.OpenModel(Model, oModel)
  if (iResult <> 0) Then
    Call APILog("OpenModel", "Could not open model using varial", -1)
```

```
  End if

  iResult = SCModel.CloseModel
  if (iResult <> 0) Then
    Call APILog("OpenModel", "Could not close model", -1)
  End if

  iResult = SCApplication.OpenModel(""&Model, oModel)
  if (iResult <> 0) Then
    Call APILog("OpenModel", "Could not open model using varial name", -1)
  End if

End Sub
'*****************************************************
'*****************************************************
Sub Calculations(bCalculation)

  if(bCalculation = False) Then
    iResult = SCModel.SetCalculation(0)
    if (iResult <> 0) Then
      Call APILog("Calculations", "Could not set calculation: ", -1)
    else
      Call APILog("Calculations", "Disabled calculation", 0)
    End if
    bCalculation = True
  else
    iResult = SCModel.SetCalculation(1)
    if (iResult <> 0) Then
      Call APILog("Calculations", "Could not set calculation: ", -1)
    else
      Call APILog("Calculations", "Enabled calculation", 0)
    End if
    iResult = SCModel.Recalculate(0, "")
    if (iResult <> 0) Then
      Call APILog("Calculations", "Could not recalculate model: ", -1)
```

```
    else
      Call APILog("Calculations", "Recalculated model", 0)
    End if
  End if

End Sub
'*********************************************************
'*********************************************************
Sub VarialTest()
  Dim iVarSCId
  Dim iVarMeaId
  Dim PropertyName
  Dim outResult
  Dim Model
  Dim CreateModel
  Dim outModel
  Dim OpenModel
  Dim ModelName

  CreateModel = "VarialTest Model"
  ModelName = "Government"

  Call APILog("VarialTest", "Starting to test different varials etc.", 0)

  iResult = SCApplication.CreateModel("VarialTest Model", ""&ModelName, outModel)
  if (iResult <> 0) Then
    Call APILog("VarialTest", "Could not create model", -1)
  End if

  'SCApplication.Sleep(10000)

  iResult = SCApplication.OpenModel(CreateModel, OpenModel)
  if (iResult <> 0) Then
    Call APILog("VarialTest", "Could not open model", -1)
  End if
```

```
iResult = SCModel.CreateScorecard("VarialTest", "VART1", 0, iVarSCId)
if (iResult <> 0) Then
  Call APILog("VarialTest", "Could not create scorecard", -1)
End if

iResult=SCModel.CreateElement(0, "VarTestMeasure", "VAME1", iVarSCId, 0, iVarMeaId)
If (iResult <> 0) Then
  Call APILog("VarialTest", "Could not create element: ", -1)
End If

iResult = SCModel.SetProperty(""&iVarMeaId, "ACCUMULATIONRULE", 7)
If (iResult <> 0 ) Then
  Call APILog("VarialTest", "Could not set property 'ACCUMALATIONRULE': ", -1)
End If

PropertyName = "DESCRIPTION"

iResult = SCModel.SetProperty(iVarMeaId, PropertyName, "This is a description")
If (iResult <> 0 ) Then
  Call APILog("VarialTest", "Could not set property 'DESCRIPTION': ", -1)
End If

PropertyName = "MEASUREUNITID"

iResult = SCModel.SetProperty(iVarMeaId, ""&PropertyName, 272)
If (iResult <> 0 ) Then
  Call APILog("VarialTest", "Could not set property 'MEASUREUNITID': ", -1)
End If

iResult = SCModel.SetProperty(iVarMeaId, "PERIODLEVELID", 36)
If (iResult <> 0 ) Then
  Call APILog("VarialTest", "Could not set property 'PERIOIDLEVELID': ", -1)
End If
```

```
iResult = SCModel.SetProperty(iVarMeaId, "VALUESETTINGID", 24)
If (iResult <> 0 ) Then
  Call APILog("VarialTest", "Could not set property 'VALUESETTINGID': ", -1)
End If

iResult = SCModel.GetProperty(""&iVarMeaId, "ACCUMULATIONRULE", outResult)
if (iResult <> 0) Then
  Call APILog("VarialTest", "Could not get property 'ACCUMULATIONRULE'", -1)
End if

PropertyName = "DESCRIPTION"

iResult = SCModel.GetProperty(iVarMeaId, PropertyName, outResult)
if (iResult <> 0) Then
  Call APILog("VarialTest", "Could not get property 'DESCRIPTION'", -1)
End if

PropertyName = "MEASUREUNITID"

iResult = SCModel.GetProperty(iVarMeaId, ""&PropertyName, outResult)
if (iResult <> 0) Then
  Call APILog("VarialTest", "Could not get property 'MEASUREUNITID'", -1)
End if

iResult = SCModel.DeleteElement(iVarSCId)
if (iResult <> 0) Then
  Call APILog("VarialTest", "Could not delete scorecard", -1)
End if

iResult = SCModel.DeleteModel
if (iResult <> 0) Then
  Call APILog("VarialTest", "Could not delete model", -1)
End if

End Sub
```

```
'*******************************************************
'*******************************************************
Sub MiscTest()
  Dim oModel
  Dim iSCId
  Dim ElementId
  Dim Properties(4,1)
  Dim ObjectId
  Dim Rights()
  Dim bArchived
  Dim GotProperties()
  Dim Types()
  Dim TypeId
  Dim Values()
  Dim outFind
  Dim i

  Call APILog("MiscTest", "Starting to do misc tests", 0)

  Call ArrayOfProperties(Properties)

  Call APILog("MiscTest", "ArrayOfProperties called", 0)

  iResult = SCApplication.CreateModel("MiscTestModel", "Government", oModel)
  if (iResult <> 0) Then
    Call APILog("MiscTest", "Could not create new model: ", -1)
  End if

  Call APILog("MiscTest", "Model Created", 0)

'  iResult = SCModel.SetModelProperty("ModelArchived", TRUE)
'  if (iResult <> 0) Then
'    Call APILog("MiscTest", "Could not set model property 'ModelArchived' = TRUE: ", -1)
'  End if
```

```
  Call APILog("MiscTest", "Model Archived", 0)

'  iResult = SCModel.SetModelProperty("ModelArchived", FALSE)
'  if (iResult <> 0) Then
'    Call APILog("MiscTest", "Could not set model property 'ModelArchived' = FALSE: ", -1)
'  End if

  Call APILog("MiscTest", "Model unarchived", 0)

  iResult = SCModel.GetModelProperty("ModelArchived", bArchived)
  if (iResult <> 0) Then
    Call APILog("MiscTest", "Coudl not get model property: ", -1)
  End if

  Call APILog("MiscTest", "Got archived status", 0)

  iResult = SCModel.GetActive("MODEL", oModel)
  if (iResult <> 0) Then
    Call APILog("MiscTest", "Could not get active model: ", -1)
  End if

  Call APILog("MiscTest", "Got active model", 0)

  iResult = SCModel.CreateScorecard("MiscTestScorecard", "MTSC1", 0, iSCId)
  if (iResult <> 0) Then
    Call APILog("MiscTest", "Could not create a new scorecard: ", -1)
  End if

  Call APILog("MiscTest", "Created SC 'MiscTestScorecard'", 0)

  'iResult = SCModel.Find("ELEMENTTYPES=SCORECARD;", "(NAME = ""Misc-
TestScorecard"") AND (SYMBOL = ""MTSC1"")", "", outFind)
  'if (iResult <> 0) Then
  '  Call APILog("MiscTest", "Could not Find Scorecard id: ", -1)
  'End if
```

```
'if IsEmpty(outFind) Then
'  Call APILog("MiscTest", "Could not FIND Scorecard id: ", -1)
'else
'  Call APILog("MiscTest", "Scorecard Id found. Comparing...", 0)
'  for i = lbound(outFind) to uBound(outFind)
'    if (""&outFind(i) <> ""&iSCId) Then
'      Call APILog("MiscTest", "Scorecard id's does not match: ", -1)
'    else
'      Call APILog("MiscTest", "Found scorecard id matches to created scorecard.", 0)
'    End if
'  Next
'End if

  iResult = SCModel.CreateElement(0, "MiscTestElement", "MiscMEA1", iSCId, 0, Elemen-
tId)
  if (iResult <> 0) Then
    Call APILog("MiscTest", "Could not create a new element: ", -1)
  End if

  Call APILog("MiscTest", "Created Element 'MiscTestElement'", 0)

  iResult = SCModel.SetProperties(ElementId, Properties)
  if (iResult <> 0) Then
    Call APILog("MiscTest", "Could not set properties for element: ", -1)
  End if

  Call APILog("MiscTest", "Set properties to the element", 0)

  iResult = SCModel.GetProperties(ElementId, GotProperties)
  if (iResult <> 0) Then
    Call APILog("MiscTest", "Could not get properties", -1)
  End if

  Call APILog("MiscTest", "get properties", 0)
```

```
iResult = SCModel.CreateObject("INFORMATIONITEM", "Jep Jep", "", ObjectId)
if (iResult <> 0) Then
  Call APILog("MiscTest", "Could not create object", -1)
End if

Call APILog("MiscTest", "Created information item", 0)

iResult = SCModel.SetObjectRights(ObjectId, "qpr", 3)
if (iResult <> 0) Then
  Call APILog("MiscTest", "Could not set object rights: ", -1)
End if

Call APILog("MiscTest", "Set rights to information item", 0)

iResult = SCModel.GetObjectRights(ObjectId, Rights)
if (iResult <> 0) Then
  Call APILog("MiscTest", "Could not get object rights", -1)
End if

Call APILog("MiscTest", "Get object rights", 0)

iResult = SCModel.GetTypes(Types)
if (iResult <> 0) Then
  Call APILog("MiscTest", "Could not get types", -1)
End if

Call APILog("MiscTest", "Got types", 0)

iResult = SCModel.GetTypeIdByName("Element", TypeId)
if (iResult <> 0) Then
  Call APILog("MiscTest", "Could not get type id by name: ", -1)
End if

Call APILog("MiscTest", "Got type id by name", 0)
```

```
  iResult = SCModel.GetValues(0, "", 0, Values)
  if (iResult <> 0) Then
    Call APILog("MiscTest", "Could not get values: ", -1)
  End if

  Call APILog("MiscTest", "Got values", 0)

  iResult = SCModel.DeleteModel
  if (iResult <> 0) Then
    Call APILog("MiscTest", "Could not delete model: ", -1)
  End if

  Call APILog("MiscTest", "Deleted model", 0)

  SCApplication.Sleep(10000)

End Sub
'*********************************************************
'*********************************************************
Sub CreateSCElements()
  Dim RootSCId
  Dim TopElementId
  Dim iSCId
  Dim ElementRights
  Dim values(6,2)
  Dim nimi
  Dim Array(0,2)
  Dim z
  Dim i
  Dim ReferenceId
  Dim ObjectId
  Dim DeleteObjectId

  i = 0
```

```
z = 0

call GetValuesFrom(values)

iResult=SCModel.CreateScorecard("Scorecard " & SCName, "SC"&SCName, 0, iSCId)
If (iResult <> 0) Then
  Call APILog("CreateSCElements", "Could not create scorecard: ", -1)
End If

iResult = SCModel.SetObjectRights(iSCId, "ViewUser", y)
if (iResult <> 0) Then
  Call APILog("CreateSCElements", "Could not set scorecard rights: ", -1)
End if

iResult = SCModel.CreateObject("INFORMATIONITEM", "This is an information item.",
"", ObjectId)
if (iResult <> 0) Then
  Call APILog("CCreateSCElements", "Could not create object", -1)
End if

iResult = SCModel.AttachObject(ObjectId, iSCId)
if (iResult <> 0) Then
  Call APILog("CreateSCElements", "Could not attach object to scorecard", -1)
End if

do while i < 4
  Dim PerspectiveId

  if (i=0) Then
    ElementRights = "NONE"
  elseif (i=1) Then
    ElementRights = "VIEW"
  elseif (i=2) Then
    ElementRights = "UPDATE"
  else
```

```
    ElementRights = "ADMIN"
  End if

  iResult=SCModel.CreateElement(0, "UserRights " &SCName&ElementRights, "Measure"
&SCName &ElementRights, iSCId, 0, PerspectiveId)
  If (iResult <> 0) Then
    Call APILog("CreateSCElements", "Could not create element: ", -1)
  End If

  iResult = SCModel.SetProperty(""&PerspectiveId, "ACCUMULATIONRULE", 7)
  If (iResult <> 0 ) Then
    Call APILog("CreateSCElements", "Could not set property 'ACCUMALATIONRULE':
", -1)
  End If

  iResult = SCModel.SetProperty(PerspectiveId, "DESCRIPTION", "This is a description")
  If (iResult <> 0 ) Then
    Call APILog("CreateSCElements", "Could not set property 'DESCRIPTION': ", -1)
  End If

  iResult = SCModel.SetProperty(PerspectiveId, "MEASUREUNITID", 272)
  If (iResult <> 0 ) Then
    Call APILog("CreateSCElements", "Could not set property 'MEASUREUNITID': ", -1)
  End If

  iResult = SCModel.SetProperty(PerspectiveId, "PERIODLEVELID", 36)
  If (iResult <> 0 ) Then
    Call APILog("CreateSCElements", "Could not set property 'PERIOIDLEVELID': ", -1)
  End If

  iResult = SCModel.SetProperty(PerspectiveId, "VALUESETTINGID", 24)
  If (iResult <> 0 ) Then
    Call APILog("CreateSCElements", "Could not set property 'VALUESETTINGID': ", -1)
  End If
```

```
    'iResult=SCModel.SetValues(PerspectiveId, values)
    'If (iResult <> 0) Then
    '  Call APILog("CreateSCElements", "Could not set values for element: ", -1)
    'End If

    Call SetMeaFormula(PerspectiveId)

    iResult = SCModel.SetObjectRights(PerspectiveId, "ViewUser", z)
    if (iResult <> 0) Then
      Call APILog("CreateSCElements", "Could not set object rights: ", -1)
    End if

    iResult = SCModel.GetGraph(PerspectiveId, ""&strDirectory &"\trend"&i &z &".jpg",
"GRAPHTYPE=TREND;PERIOD=-1;SERIES=-1")
'    if (iResult <> 0) Then
'      Call APILog("CreateSCElements", "Could not get graph: ", -1)
'    End if

    iResult = SCModel.GetGraph(PerspectiveId, ""&strDirectory &"\template"&i &z &".jpg",
"WIDTH=800; HEIGHT=400;GRAPHTYPE=CHART;PERIOD=-1;SERIES=-1; TEM-
PLATE=Month Template;")
'    if (iResult <> 0) Then
'      Call APILog("CreateSCElements", "Could not get graph template: ", -1)
'    End if

    'iResult = SCModel.GetGraph(PerspectiveId, ""&strDirectory &"\chart"&i &z&".jpg",
"GRAPHTYPE=CHART;PERIOD=-4;SERIES=-1")
    'if (iResult <> 0) Then
    '  Call APILog("CreateSCElements", "Could not get element chart graph: ", -1)
    'End if

    iResult = SCModel.CreateReferenceElement(PerspectiveId, iSCId, 0, ReferenceId)
    if (iResult <> 0) Then
      Call APILog("CreateSCElements", "Could not create reference element: ", -1)
    End if
```

```
    z = z + 1
    i = i + 1
  loop

  iResult = SCModel.GetObjectIdByName("INFORMATIONITEM", "This is an informa-
tion item.", DeleteObjectId)
  if (iResult <> 0) Then
    Call APILog("CreateSCElements", "Could not get object id by name: ", -1)
  End if

  iResult = SCModel.DetachObject(DeleteObjectId, iSCId)
  if (iResult <> 0) Then
    Call APILog("CreateSCElements", "Could not detach object to scorecard", -1)
  End if

  iResult = SCModel.DeleteObject(DeleteObjectId)
  if (iResult <> 0) Then
    Call APILog("CreateSCElements", "Could not delete object", -1)
  End if

  y = y + 1

  Call APILog("CreateSCElements", "Created: Scorecard "& SCName, 0)
End Sub
'*********************************************
'*********************************************
Sub Modify()
  Dim RootSCId
  Dim TopElementId
  Dim iSCId
  Dim ElementRights
  Dim nimi
  Dim z
  Dim i
```

```
Dim name
Dim values2(6,2)
Dim iCopyId
Dim iActiveId
Dim oModel
Dim pass
Dim fail

pass = 0
fail = -1

i = 0
z = 0

Call APILog("Modify", "Starting to modify objects", 0)

call GetValuesFrom2(values2)

do while i < 4
 if (i=0) Then
  SCName = "NONE"
 elseif (i=1) Then
  SCName = "VIEW"
 elseif (i=2) Then
  SCName = "UPDATE"
 elseif (i=3) Then
  SCName = "ADMIN"
 End if
 do while z < 4
  if (z=0) Then
   ElementRights = "NONE"
  elseif (z=1) Then
   ElementRights = "VIEW"
  elseif (z=2) Then
   ElementRights = "UPDATE"
```

```
    elseif (z=3) Then
      ElementRights = "ADMIN"
    End if

    if (ElementRights = "NONE") Then
      pass = - 1
      fail = 0
    Else
      pass = 0
      fail = -1
    End if

    Call APILog("Modify", "Scorecard: "&SCName &ElementRights, 0)

    iResult = SCModel.GetElementId("SC"&SCName, "Meas-
ure"&SCName&ElementRights, iSCId)
    if (iResult <> 0) Then
      Call APILog("Modify", "Could not get element id: ", fail)
    else
      Call APILog("Modify", "Get element id", pass)
    End if

    iResult = SCModel.GetGraph(iSCId, ""&strDirectory &"trend"&i &z &".jpg",
"GRAPHTYPE=TREND;PERIOD=-1;SERIES=-1")
'     if (iResult <> 0) Then
'       Call APILog("Modify", "Could not get graph: ", pass)
'     else
'       Call APILog("Modify", "User got graph from element", fail)
'     End if

    iResult = SCModel.SetObjectRights(iSCId, ""&ViewUser, 3)
    if (iResult <> 0) Then
      if(ViewUser = "ViewUser") Then
        fail = 0
      End if
```

```
    Call APILog("Modify", "Could not set object rights: ", fail)
  else
    if(ViewUser = "ViewUser") Then
      pass = -1
    End if
    Call APILog("Modify", "Set object rights", pass)
  End if

  iResult = SCModel.RemoveObjectRights(iSCId, ""&stUser)
  if (iResult <> 0) Then
    if(ViewUser = "ViewUser") Then
      fail = 0
    End if
    Call APILog("Modify", "Could not remove object rights: ", fail)
  else
    if(ViewUser = "ViewUser") Then
      pass = -1
    End if
    Call APILog("Modify", "Removed object rights", pass)
  End if

  iResult = SCModel.GetProperty(iSCId, "NAME", name)
  if (iResult <> 0) Then
    if(ElementRights = "VIEW" OR ElementRights = "UPDATE" OR ElementRights =
"ADMIN") Then
      fail = -1
    else
      fail = 0
    End if
    Call APILog("Modify", "Could not get element name: ", fail)
  else
    if(ElementRights = "VIEW" OR ElementRights = "UPDATE" OR ElementRights =
"ADMIN") Then
      pass = 0
    else
```

```
    pass = -1
  End if
  Call APILog("Modify", "Get property 'NAME'", pass)
End if

iResult = SCModel.SetProperty(iSCId, "NAME", "Changed "&name)
if (iResult <> 0) Then
  if(ElementRights = "UPDATE" OR ElementRights = "ADMIN") Then
    pass = 0
    fail = -1
  else
    pass = -1
    fail = 0
  End if
  Call APILog("Modify", "Could not set Element name: ", fail)
else
  Call APILog("Modify", "Set new Element name", pass)
End if

iResult = SCModel.SetProperty(iSCId, "ACCUMULATIONRULE", 4)
If (iResult <> 0 ) Then
  if(ElementRights = "UPDATE" OR ElementRights = "ADMIN") Then
    pass = 0
    fail = -1
  else
    pass = -1
    fail = 0
  End if
  Call APILog("Modify", "Could not set property 'ACCUMALATIONRULE': ", fail)
else
  Call APILog("Modify", "Set property 'ACCUMALATIONRULE'", pass)
End If

iResult = SCModel.SetProperty(iSCId, "DESCRIPTION", "CHANGED!!!!!!")
If (iResult <> 0 ) Then
```

```
  if(ElementRights = "UPDATE" OR ElementRights = "ADMIN") Then
    pass = 0
    fail = -1
  else
    pass = -1
    fail = 0
  End if
  Call APILog("Modify", "Could not set property 'DESCRIPTION': ", fail)
else
  Call APILog("Modify", "Set property 'DESCRIPTION'", pass)
End If

iResult = SCModel.SetProperty(iSCId, "MEASUREUNITID", 102)
If (iResult <> 0 ) Then
  if(ElementRights = "UPDATE" OR ElementRights = "ADMIN") Then
    pass = 0
    fail = -1
  else
    pass = -1
    fail = 0
  End if
  Call APILog("Modify", "Could not set property 'MEASUREUNITID': ", fail)
else
  Call APILog("Modify", "Set property 'MEASUREUNITID'", pass)
End If

iResult = SCModel.SetProperty(iSCId, "PERIODLEVELID", 35)
If (iResult <> 0 ) Then
  if(ElementRights = "UPDATE" OR ElementRights = "ADMIN") Then
    pass = 0
    fail = -1
  elseif(ElementRights = "VIEW" OR ElementRights = "NONE") Then
    pass = -1
    fail = 0
  End if
```

```
    Call APILog("Modify", "Could not set property 'PERIOIDLEVELID': ", fail)
   else
    Call APILog("Modify", "Set property 'PERIODLEVELID'", pass)
   End If

   iResult = SCModel.SetProperty(iSCId, "VALUESETTINGID", 24)
   If (iResult <> 0 ) Then
    if(ElementRights = "UPDATE" OR ElementRights = "ADMIN") Then
     pass = 0
     fail = -1
    else
     pass = -1
     fail = 0
    End if
    Call APILog("Modify", "Could not set property 'VALUESETTINGID': ", fail)
   else
    Call APILog("Modify", "Set property 'VALUESETTINGID'", pass)
   End If

   iResult=SCModel.SetValues(iSCId, values2)
   If (iResult <> 0) Then
    Call APILog("Modify", "Could not set values for element: ", fail)
   else
    Call APILog("Modify", "Set new values for element", pass)
   End If

   Call SetMeaFormula2(iSCId, ElementRights)

   z = z + 1
  loop
  i = i + 1
  z = 0
loop

Dim TestSC
```

```
  dim o
  Dim qCount

  qCount = 0

  iResult = SCModel.GetScorecards(TestSC)
  if (iResult <> 0) Then
    Call APILog("Modify", "Could not get scorecards: ", -1)
  else
    for o = lbound(TestSC, 1) to uBound(TestSC ,1)
      Dim outFindList
      Dim q
      Dim outName123

      iResult = SCMo-
del.Find("SCORECARDIDS="&TestSC(o,1)&";ELEMENTTYPES=ELEMENT", "", "",
outFindList)
      if (iResult <> 0) Then
        Call APILog("Modify", "Could not find any elements or scorecards: " -1)
      else
        for q = lbound(outFindList) to uBound(outFindList)
          qCount = qCount + 1
        Next
      End if

      if (qCount = 7) Then
        Call APILog("Modify", "Correct amount of elements found", 0)
      elseif(qCount < 7) Then
        Call APILog("Modify", "User did not find enough scorecards/elements. Minor user
rights issue", -1)
      else
        Call APILog("Modify", "User found too many scorecards/elements. MAJOR ISSUE!!!!",
-1)
      End if
```

```
    qCount = 0

  Next
End if

iResult = SCModel.ExportModel("c:\Export_"&SCName &".smf")
if (iResult <> 0) Then
  Call APILog("Modify", "Could not export model: ", fail)
else
  Call APILog("Modify", "Exported model", pass)
End if

iResult = SCModel.SaveAs(""&Viewuser &"_newModel")
if (iResult <> 0) Then
  Call APILog("Modify", "Could not save model to server: ", fail)
else
  Call APILog("Modify", "Saved model as server model", pass)
End if

iResult = SCModel.Recalculate(0,"")
if (iResult <> 0) Then
  Call APILog("Modify", "Could not recalculate model: ", fail)
else
  Call APILog("Modify", "Recalculated model", pass)
End if

End Sub
'*********************************************
'*********************************************
Sub GetScorecards(iSCNoneId, iSCViewId, iSCUpdateId, iSCAdminId)
  Dim Scorecards
  Dim i

  Call APILog("GetScorecards", "Getting scorecard id's", 0)
```

```
  iResult = SCModel.GetScorecards(Scorecards)
  if (iResult <> 0) Then
    Call APILog("GetScorecards", "Could not get scorecards", -1)
  else
    Call APILog("GetScorecards", "Got scorecards", 0)
  End if

  if (IsEmpty(Scorecards)=True) Then

  Else
    for i = lbound(Scorecards, 1) to ubound(Scorecards, 1)
      if (Scorecards(i,0) = "Scorecard NONE") Then
        iSCNoneId = Scorecards (i,1)
      elseif (Scorecards(i,0) = "Scorecard VIEW") Then
        iSCViewId = Scorecards (i,1)
      elseif (Scorecards(i,0) = "Scorecard UPDATE") Then
        iSCUpdateId = Scorecards (i,1)
      elseif (Scorecards (i,0) = "Scorecard ADMIN") Then
        iSCAdminId = Scorecards (i,1)
      End if
    next
  End if
End Sub
'*******************************************
'*******************************************
Sub CopyScorecards()
  Dim x
  Dim iCopyId
  Dim NewId
  Dim iSCNoneId
  DIM iSCViewId
  Dim iSCUpdateId
  Dim iSCAdminId

  Call APILog("CopyScorecards", "Here it comes", 0)
```

```
  Call GetScorecards(iSCNoneId, iSCViewId, iSCUpdateId, iSCAdminId)

  for x = 0 to 3
    if (x = 0) Then
      iCopyId = iSCNoneId
    elseif (x = 1) Then
      iCopyId = iSCViewId
    elseif (x = 2) Then
      iCopyId = iSCUpdateId
    else
      iCopyId = iSCAdminId
    End if

    iResult = SCModel.CopyScorecard(iCopyId, iCopyId, NewId)
    if (iResult <> 0) Then
      Call APILog("CopyScorecards", "Could not copy scorecard: ", 0)
    else
      Call APILog("CopyScorecards", "Copied scorecards", -1)
    End if
  next
End Sub
'*******************************************
'*******************************************
Sub DeleteScorecards(user)
  Dim iSCNoneId
  DIM iSCViewId
  Dim iSCUpdateId
  Dim iSCAdminId
  Dim iDeleteId
  Dim x
  Dim pass
  Dim fail

  Call APILog("DeleteScorecards", "Here it comes", 0)
```

```
 Call GetScorecards(iSCNoneId, iSCViewId, iSCUpdateId, iSCAdminId)

 for x = 0 to 3
  if (x = 0) Then
   iDeleteId = iSCNoneId
  elseif (x = 1) Then
   iDeleteId = iSCViewId
  elseif (x = 2) Then
   iDeleteId = iSCUpdateId
  else
   iDeleteId = iSCAdminId
  End if

  iResult = SCModel.DeleteElement(iDeleteId)
  if (iResult <> 0) Then
   if (user = "qpr")Then
    fail = -1
   else
    fail = 0
   End if
   Call APILog("DeleteScorecards", "Could not delete scorecard: ", fail)
  else
   if (user = "qpr")Then
    pass = 0
   else
    pass = -1
   End if
   Call APILog("DeleteScorecards", "Deleted scorecard", pass)
  End if
 next
End Sub
'*********************************************
'*********************************************
Sub DeleteModel()
```

```
  Dim oModel
  Dim pass
  Dim fail

  iResult = SCApplication.OpenModel(""&Viewuser &"_newModel", oModel)
  if (iResult <> 0) Then
    if (SCApplication.GetErrorMessage(iResult) = "") Then
      fail = -1
    else
      fail = 0
    End if
    Call APILog("Modify", "Could not open model: ", fail)
  else
    Call APILog("Modify", "Opened model: "&Viewuser &"_newModel", 0)
  End if

  iResult = SCModel.DeleteModel
  if (iResult <> 0) Then
    if (SCApplication.GetErrorMessage(iResult) = "") Then
      fail = -1
    else
      fail = 0
    End if
    Call APILog("Modify", "Could not delete model: ", fail)
  else
    Call APILog("Modify", "Deleted model", pass)
  End if
End Sub
'*****************************************
'*****************************************
Function GetValuesFrom(values)
    Values(0,0) = "ACT"                    'seriessymbol
    Values(0,1) = "1.1.2002"               'date
    Values(0,2) = "1,4"                                 'value
```

```
    Values(1,0) = "ACT"
    Values(1,1) = "5.7.2002"
    Values(1,2) = "2,66"

    Values(2,0) = "ACT"
    Values(2,1) = "1.1.2003"
    Values(2,2) = "3,4"

    Values(3,0) = "ACT"
    Values(3,1) = "1.12.2003"
    Values(3,2) = "7,9"

    Values(4,0) = "ACT"
    Values(4,1) = "1.1.2004"
    Values(4,2) = "14,7"

    Values(5,0) = "ACT"
    Values(5,1) = "1.12.2004"
    Values(5,2) = "22,7"

    Values(6,0) = "Text"
    Values(6,1) = "1.12.2004"
    Values(6,2) = "Moro"
End Function
'************************************************
'*******************************************
Function GetValuesFrom2(values2)
    Values2(0,0) = "ACT"                    'seriessymbol
    Values2(0,1) = "1.1.2002"                           'date
    Values2(0,2) = "666"                                'value

    Values2(1,0) = "ACT"
    Values2(1,1) = "5.7.2002"
    Values2(1,2) = "666"
```

```
    Values2(2,0) = "ACT"
    Values2(2,1) = "1.1.2003"
    Values2(2,2) = "666"

    Values2(3,0) = "ACT"
    Values2(3,1) = "1.12.2003"
    Values2(3,2) = "666"

    Values2(4,0) = "ACT"
    Values2(4,1) = "1.1.2004"
    Values2(4,2) = "666"

    Values2(5,0) = "ACT"
    Values2(5,1) = "1.12.2004"
    Values2(5,2) = "666"

    Values2(6,0) = "Text"
    Values2(6,1) = "1.12.2004"
    Values2(6,2) = "Changed"
End Function
'************************************************
'****************************************
Function ArrayOfProperties(Properties)
    Properties(0,0) = "DESCRIPTION"
    Properties(0,1) = "This tests set properties"

    Properties(1,0) = "MEASUREUNITID"
    Properties(1,1) = "272"

    Properties(2,0) = "ACCUMULATIONRULE"
    Properties(2,1) = "7"

    Properties(3,0) = "VALUESETTINGID"
    Properties(3,1) = "24"
```

```
    Properties(4,0) = "PERIODLEVELID"
    Properties(4,1) = "36"
End Function
'************************************************
'************************************************
Function SetMeaFormula(PerspectiveId)
  Dim fail

  if (ViewUser = "ViewUser") Then
    fail = 0
  Else
    fail = -1
  End if

  iResult = SCModel.SetFormula(PerspectiveId, "UALA", "AVERAGE_(ACT()+1)")
  If (iResult <> 0) Then
    Call APILog("SetMeaFormula", "Could not set formula for 'UALA': ", fail)
  End If

  iResult = SCModel.SetFormula(PerspectiveId, "UTAR", "AVERAGE_(ACT()-1)")
  If (iResult <> 0) Then
    Call APILog("SetMeaFormula", "Could not set formula for 'UTAR': ", fail)
  End If

  iResult = SCModel.SetFormula(PerspectiveId, "LTAR", "AVERAGE_(ACT()-2)")
  If (iResult <> 0) Then
    Call APILog("SetMeaFormula", "Could not set formula for 'LTAR': ", fail)
  End If

  iResult = SCModel.SetFormula(PerspectiveId, "LALA", "AVERAGE_(ACT()-3)")
  If (iResult <> 0) Then
    Call APILog("SetMeaFormula", "Could not set formula for 'LALA': ", fail)
  End If
End Function
'************************************************
```

```
'*********************************************
Function SetMeaFormula2(iSCId, ElementRights)
  Dim fail

  if (ViewUser = "ViewUser") Then
    fail = 0
  Else
    fail = -1
  End if

  iResult = SCModel.SetFormula(iSCId, "UALA", "AVERAGE_(ACT()+100)")
  If (iResult <> 0) Then
    Call APILog("SetMeaFormula2", "Could not set formula for 'UALA': ", fail)
  End If

  iResult = SCModel.SetFormula(iSCId, "UTAR", "AVERAGE_(ACT()-100)")
  If (iResult <> 0) Then
    Call APILog("SetMeaFormula2", "Could not set formula for 'UTAR': ", fail)
  End If

  iResult = SCModel.SetFormula(iSCId, "LTAR", "AVERAGE_(ACT()-100)")
  If (iResult <> 0) Then
    Call APILog("SetMeaFormula2", "Could not set formula for 'LTAR': ", fail)
  End If

  iResult = SCModel.SetFormula(iSCId, "LALA", "AVERAGE_(ACT()-300)")
  If (iResult <> 0) Then
    Call APILog("SetMeaFormula2", "Could not set formula for 'LALA': ", fail)
  End If
End Function
'*************************************************
'*************************************************
Sub APILog(log1, log2, error)
  Dim objFSO, objFolder, objShell, objTextFile
  Dim dDate
```

```
  Dim dTime
  Dim strFile
  Dim objFSO2, objFolder2, objShell2, strDirectory2

  strDirectory2 = "<PFolder>"

  if (""&log2 = "Starting to run SC_Intermediate") Then
    Set objFSO2 = CreateObject("Scripting.FileSystemObject")
    if objFSO2.FolderExists(strDirectory2) Then
      Set objFolder2 = objFSO2.GetFolder(strDirectory2)
      Call APILog("APILog", "Folder has already been created", 0)
    else
      Set objFolder2 = objFSO2.CreateFolder(strDirectory2)
      Call APILog("APILog", "Created folder : "&strDirectory2, 0)
    End if
  End if

  if (error = -1 OR error = "") Then
    completed = False
  End if

  strFile = "<PFolder>\Logs\SC_INT_Log.txt"

  dTime = Time
  dDate = Date

  Set objFSO = CreateObject("Scripting.FileSystemObject")

  If objFSO.FileExists(""&strFile) = 0 Then
    Set objTextFile = objFSO.CreateTextFile(""&strFile)
    objTextFile.WriteLine(""&dTime &"                    "&dDate &"  "&log1 &" "&log2 +
SCApplication.GetErrorMessage(iResult))
    if (completed = False) Then
      objTextFile.WriteLine(""&dTime &"                    "&dDate &"  "&log1 &" Er-
ror:"&log2 + SCApplication.GetErrorMessage(iResult))
```

```
    SCApplication.Quit
   End if
   objTextFile.Close
  Else
   Set objTextFile = objFSO.OpenTextFile(""&strFile, 8, True)
   objTextFile.WriteLine(""&dTime &"                    "&dDate &"  "&log1 &" "&log2 +
SCApplication.GetErrorMessage(iResult))
   if (completed = False) Then
    objTextFile.WriteLine(""&dTime &"                    "&dDate &"  "&log1 &" Er-
ror:"&log2 + SCApplication.GetErrorMessage(iResult))
    SCApplication.Quit
   End if
   objTextFile.Close
  End If

  if (log2 = "SC_Intermediate completed") Then
   if (completed = False) Then
    Set objTextFile = objFSO.OpenTextFile(""&strFile, 8, True)
    objTextFile.WriteLine(""&dTime &"                    "&dDate &"  Error: Errors occurred
while executing script")
    objTextFile.Close
   else
    Set objTextFile = objFSO.OpenTextFile(""&strFile, 8, True)
    objTextFile.WriteLine(""&dTime &"                    "&dDate &"  All OK!")
    objTextFile.Close
   End if
  End if
End Sub
'*************************************************************************
'*************************************************************************
```

```
<QPR_SCRIPT_FILE language = vbscript TimeOut=0>
Option Explicit

Private oModel
Private stUser
private stDeleteUser
Private stUserPw
Private n
Private iId
Private iId2
Private aId
Private zId
Private xId
Private i
Private BaseModel
Private stPath
Private ReplacedModel
Private ApiResult
Dim ShareArray(16,0)
Dim outIds
Dim ElemTypeId

  stUserPw = ""
  BaseModel = "UserRightsTest"
  ReplacedModel = "ReplaceModel"
  stPath = "C:\Regression"
  n = 0
'*****************************************************************************
'*****************************************************************************
Sub Main()

  Dim iResult
  Dim iActionId
  Dim iCopyActionId
  Dim objFSO, objFolder, objShell

  Call APILog("Main", "Starting to run PG_Advanced'", 0, iResult)

  Set objFSO = CreateObject("Scripting.FileSystemObject")
  if objFSO.FolderExists(stPath) Then
    Set objFolder = objFSO.GetFolder(stPath)
    Call APILog("Main", "Folder already created", 0, 0)
  else
    Set objFolder = objFSO.CreateFolder(stPath)
    Call APILog("Main", "Created folder", 0, 0)
  End if

  Call APILog("Main", "Setting UI mode (0)", 0, iResult)
  PGApplication.SetUIMode(0)
```

```
Call ConnectToServer()

Call GetGraphsAndVersion()

Call CreateBaseModelAndReplaceModel(oModel)

do while n < 9
  if (n = 0 OR n = 1 OR n = 2) then
    stUser = "ViewUser"
  elseif (n = 3 OR n = 4 OR n = 5) then
    stUser = "UpdateUser"
  elseif (n = 6 OR n = 7 OR n = 8) then
    stUser = "AdminUser"
  End if

  Call Authenticate()

  Call CreateModel(oModel)

  if(n = 0 OR n = 3 OR n = 6) then
    stDeleteUser = "ViewUser"
    Call AuthenticateDeleteUser()
  elseif(n = 1 OR n = 4 OR n = 7) then
    stDeleteUser = "UpdateUser"
    Call AuthenticateDeleteUser()
  elseif(n = 2 OR n = 5 OR n = 8) then
    stDeleteUser = "AdminUser"
    Call AuthenticateDeleteUser()
  End if

  Call ExportImportXML()

  'Call ReplaceModel()

  Call DeleteElement(oModel)

  n = n + 1
  Call APILog("Main", ""&BaseModel &" loop number "&n , 0, iResult)
loop

'Call DeleteModels(ReplacedModel)

Call DeleteModels(BaseModel)

'if objFSO.FolderExists(stPath) Then
'  objFSO.DeleteFolder(stPath)
'  Call APILog("Main", "Deleted directory", 0, 0)
'else
'  Call APILog("Main", "Directory not found", -1, 0)
'End if

Call OpenModel
```

```
  Call ArrayForAll(ShareArray)

  Call ProcessStepTest()

  'Call FlowTest()

  Call OrganizationItemTest()

  Call CaseTest()

  Call InformationItemTest()

  Call FlowCompositionTest()

  Call MeasureTest()

  Call MeasureDataTest()

  Call NoteTest()

  Call ModelTest()

  Call ResourceTest()

  Call StoreTest()


  Call APILog("Main", "Completed PG_Advanced", 0, iResult)

  PGApplication.Quit(1)

End Sub
'*****************************************************************************
'*****************************************************************************
Sub ConnectToServer()
  Dim iResult

  iResult = PGApplication.ConnectToServer("localhost", <PortN>)
  If (iResult <> 0) Then
    Call APILog("ConnectToServer", "Could not connect to server: ", -1, iResult)
  End If

End Sub
'*****************************************************************************
'*****************************************************************************
Sub Authenticate()
  Dim iResult

  Call APILog("Authenticating user: "&stUser ,
"++++++++++++++++++++++++++++++++++++++++++++++", 0, iResult)

  iResult = PGApplication.Authenticate(stUser, ""&stUserPw, "")
```

```
  If (iResult <> 0) Then
    Call APILog("Authenticate", "Could not authenticate user: ", -1, iResult)
  End If
End Sub
'*****************************************************************************
'*****************************************************************************
Sub AuthenticateDeleteUser()
  Dim iResult

  Call APILog("Authenticating DeleteUser: "&stDeleteUser,
"++++++++++++++++++++++++++++++++++++++++++++++++", 0, iResult)

  iResult = PGApplication.Authenticate(""&stDeleteUser, ""&stUserPw, "")
  If (iResult <> 0) Then
    Call APILog("ConnectToServerAndAuthenticate", "Could not authenticate user: ", -1, iRe-
sult)
  End If
End Sub
'*****************************************************************************
'*****************************************************************************
Sub GetGraphsAndVersion()
  Dim iResult
  Dim ViewId
  Dim TestModel
  Dim ModelList
  Dim i
  Dim ModelName
  Dim BranchId

  iResult = PGApplication.Authenticate("AdminUser", "", "")
  if (iResult <> 0) Then
    Call APILog("GetGraphs", "Could not auteticate AdminUser: ", -1, iResult)
  End if

  iResult = PGApplication.GetServerModels(ModelList)
  if (iResult <> 0) Then
    Call APILog("GetGraphs", "Could not get server models: ", -1, iResult)
  End if

  for i = lBound(ModelList,1) to uBound(ModelList,1)
    if (""&ModelList(i,3) = "7") Then
      ModelName = ""&ModelList(i,9)
    End if
  Next

  iResult = PGApplication.OpenServerModel(""&ModelName, TestModel)
  if (iResult <> 0) Then
    Call APILog("GetGraphs", "Could not open model: ", -1, iResult)
  End if

  iResult = PGModel.CreateBranch("TestBranch", "", BranchId)
  if (iResult <> 0) Then
```

```
   Call APILog("CreateBaseModelAndReplaceModel", "Could not create branch: ", -1, iRe-
sult)
  End if

  iResult = PGModel.GetActive("VIEWSETTINGS", ViewId)
  if (iResult <> 0) Then
    Call APILog("GetGraphs", "Could not get view setting id: ", -1, iResult)
  End if

  iResult = PGModel.GetGraph(0, "C:\Regression\TestGraph.jpg",
"TYPE=JPG;VIEWSETTINGS="&ViewId)
  if (iResult <> 0) Then
     Call APILog("GetGraphs", "Could not get graph: ", -1, iResult)
  End if

  iResult = PGModel.CloseModel(0)
End Sub
'*****************************************************************************
'*****************************************************************************
Sub CreateBaseModelAndReplaceModel(oModel)
  Dim iResult
  Dim ElemId
  Dim ModelId
  Dim arrRights
  Dim i


  Call APILog("CreateBaseModelAndReplaceModel", "Starting to create BaseModel and Rep-
laceModel", 0, iResult)

   'Movid this to GetGraphsAndVersion
  'iResult = PGApplication.Authenticate("AdminUser", "", "")
  'if (iResult <> 0) Then
  '  Call APILog("CreateBaseModelAndReplaceModel", "Could not auteticate AdminUser: ", -
1, iResult)
  'End if

  iResult = PGApplication.CreateModel(True, "", ModelId)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not create a new model: ", -1,
iResult)
  End if

  iResult = PGModel.SetModelProperty("ModelVersionManagement", 1)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not set model property: ", -1,
iResult)
  End if

  iResult = PGModel.SaveServerModelAs("\\"&ReplacedModel)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not save replace model: ", -1,
iResult)
```

```
  End if

  'iResult = PGModel.CreateBranch("TestBranch", "", BranchId)
  'if (iResult <> 0) Then
  '  Call APILog("CreateBaseModelAndReplaceModel", "Could not create branch: ", -1, iResult)
  'End if

  iResult = PGModel.CloseModel(0)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not close model: ", -1, iResult)
  End if

  Call APILog("CreateBaseModelAndReplaceModel", "Created ReplaceModel. Starting to create BaseModel", 0, 0)

  iResult = PGApplication.CreateModel(True, "", oModel)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not create a new model: ", -1, iResult)
  End if

  iResult = PGModel.CreateElement("Subprocess", "Test SubProcess", "", ElemId)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not createElement: ", -1, iResult)
  End if

  Call SetObjRights(0, "ViewUser", "ApplyToChildLevels=true", 1)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not createElement: ", -1, iResult)
  End if

  Call SetObjRights(ElemId, "ViewUser", "ApplyToChildLevels=false", 0)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not set object rights 1: ", -1, iResult)
  End if

  Call SetObjRights(0, "UpdateUser", "ApplyToChildLevels=true", 3)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not set object rights 2: ", -1, iResult)
  End if

  Call SetObjRights(0, "AdminUser", "ApplyToChildLevels=true", 0)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not set object rights 3: ", -1, iResult)
  End if

  Call GetObjRights(ElemId)
```

```
  Call GetObjRights(0)

  Dim outValueVersion

  iResult = PGModel.SetModelProperty("MODELVERSIONMANAGEMENT", 0)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not set version management: ",
-1, iResult)
  End if

  iResult = PGModel.GetModelProperty("MODELVERSIONMANAGEMENT", outValu-
eVersion)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not get version management: ",
-1, iResult)
  End if

  if (""&outValueVersion <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Wrong version management: ", -1,
iResult)
  End if

  iResult = PGModel.SaveServerModelAs("\\"&BaseModel)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not save base model: ", -1,
iResult)
  End if

  Call GetSetModelProperty()

  iResult = oModel.CloseModel(0)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not close/save BaseModel: ", -
1, iResult)
  End if
End Sub
'*****************************************************************************
'*****************************************************************************
Sub CreateModel(oModel)
  Dim iResult
  Dim x
  Dim y
  Dim pass
  Dim fail
  Dim OrgId

  iResult = PGApplication.OpenServerModel("\\"&BaseModel, oModel)
  If (iResult <> 0) Then
    Call APILog("CreateModel", "Could not open server model: ", -1, iResult)
  else
    Call APILog("CreateModel", "User: "&stUser&" opened server model: ", 0, iResult)
  End If
```

```
  iResult = PGModel.CreateElement("Organization item type", "Test", "InstanceGraphical-
Properties=500,500,200,200", OrgId)
  If (iResult <> 0) Then
    if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
      fail = -1
    elseif(stUser = "ViewUser") Then
      fail = 0
    End if
    Call APILog("CreateModel", "Could not create Organization Item: ", fail, iResult)
  else
    if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
      pass = 0
    elseif(stUser = "ViewUser")Then
      pass = -1
    End if
    Call APILog("CreateModel", "User: "&stUser&" created Organization Item", pass, iResult)
  End If

  iResult = PGModel.CreateElement("Subprocess", "Main 1", "InstanceGraphicalProper-
ties=100,100,100,100;", iId)
  If (iResult <> 0) Then
    if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
      fail = -1
    elseif(stUser = "ViewUser") Then
      fail = 0
    End if
    Call APILog("CreateModel", "Could not create element 'Main 1': ", fail, iResult)
  else
    if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
      pass = 0
    elseif(stUser = "ViewUser")Then
      pass = -1
    End if
    Call APILog("CreateModel", "User: "&stUser&" created 1st Subprocess", pass, iResult)
  End If
  for y = 0 To 1
    iResult = PGModel.CreateElement("Subprocess", "Sub"& y, "ParentEle-
ment=iId;InstanceGraphicalProperties=150,150,150,150;", aId)
    If (iResult <> 0) Then
      if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
        fail = -1
      elseif(stUser = "ViewUser") Then
        fail = 0
      End if
      Call APILog("CreateModel", "Could not create element 'Sub"&y &"': ", fail, iResult)
    End If
    for x = 0 To 1
      iResult = PGModel.CreateElement("Activity", "Activity 1", "ParentEle-
ment=aId;InstanceGraphicalProperties=170,170,170,170;", zId)
      If (iResult <> 0) Then
        if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
          fail = -1
```

```
      elseif(stUser = "ViewUser") Then
        fail = 0
      End if
      Call APILog("CreateModel", "Could not create activity 'Activity 1': ", fail, iResult)
    End If

    iResult = PGModel.CreateElement("Activity", "Activity 2", "ParentElement="&aId&";",
xId)
    If (iResult <> 0) Then
      if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
        fail = -1
      elseif(stUser = "ViewUser") Then
        fail = 0
      End if
      Call APILog("CreateModel", "Could not create activity 'Activity 2': ", fail, iResult)
    End If
  Next
 Next

 iResult = PGModel.CreateElement("Subprocess", "Main 2", "InstanceGraphicalProper-
ties=70,70,70,70;", iId2)
 If (iResult <> 0) Then
  if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
    fail = -1
  elseif(stUser = "ViewUser") Then
    fail = 0
  End if
  Call APILog("CreateModel", "Could not create element 'Main 2': ", fail, iResult)
 else
  if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
    pass = 0
  elseif(seUser = "ViewUser")Then
    pass = -1
  End if
  Call APILog("CreateModel", "User: "&stUser&" created 2nd Subprocess", pass, iResult)
 End If
 for y = 0 To 1
  iResult = PGModel.CreateElement("Subprocess", "Sub"& y, "ParentEle-
ment=iId2;InstanceGraphicalProperties=70,70,70,70;", aId)
  If (iResult <> 0) Then
    if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
      fail = -1
    elseif(stUser = "ViewUser") Then
      fail = 0
    End if
    Call APILog("CreateModel", "Could not create element 'Sub"&y &"': ", fail, iResult)
  End If
  for x = 0 To 1
    iResult = PGModel.CreateElement("Activity", "Activity 1", "ParentEle-
ment=aId;InstanceGraphicalProperties=170,170,170,170;", zId)
    If (iResult <> 0) Then
      if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
        fail = -1
```

```
      elseif(stUser = "ViewUser") Then
        fail = 0
      End if
      Call APILog("CreateModel", "Could not create activity 'Activity 1': ", fail, iResult)
    End If

    iResult = PGModel.CreateElement("Activity", "Activity 2", "ParentEle-
ment=aId;InstanceGraphicalProperties=50,50,50,50;", xId)
    If (iResult <> 0) Then
      if(stUser = "AdminUser" OR stUser = "UpdateUser") Then
        fail = -1
      elseif(stUser = "ViewUser") Then
        fail = 0
      End if
      Call APILog("CreateModel", "Could not create activity 'Activity 2': ", fail, iResult)
    End If
  Next
 Next

 iResult = PGModel.SaveModel()
 if (iResult <> 0) Then
  Call APILog("CreateModel", "Could not save server model: ", -1, iResult)
 else
  Call APILog("CreateModel", "User: "&stUser&" saved server model", 0, iResult)
 End if

 iResult = PGModel.SaveFileModelAs(""&stPath&"\"&stUser&n&".pmf")
 if (iResult <> 0) Then
  if(stUser = "AdminUser") Then
   fail = -1
  elseif(stUser = "ViewUser" OR stUser = "UpdateUser") Then
   fail = 0
  End if
  Call APILog("CreateModel", "Could not save file model: ", fail, iResult)
 else
  if(stUser = "AdminUser") Then
   pass = 0
  else
   pass = -1
  End if
  Call APILog("CreateModel", "User: "&stUser&" saved file model", pass, iResult)
 End if

 iResult = PGModel.CloseModel(0)
 If (iResult <> 0) Then
  Call APILog("CreateModel", "Could not close server model: ", -1, iResult)
 else
  Call APILog("CreateModel", "Closed model", 0, iResult)
 End If

End Sub
'*****************************************************************************
'*****************************************************************************
```

```
Sub ExportImportXML()
  Dim ExportModel
  Dim iResult
  Dim pass
  Dim fail

  Call APILog("ExportImportXML", "Exporting model to XML", 0, iResult)

  iResult = PGApplication.OpenServerModel("\\"&BaseModel, oModel)
  If (iResult <> 0) Then
    Call APILog("ExportImportXML", "Could not open server model: ", -1, iResult)
  else
    Call APILog("ExportImportXML", "User: "&stDeleteUser&" opened server model: ", 0,
iResult)
  End If

  iResult = PGModel.ExecuteXMLTask("Export All.PGT",
""&stPath&"\"&stUser&n&".xml", "", "")
  if (iResult <> 0) Then
    if(stDeleteUser = "ViewUser") Then
      fail = 0
    elseif(stDeleteUser = "AdminUser"  OR stDeleteUser = "UpdateUser")Then
      fail = -1
    End if
    Call APILog("ExportImportXML", ""&stDeleteUser&" Could not export model to XML:
", fail, iResult)
  else
    if(stDeleteUser = "ViewUser")Then
      pass = -1
    else
      pass = 0
    End if
    Call APILog("ExportImportXML", "User: "&stDeleteUser&" exported model to XML",
pass, iResult)
  End if

  iResult = PGModel.CloseModel(1)
  if (iResult <> 0) Then
    Call APILog("ExportImportXML", "Could not close model: ", -1, iResult)
  else
    Call APILog("ExportImportXML", "Closed model", 0, iResult)
  End if

  Call APILog("ExportImportXML", "Importing model from XML", 0, iResult)

  iResult = PGApplication.OpenServerModel("\\"&BaseModel, oModel)
  If (iResult <> 0) Then
    Call APILog("ExportImportXML", "Could not open server model: ", -1, iResult)
  else
    Call APILog("ExportImportXML", "User: "&stDeleteUser&" opened server model: ", 0,
iResult)
  End If
```

```
  iResult = PGModel.ExecuteXMLTask("Import All.PGT",
""&stPath&"\"&stUser&n&".xml", "", "ReplaceCreate")
  if (iResult <> 0) Then
    if(stDeleteUser = "ViewUser" OR stDeleteUser = "UpdateUser") Then
      fail = 0
    elseif (stDeleteUser = "AdminUser") Then
      fail = -1
    End if
    Call APILog("ExportImportXML", "Could not import model: ", fail, iResult)
  else
    if(stDeleteUser = "ViewUser" OR stDeleteUser = "UpdateUser") Then
      pass = -1
    elseif(stDeleteUser = "AdminUser")Then
      pass = 0
    End if
    Call APILog("ExportImportXML", "User: "&stDeleteUser&" imported model from
XML", pass, iResult)
  End if

  iResult = PGModel.CloseModel(1)
  if (iResult <> 0) Then
    Call APILog("ExportImportXML", "Could not close model: ", -1, iResult)
  End if

End sub
'*****************************************************************************
'*****************************************************************************
Sub ReplaceModel()
  Dim iResult
  Dim pass
  Dim fail

  Call APILog("ReplaceModel", "Starting to replace model", 0, iResult)

  iResult = PGApplication.OpenServerModel("\\"&BaseModel, oModel)
  If (iResult <> 0) Then
    Call APILog("ReplaceModel", "Could not open server model: ", -1, iResult)
  else
    Call APILog("ReplaceModel", "User: "&stDeleteUser&" opened server model", 0, iResult)
  End If

  iResult = PGModel.ReplaceModel("\\"&ReplacedModel, "ReplaceUserRights=TRUE")
  if (iResult <> 0)Then
    if (stDeleteUser = "ViewUser" OR stDeleteUser = "UpdateUser") Then
      fail = 0
    elseif (stDeleteUser = "AdminUser") Then
      fail = -1
    End if
    Call APILog("ReplaceModel", "Could not replace model: ", fail, iResult)
  else
    if (stDeleteUser = "ViewUser" OR stDeleteUser = "UpdateUser") Then
      pass = -1
    elseif(stDeleteUser = "AdminUser")Then
```

```
    pass = 0
   End if
   Call APILog("ReplaceModel", "User: "&stDeleteUser&" replaced model", pass, iResult)
  End if

  iResult = PGModel.CloseModel(1)
  if (iResult <> 0) Then
   Call APILog("ReplaceModel", "Could not close model: ", -1, iResult)
  End if

End sub
'*****************************************************************************
'*****************************************************************************
Sub DeleteElement(oModel)
  Dim iResult
  Dim pass
  Dim fail

  iResult = PGApplication.OpenServerModel("\\"&BaseModel, oModel)
  if (iResult <> 0) Then
   Call APILog("DeleteElement", "Could not open server model: ", -1, iResult)
  else
   Call APILog("DeleteElement", "User: "&stDeleteUser&" opened server model", 0, iResult)
  End If

  iResult = PGModel.DeleteElement(iId, "")
  If (iResult <> 0) Then
   if (stDeleteUser = "ViewUser") Then
    pass = -1
    fail = 0
   elseif (stDeleteUser = "UpdateUser" OR stDeleteUser = "AdminUser") Then
    if (stUser = "AdminUser" OR stUser = "UpdateUser") Then
     fail = -1
    elseif (stUser = "ViewUser") Then
     fail = 0
    End if
   End if
   Call APILog("DeleteElement", "Could not delete 1st Subprocess: ", fail, iResult)
  else
   if (stDeleteUser = "ViewUser") Then
    pass = -1
   else
    pass = 0
   End if
   Call APILog("DeleteElement", "User: "&stDeleteUser&" deleted 1st Subprocess", pass,
iResult)
  End If

  iResult = PGModel.DeleteElement(iId2, "")
  If (iResult <> 0) Then
   if (stDeleteUser = "ViewUser") Then
    fail = 0
   elseif (stDeleteUser = "UpdateUser" OR stDeleteUser = "AdminUser") Then
```

```
    if (stUser = "AdminUser" OR stUser = "UpdateUser") Then
      fail = -1
    elseif (stUser = "ViewUser") Then
      fail = 0
    End if
  End if
  Call APILog("DeleteElement", "Could not delete 2nd Subprocess: ", fail, iResult)
 else
  if (stDeleteUser = "ViewUser") Then
    pass = -1
  elseif(stDeleteUser = "UpdateUser" OR stDeleteUser = "AdminUser") Then
    if (stUser = "AdminUser" OR stUser = "UpdateUser")Then
      pass = 0
    elseif(stUser = "ViewUser")Then
      pass -1
    End if
  End if
  Call APILog("DeleteElement", "User: "&stDeleteUser&" deleted 2nd Subprocess", pass,
iResult)
 End If

 iResult = PGModel.CloseModel(1)
 If (iResult <> 0) Then
  Call APILog("DeleteElement", "Could not close server model: ", -1, iResult)
 End If
End Sub
'*****************************************************************************
'*****************************************************************************
Sub DeleteModels(Model)
 Dim iResult
 Dim i
 Dim x
 Dim sModel
 Dim Test

 for i = 0 to 2
  if (i = 0) Then
    stDeleteUser = "ViewUser"
    Call AuthenticateDeleteUser()
  elseif (i = 1) Then
    stDeleteUser = "UpdateUser"
    Call AuthenticateDeleteUser()
  else
    stDeleteUser = "AdminUser"
    Call AuthenticateDeleteUser()
  End if

  iResult = PGApplication.OpenServerModel("\\"&Model, oModel)
  if (iResult <> 0) Then
    Call APILog("DeleteModels", "Could not open "&Model&": ", -1, iResult)
  End if

  iResult = PGModel.DeleteModel
```

```
    if (iResult <> 0) Then
      if (stDeleteUser = "ViewUser" OR stDeleteUser = "UpdateUser") Then
        Call APILog("DeleteModels", ""&stDeleteUser&" could not delete model", 0, 0)
      else
        Call APILog("DeleteModels", "AdminUser could not delete model: ", -1, iResult)
      End if
    else
      if (stDeleteUser = "ViewUser" OR stDeleteUser = "UpdateUser") Then
        Call APILog("DeleteModels", ""&stDeleteUser&" could delete model "&Model, -1, iResult)
      else
        Call APILog("DeleteModels", "AdminUser deleted model "&sModel, 0, 0)
      End if
    End if
  Next
End Sub
'*****************************************************************************
'*****************************************************************************
Sub SetObjRights(ElemId, User, Param, Rights)
  Dim iResult

  iResult = PGModel.SetObjectRights(ElemId, User, Param, Rights)
  if (iResult <> 0) Then
    if (""&User = "AdminUser") Then
      Call APILog("SetObjRights", "Could not set rights for AdminUser", 0, 0)
    else
      Call APILog("SetObjRights", "Could not set object rights to "&User&": ", -1, iResult)
    End if
  elseif (iResult = 0 AND ""&User = "AdminUser") Then
    Call APILog("SetObjRights", "Object rights set for AdminUser!!!", -1, 0)
  End if
End Sub
'*****************************************************************************
'*****************************************************************************
Sub GetObjRights(ElemId)
  Dim iResult
  Dim i
  Dim arrRights

  iResult = PGModel.GetObjectRights(ElemId, "IncludeInheritedRights=false", arrRights)
  if (iResult <> 0) Then
    Call APILog("CreateBaseModelAndReplaceModel", "Could not get object rights: ", -1, iResult)
  End if

  for i = lbound(arrRights, 1) to uBound(arrRights, 1)
    if (""&arrRights(i,1) = "ViewUser") Then
      if (ElemId = 0) Then
        if (""&arrRights(i,3) = 1) Then
          Call APILog("GetObjRights", "ViewUser has correct rights to main process level", 0, 0)
        else
          Call APILog("GetObjRights", "ViewUser has incorrect rights to main process level. AccessLevel: "&arrRights(i,3), -1, 0)
```

```
      End if
    else
      if (""&arrRights(i,3) = 0) Then
        Call APILog("CreateBaseModelAndReplaceModel", "ViewUser has correct rights to
subprocess", 0, 0)
      else
        Call APILog("CreateBaseModelAndReplaceModel", "ViewUser has incorrect rights to
subprocess. AccessLevel: "&arrRights(i,3), -1, 0)
      End if
    End if
  elseif(""&arrRights(i,1) = "UpdateUser") Then
    if (""&arrRights(i,3) = 3) Then
      Call APILog("CreateBaseModelAndReplaceModel", "UpdateUser has correct rights to
subprocess", 0, 0)
    else
      Call APILog("CreateBaseModelAndReplaceModel", "UpdateUser has incorrect rights to
subprocess. AccessLevel: "&arrRights(i,3), -1, 0)
    end if
  elseif(""&arrRights(i,1) = "AdminUser") Then
    if (""&arrRights(i,3) = 4) Then
      Call APILog("CreateBaseModelAndReplaceModel", "AdminUser has correct rights to
subprocess", 0, 0)
    else
      Call APILog("CreateBaseModelAndReplaceModel", "AdminUser has incorrect rights to
subprocess. AccessLevel: "&arrRights(i,3), -1, 0)
    End if
  End if
 Next
End Sub
'*****************************************************************************
'*****************************************************************************
Sub GetSetModelProperty()
 Dim iResult
 Dim number
 Dim z
 Dim PropertyName(6)
 Dim PropertyValue(3)
 Dim outValue

 Call APILog("GetSetModelProperty", "Starting to set/get model properties", 0, 0)

 PropertyName(0) = "VersionNumber"
 PropertyName(1) = "Status"
 PropertyName(2) = "Author"
 PropertyName(3) = "Comment"
 PropertyName(4) = "LastChanged"
 PropertyName(5) = "ModelCreated"
 PropertyName(6) = "LastSaved"
 PropertyValue(0) = "1.123"
 PropertyValue(1) = "0"
 PropertyValue(2) = "qpr"
 PropertyValue(3) = "This comment is created via API"
```

```
  for z = 0 to 3
    iResult = PGModel.SetModelProperty(""&PropertyName(z), ""&PropertyValue(z))
    if (iResult <> 0) Then
      Call APILog("GetSetModelProperty", "Could not set model property
"&PropertyName(z)&": ", -1, iResult)
    End if

    iResult = PGModel.GetModelProperty(""&PropertyName(z), outValue)
    if (iResult <> 0) Then
      Call APILog("GetSetModelProperty", "Could not get model property
"&PropertyName(z)&": ", -1, iResult)
    End if

    if (""&PropertyValue(z) = ""&outValue) Then
      Call APILog("GetSetModelProperty", "Get/Set had same value", 0, 0)
    else
      Call APILog("GetSetModelProperty", "Get/Set had different value!", -1, 0)
    End if
  next

  for z = 4 to 6
    iResult = PGModel.GetModelProperty(""&PropertyName(z), outValue)
    if (iResult <> 0) Then
      Call APILog("GetSetModelProperty", "Could not get model property
"&PropertyName(z)&": ", -1, iResult)
    End if
  next
End Sub
'*****************************************************************************
'*****************************************************************************
Sub OpenModel()
  Dim oModel
  Dim iResult

  Call APILog("OpenModel", "Starting to open the model", 0, iResult)

  iResult = PGApplication.OpenServerModel("\\Dentorex Group Processes", oModel)
  if (iResult <> 0) Then
    Call APILog("OpenModel", "Could not open model: ", -1, iResult)
  End if

End Sub
'*****************************************************************************
'*****************************************************************************
Sub ProcessStepTest()
  Dim values(21,0)
  Dim sCase
  Dim iResult

  sCase = "ProcessStepTest"

  Call APILog(""&sCase, "Starting to test Process Step", 0, iResult)
```

```
  Call ProcessStepArray(values)

  Call GetElementIds("KIND=ELEMENTS;ELEMENTTYPES=ALLPROCESSSTEPS")
  'Call GetElementIds("KIND=ELEMENTS;ELEMENTTYPES=ALLSUBPROCESSES")

  Call GetElementProperties(values, sCase)

  Call APILog(""&sCase, "Process Step test completed", 0, iResult)
End Sub
'*************************************************************************
'*************************************************************************
Sub FlowTest()
  Dim values(14,0)
  Dim sCase
  Dim iResult

  sCase = "FlowTest"

  Call APILog(""&sCase, "Starting to test Flow", 0, iResult)

  Call FlowArray(values)

  Call GetElementIds("KIND=ELEMENTS;ELEMENTTYPES=ALLFLOWS")

  Call GetElementProperties(values, sCase)

  Call APILog(""&sCase, "Flow test completed", 0, iResult)
End Sub
'*************************************************************************
'*************************************************************************
Sub OrganizationItemTest()
  Dim values(4,0)
  Dim sCase
  Dim iResult
  Dim outValue

  sCase = "OrganizationItemTest"

  Call APILog(""&sCase, "Starting to test Organization Item", 0, iResult)

  Call OrganizationItemArray(values)

  Call GetElementTypeId(values, sCase, "Organization item type")

  Call APILog(""&sCase, "Organization item test completed", 0, iResult)
End Sub
'*************************************************************************
'*************************************************************************
Sub CaseTest()
  Dim values(1,0)
  Dim sCase
  Dim iResult
```

```
  sCase = "CaseTest"

  Call APILog(""&sCase, "Starting to test Case", 0, iResult)

  Call CaseArray(values)

  Call GetElementTypeId(values, sCase, "Case type")

  Call APILog(""&sCase, "Case test completed", 0, iResult)
End Sub
'*************************************************************************
'*************************************************************************
Sub InformationItemTest()
  Dim values(8,0)
  Dim sCase
  Dim iResult

  sCase = "InformationItemTest"

  Call APILog(""&sCase, "Starting to test Information Item", 0, iResult)

  Call InformationItemArray(values)

  Call GetElementTypeId(values, sCase, "Information item type")

  Call APILog(""&sCase, "Information item test completed", 0, iResult)
End Sub
'*************************************************************************
'*************************************************************************
Sub FlowCompositionTest()
  Dim values(3,0)
  Dim sCase
  Dim iResult

  sCase = "FlowCompositionTest"

  Call APILog(""&sCase, "Starting to test Flow Composition", 0, iResult)

  Call FlowCompositionArray(values)

  Call GetElementTypeId(values, sCase, "Composition flow type")

  Call APILog(""&sCase, "Composition Flow test completed", 0, iResult)
End Sub
'*************************************************************************
'*************************************************************************
Sub MeasureTest()
  Dim values(4,0)
  Dim sCase
  Dim iResult

  sCase = "MeasureTest"
```

```
  Call APILog(""&sCase, "Starting to test Measure", 0, iResult)

  Call MeasureArray(values)

  Call GetElementTypeId(values, sCase, "Measure type")

  Call APILog(""&sCase, "Measure test completed", 0, iResult)
End Sub
'*************************************************************************
'*************************************************************************
Sub MeasureDataTest()
  Dim values(2,0)
  Dim sCase
  Dim iResult

  sCase = "MeasureDataTest"

  Call APILog(""&sCase, "Starting to test Measure Data", 0, iResult)

  Call MeasureDataArray(values)

  Call GetElementTypeId(values, sCase, "Measure data type")

  Call APILog(""&sCase, "Measure Data test completed", 0, iResult)
End Sub
'*************************************************************************
'*************************************************************************
Sub NoteTest()
  Dim values(2,0)
  Dim sCase
  Dim iResult

  sCase = "NoteTest"

  Call APILog(""&sCase, "Starting to test Note", 0, iResult)

  Call NoteArray(values)

  Call GetElementTypeId(values, sCase, "Note type")

  Call APILog(""&sCase, "Note test completed", 0, iResult)
End Sub
'*************************************************************************
'*************************************************************************
Sub ModelTest()
  Dim values(11,0)
  Dim sCase
  Dim iResult
  Dim Id

  sCase = "ModelTest"

  Call APILog(""&sCase, "Starting to test Model", 0, iResult)
```

```
  Call ModelArray(values)

  iResult = PGModel.GetActive("PROCESSMODEL", Id)
  if (iResult <> 0) Then
    Call APILog(""&sCase, "Could not get id: ", -1, iResult)
  End if

  Call GetSingleElementProperty(Id, values, sCase)

  Call APILog(""&sCase, "Model test completed", 0, iResult)
End Sub
'*****************************************************************************
'*****************************************************************************
Sub ResourceTest()
  Dim values(19,0)
  Dim sCase
  Dim iResult

  sCase = "ResourceTest"

  Call APILog(""&sCase, "Starting to test Resource", 0, iResult)

  Call ResourceArray(values)

  Call GetElementTypeId(values, sCase, "Resource type")

  Call APILog(""&sCase, "Resource test completed", 0, iResult)
End Sub
'*****************************************************************************
'*****************************************************************************
Sub StoreTest()
  Dim values(4,0)
  Dim sCase
  Dim iResult

  sCase = "StoreTest"

  Call APILog(""&sCase, "Starting to test Store", 0, iResult)

  Call StoreArray(values)

  Call GetElementTypeId(values, sCase, "Store type")

  Call APILog(""&sCase, "Store test completed", 0, iResult)
End Sub
'*****************************************************************************
'*****************************************************************************
Sub GetElementProperties(values, sCase)
  Dim i, z, x
  Dim ElemId
  Dim sProperty
  Dim outValue
```

```
  Dim iResult
  Dim outArray

  for i = lbound(outIds) to uBound(outIds)
    ElemId = ""&outIds(i)
    for z = 0 to 15
      iResult = PGModel.GetProperty(ElemId, ""&ShareArray(z,0), outValue)
      if (iResult <> 0) Then
        Call APILog("GetElementProperties - "&sCase, "Could not get "&ShareArray(z,0), -1,
iResult)
      End if
    Next
    for z = lbound(values, 1) to uBound(values, 1)
      iResult = PGModel.GetProperty(ElemId, ""&values(z,0), outValue)
      if (iResult <> 0) Then
        Call APILog("GetElementProperties - "&sCase, "Could not get "&values(z,0), -1, iRe-
sult)
      End if
    Next

    iResult = PGModel.GetProperties(ElemId, outArray)
    if (iResult <> 0) Then
      Call APILog("GetElementProperties", "Could not get properties: ", -1, iResult)
    End if
  Next
End Sub
'*************************************************************************
'*************************************************************************
Sub GetSingleElementProperty(Id, values, sCase)
  Dim iResult
  Dim i
  Dim outValue
  Dim outArray

  for i = 0 to 15
    iResult = PGModel.GetProperty(Id, ""&ShareArray(i,0), outValue)
    if (iResult <> 0) Then
      Call APILog("GetSingleElementProperty", "Could not get "&ShareArray(i,0), -1, iResult)
    End if
  next

  for i = lbound(values) to ubound(values)
    iResult = PGModel.GetProperty(Id, ""&values(i,0), outvalue)
    if (iResult <> 0) Then
      Call APILog("GetSingleElementProperty", "Could not get "&values(i,0), -1, iResult)
    End if
  next

  iResult = PGModel.GetProperties(Id, outArray)
  if (iResult <> 0) Then
    Call APILog("GetSingleElementProperty", "Could not get properties: ", -1, iResult)
  End if
End Sub
```

```
'*****************************************************************************
'*****************************************************************************
Function GetElementIds(Parameters)
  Dim iResult

  iResult = PGModel.Find(Parameters, "", "", outIds)
  if (iResult <> 0) Then
    Call APILog("GetElementIds", "Could not find anything: ", -1, iResult)
  End if

End Function
'*****************************************************************************
'*****************************************************************************
Function GetElementTypeId(values, sCase, sType)
  Dim iResult
  Dim outArray
  Dim outValue
  Dim i
  Dim ElemId
  Dim z
  Dim outValue2
  Dim outArray2

  z = 0

  iResult = PGModel.Find("KIND=ELEMENTTYPES;", "", "", outArray)
  if (iResult <> 0) Then
    Call APILog("GetElementTypeId", "Could not find anything: ", -1, iResult)
  End if

  for i = lbound(outArray) to ubound(outArray)
    ElemId = ""&outArray(i)

    iResult = PGModel.GetProperty(ElemId, "Name", outValue)
    if (iResult <> 0) Then
      Call APILog("GetElementTypeId", "Could not get property: ", -1, iResult)
    End if

    if (""&outvalue = ""&sType) Then
      ElemTypeId = ""&ElemId
    End if
  next

  iResult = PGModel.Find("", "", "", outArray)
  if (iResult <> 0) Then
    Call APILog("GetElementTypeId", "Could not find anything 2: ", -1, iResult)
  End if

  for i = lbound(outArray) to ubound(outArray)
    ElemId = outArray(i)

    iResult = PGModel.GetProperty(ElemId, "ElementTypeId", outValue)
    if (iResult <> 0) Then
```

```
    Call APILog("GetElementTypeId", "Could not get elementtype property: ", -1, iResult)
   End if

   if (""&outValue = ""&ElemTypeId) Then
    for z = 0 to 15
     iResult = PGModel.GetProperty(ElemId, ""&ShareArray(z,0), outValue2)
     if (iResult <> 0) Then
      Call APILog("GetElementTypeId - "&sCase, "Could not get "&ShareArray(z,0), -1,
iResult)
     End if
    next
    for z = lbound(values) to ubound(values)
     iResult = PGModel.GetProperty(ElemId, ""&values(z,0), outValue2)
     if (iResult <> 0) Then
      Call APILog("GetElementTypeId - "&sCase, "Could not get "&values(z,0), -1, iResult)
     End if
    next

    iResult = PGModel.GetProperties(ElemId, outArray2)
    if (iResult <> 0) Then
     Call APILog("GetElementTypeId", "Could not get properties: ", -1, iResult)
    End if
   End if
  next
End Function
'*************************************************************************
'*************************************************************************
Function ArrayForAll(Array)
  Array(0,0) = "Id"
  Array(1,0) = "ParentId"
  Array(2,0) = "ElementTypeId"
  Array(3,0) = "Name"
  Array(4,0) = "Description"
  Array(5,0) = "Instances"
  Array(6,0) = "InstanceGraphicalProperties"
  Array(7,0) = "InformationItems"
  Array(8,0) = "Notes"
  Array(9,0) = "MeasureData"
  Array(10,0) = "Resources"
  Array(11,0) = "Priority"
  Array(12,0) = "CanBeSuspended"
  Array(13,0) = "ScorecardModel"
  Array(14,0) = "ScorecardSymbol"
  Array(15,0) = "MeasureSymbol"
  Array(16,0) = "Symbol"
End Function
'*************************************************************************
'*************************************************************************
Function ProcessStepArray(values)
  values(0,0) = "InstanceIds"
  values(1,0) = "IncomingFlows"
  values(2,0) = "OutGoingFlows"
  values(3,0) = "Owner"
```

```
  values(4,0) = "ProcessStepsOnProcessLevel"
  values(5,0) = "Groups"
  values(6,0) = "Flows"
  values(7,0) = "Stores"
  values(8,0) = "FlowCompositions"
  values(9,0) = "SimulationEntryRule"
  values(10,0) = "EntryRules"
  values(11,0) = "EntryRuleCaseMatching"
  values(12,0) = "SimulationExitRule"
  values(13,0) = "ExitRules"
  values(14,0) = "ProcessingTimeType"
  values(15,0) = "ProcessingTimes"
  values(16,0) = "InputFrequencyType"
  values(17,0) = "InputFrequencies"
  values(18,0) = "IsStartingProcessStep"
  values(19,0) = "MaxInputs"
  values(20,0) = "ModelingDirection"
  values(21,0) = "IsUseDefaultModelingDirection"
End Function
'*************************************************************************
'*************************************************************************
Function FlowArray(values)
  values(0,0) = "FlowObject"
  values(1,0) = "From"
  values(2,0) = "FromInstance"
  values(3,0) = "To"
  values(4,0) = "ToInstance"
  values(5,0) = "Owner"
  values(6,0) = "Document"
  values(7,0) = "ProcessingTimeType"
  values(8,0) = "ProcessingTimes"
  values(9,0) = "InputFrequencyType"
  values(10,0) = "InputFrequencies"
  values(11,0) = "IsStartingFlow"
  values(12,0) = "MaxInputs"
  values(13,0) = "IsDocument"
  values(14,0) = "ProcessLevels"
End Function
'*************************************************************************
'*************************************************************************
Function OrganizationItemArray(values)
  values(0,0) = "ParentOrganizationUnit"
  values(1,0) = "ChildOrganizationUnits"
  values(2,0) = "ProcessSteps"
  values(3,0) = "Persons"
  values(4,0) = "OrganizationType"
End Function
'*************************************************************************
'*************************************************************************
Function CaseArray(values)
  values(0,0) = "SimulationStarted"
  values(1,0) = "SimulationEnded"
End Function
```

```
'*************************************************************************
'*************************************************************************
Function InformationItemArray(values)
  values(0,0) = "ModelObjects"
  values(1,0) = "Flows"
  values(2,0) = "Links"
  values(3,0) = "IsEmbedded"
  values(4,0) = "EmbeddedInformation"
  values(5,0) = "DocumentType"
  values(6,0) = "ChildInformationItems"
  values(7,0) = "ParentInformationItems"
  values(8,0) = "FlowCompositions"
End Function
'*************************************************************************
'*************************************************************************
Function FlowCompositionArray(values)
  values(0,0) = "ProcessLevel"
  values(1,0) = "Document"
  values(2,0) = "ChildCompositions"
  values(3,0) = "ParentComposition"
End Function
'*************************************************************************
'*************************************************************************
Function MeasureArray(values)
  values(0,0) = "MeasureValueType"
  values(1,0) = "MeasureSubprocessCalculationType"
  values(2,0) = "MeasureUnitType"
  values(3,0) = "UnitNames"
  values(4,0) = "LinkedMeasureData"
End Function
'*************************************************************************
'*************************************************************************
Function MeasureDataArray(values)
  values(0,0) = "Measure"
  values(1,0) = "MeasureDataList"
  values(2,0) = "ModelObject"
End Function
'*************************************************************************
'*************************************************************************
Function NoteArray(values)
  values(0,0) = "ParentNote"
  values(1,0) = "ChildNotes"
  values(2,0) = "ModelObject"
End Function
'*************************************************************************
'*************************************************************************
Function ModelArray(values)
  values(0,0) = "FeedbackEmailAddress"
  values(1,0) = "CreationDate"
  values(2,0) = "ModificationDate"
  values(3,0) = "Template"
  values(4,0) = "ChangeLog"
  values(5,0) = "IsUseLog"
```

```
  values(6,0) = "IsForceExclusive"
  values(7,0) = "ModelingOptions"
  values(8,0) = "BaseModelingOptions"
  values(9,0) = "ItemsInLog"
  values(10,0) = "IsPublished"
  values(11,0) = "AutomaticInformationItemRefresh"
End Function
'*************************************************************************
'*************************************************************************
Function ResourceArray(values)
  values(0,0) = "ResourceType"
  values(1,0) = "AmountOfResourceAvailable"
  values(2,0) = "UnitCost"
  values(3,0) = "UnitName"
  values(4,0) = "MaximumAmountInStore"
  values(5,0) = "OrderingLimit"
  values(6,0) = "IncrementPerRegularOrder"
  values(7,0) = "RegularOrderInterval"
  values(8,0) = "IncrementPerOrder"
  values(9,0) = "OrderingCost"
  values(10,0) = "OrderingTime"
  values(11,0) = "TimeUnit"
  values(12,0) = "ObjectsUsing"
  values(13,0) = "ChildResources"
  values(14,0) = "ParentResources"
  values(15,0) = "ResourcesInResourcePool"
  values(16,0) = "InResourcePools"
  values(17,0) = "OrganizationUnit"
  values(18,0) = "ResponsibleOfFlows"
  values(19,0) = "ProcessLevel"
End Function
'*************************************************************************
'*************************************************************************
Function StoreArray(values)
  values(0,0) = "InstanceIds"
  values(1,0) = "IncomingFlows"
  values(2,0) = "OutgoingFlows"
  values(3,0) = "StoreType"
  values(4,0) = "SimulationRules"
End Function
'*************************************************************************
'*************************************************************************
'*************************************************************************************
'*************************************************************************************
Sub APILog(log1, log2, error, iResult)
  Dim objFSO, objFolder, objShell, objTextFile
  Dim dDate
  Dim dTime
  Dim strFile
  Dim objFSO2, objFolder2, objShell2, strDirectory2

  strDirectory2 = "C:\Regression"
```

```
 if (""&log2 = "Starting to run 'PG_UserRights.pmf"") Then
  Set objFSO2 = CreateObject("Scripting.FileSystemObject")
  if objFSO2.FolderExists(strDirectory2) Then
   Set objFolder2 = objFSO2.GetFolder(strDirectory2)
   Call APILog("APILog", "Folder has already been created", 0, 0)
  else
   Set objFolder2 = objFSO2.CreateFolder(strDirectory2)
   Call APILog("APILog", "Created folder : "&strDirectory2, 0, 0)
  End if
 End if

 Dim i

 strFile = "APILog.txt"

 dTime = Time
 dDate = Date

 if (error = -1 OR error = "") Then
  ApiResult = ApiResult + 1

  'MsgBox ""&log1&", "&log2&": " + PGApplication.GetErrorMessage(iResult)
 End If

 Set objFSO = CreateObject("Scripting.FileSystemObject")

 If objFSO.FileExists(""&strFile) = 0 Then
  Set objTextFile = objFSO.CreateTextFile(""&strFile)
  objTextFile.WriteLine(""&dTime &"                    "&dDate &"  "&log1 &"     "&log2 +
PGApplication.GetErrorMessage(iResult))
  if (APIResult <> 0) Then
   objTextFile.WriteLine(""&dTime &"                    "&dDate &"  Error:"&log1 &"
              "&log2 + PGApplication.GetErrorMessage(iResult))
   'PGApplication.Quit(0)
  End if
  objTextFile.Close
 Else
  Set objTextFile = objFSO.OpenTextFile(""&strFile, 8, True)
  objTextFile.WriteLine(""&dTime &"                    "&dDate &"  "&log1 &"     "&log2 +
PGApplication.GetErrorMessage(iResult))
  if (APIResult <> 0) Then
   objTextFile.WriteLine(""&dTime &"                    "&dDate &"  Error:"&log1 &"
               "&log2 + PGApplication.GetErrorMessage(iResult))
              'TEST
              APIResult = 0
              'End Testi
   'PGApplication.Quit(0)
  End if
  objTextFile.Close
 End If

 if (""&log2 = "Completed PG_Advanced") Then
  Set objFSO = CreateObject("Scripting.FileSystemObject")
```

```
   Set objTextFile = objFSO.OpenTextFile(""&strFile, 8, True)
   if (ApiResult = 0) Then
     objTextFile.WriteLine(""&dTime &"                "&dDate &"                All OK!")
   else
     objTextFile.WriteLine(""&dTime &"                "&dDate &"                Error:
Errors occurred while executing script!")
   End if
   objTextFile.Close
  End if

End Sub
'*************************************************************************************
'*************************************************************************************
```

```
<QPR_SCRIPT_FILE language = vbscript TimeOut=0>

Option Explicit

Private stUser                          'Username
Private stUserPw            'Users password
Private y
Private iResult
Private ViewUser
Public completed
Private objFSO, objFolder, objShell, strDirectory
Dim RangeId
Dim RecArray()
Dim AlertId
Dim iSCId
Dim iChildSCId
Dim PropSCId
Dim ElementId
Dim PropElemId
Dim SeriesId
Dim TopElementId
Dim InfoItemId

  stUser = "qpr"
  stUserPw = "demo"
  y = 0
  completed = True
  ViewUser = "ViewUser"
  strDirectory = "c:\Regression"

Private SCName              'Helps identificate Scorecards
'*****************************************************
'*****************************************************
```

```
Sub Main()

  Dim n
  Dim i
  Dim oModel
  Dim sModelName
  Dim bCalculation

  i = 0
  n = 0
  bCalculation = False

  Call APILog("Main", "Starting to run SC_Advanced", 0)

  iResult = SCApplication.SetUIMode(0)

  Set objFSO = CreateObject("Scripting.FileSystemObject")
  if objFSO.FolderExists(strDirectory) Then
    Set objFolder = objFSO.GetFolder(strDirectory)
    Call APILog("Main", "Folder has already been created", 0)
  else
    Set objFolder = objFSO.CreateFolder(strDirectory)
    Call APILog("Main", "Created folder : "&strDirectory, 0)
  End if

  Call ConnectToServerAndAuthenticate()

  Call Applications()

  'Call ExternalScript()

  Call VarialTest()

  '*************************
  'SC26398B
```

```
'Call SaveAndMerge(2)
'************************

Call MiscTest()

Call OpenModel(oModel, sModelName)

Call Calculations(bCalculation)

i = 0
do while i < 4
  if (i=0) Then
    SCName = "NONE"
  elseif (i=1) Then
    SCName = "VIEW"
  elseif (i=2) Then
    SCName = "UPDATE"
  elseif (i=3) Then
    SCName = "ADMIN"
  End if
  Call CreateSCElements()
  i = i + 1
loop

Call Calculations(bCalculation)

Call Authenticate()

Call OpenModel(oModel, sModelName)

Call CopyScorecards()

Call Modify()

Call DeleteScorecards(ViewUser)
```

```
Call DeleteModel()

Call ConnectToServerAndAuthenticate

Call OpenModel(oModel, sModelName)

Call DeleteScorecards(stUser)

'*************************
'SC26398B
'Call SaveAndMerge(1)
'*************************

'*************************
'SC26398B
'Call DeleteModels()
'*************************

  Call CreateModel()

Call CreateSCElement()

Call Scorecard()

Call Element()

Call PeriodLevel()

Call Range()

Call ChartTemplate()

Call Series()
```

```
Call ValueSetting()

Call InformationItem()

Call MeasureUnit()

'********************
'BUG: SMTP Error!
'Call Alert()
'********************

iResult = SCApplication.OpenModel("Dentorex Group Scorecard", oModel)
if (iResult <> 0) Then
  Call APILog("External - Main", "Could not open Dentorex", -1)
End if

'Call LinkedData()

iResult = SCApplication.OpenModel("SetGetPropertyTest", oModel)
if (iResult <> 0) Then
  Call APILog("External - Main", "Could not open SetGetPropertyTest", -1)
End if

iResult = SCModel.DeleteModel
if (iResult <> 0) Then
  Call APILog("External - Main", "Could not delete model: ", -1)
End if

if objFSO.FolderExists(strDirectory) Then
  objFSO.DeleteFolder(strDirectory)
  Call APILog("Main", "Deleted directory", 0)
else
  Call APILog("Main", "Directory not found", -1)
End if
```

```
  Call APILog("Main", "SC_Advanced complete", 0)

  SCApplication.Quit

End Sub
'*****************************************************
'*****************************************************
Sub ConnectToServerAndAuthenticate()

  iResult = SCApplication.ConnectToServer("localhost", <PortN>)
  if (iResult <> 0) Then
    Call APILog("ConnectToServerAndAuthenticate", "Could not connect to server: ", -1)
  End if

  iResult = SCApplication.Authenticate(""&stUser, ""&stUserPw, "")
  if (iResult <> 0) Then
    iResult = SCApplication.WriteLog("API: ConnectToServerAndAuthenticate", "Authentica-
tion failed", -1)
    Call APILog("ConnectToServerAndAuthenticate", "Authentication failed: ", -1)
  End if

End Sub
'*****************************************************
'*****************************************************
Sub Authenticate()

  Call APILog("Authenticate", "Autenticating 'ViewUser'", 0)

  iResult = SCApplication.Authenticate(""&ViewUser, "", "")
  if (iResult <> 0) Then
    iResult = SCApplication.WriteLog("API: ConnectToServerAndAuthenticate", "Authentica-
tion failed", -1)
    Call APILog("Authenticate", "Authentication failed: ", -1)
  End if
```

```
End Sub
'*****************************************************
'*****************************************************
Sub Applications()
  Dim AppProperty
  Dim PropertyName
  Dim i
  Dim Models()
  Dim ModelName
  Dim NeedModel
  Dim UserGroupArray()
  Dim UserId
  Dim UserName
  Dim UserInfoArray(0,3)

  NeedModel = FALSE

  for i = 0 to 2
   if (i = 0) Then
    PropertyName = "NAME"
   elseif (i = 1) Then
    PropertyName = "VERSION"
   else
    PropertyName = "EXEPATH"
   End if

   iResult = SCApplication.GetProperty(PropertyName, AppProperty)
   if (iResult <> 0) Then
    Call APILog("Applications", "Could not get property: ", -1)
   End if

   Call APILog("Applications", ""&PropertyName &": "&AppProperty, 0)
  Next

  i = 0
```

```
iResult = SCApplication.GetModels(Models)
if (iResult <> 0) Then
  Call APILog("Applications", "Could not get models: ", -1)
End if

for I = lbound(Models, 1) to ubound(Models, 1)
  ModelName = Models(i, 0)
  if (ModelName = "Government") Then
    NeedModel = TRUE
  End if
Next

if (NeedModel = FALSE) Then
  Call APILog("Applications", "Government - model not found. Test will fail.", -1)
else
  Call APILog("Applications", "Government - model found", 0)
End if

iResult = SCApplication.GetGroupInfo(-1, UserGroupArray)
if (iResult <> 0) Then
  Call APILog("Applications", "Could not get group info: ", -1)
End if

iResult = SCApplication.GetuserId(stUser, UserId)
if (iResult <> 0) Then
  Call APILog("Applications", "Could not get User Id: ", -1)
End if

iResult = SCApplication.GetuserName(UserId, UserName)
if (iResult <> 0) Then
  Call APILog("Applications", "Could not get user name: ", -1)
End if

if(stUser = UserName) Then
```

```
  Call APILog("Applications", "User is correct", 0)
 else
  Call APILog("Applications", "Wrong user in use. Some error in script: ", -1)
 End if

 iResult = SCApplication.GetUserInfo(userId, UserInfoArray)
 if (iResult <> 0) Then
  Call APILog("Applications", "Could not get user info", -1)
 End if

 if (userInfoArray(0,0) = UserId AND UserInfoArray(0,1) = stUser) Then
  Call APILog("Applications", "User name and id matches", 0)
 else
  Call APILog("Applications", "User name and Id doesn't match: ", -1)
 End if
End sub
'*****************************************************
'*****************************************************
Sub SaveAndMerge(number)
 Dim ModelId
 Dim ImportedId

 Call APILog("SaveAndMerge", "Saving and merging models", 0)

 iResult = SCApplication.CreateModel("Merge1", "Government", ModelId)
 if (iResult <> 0) Then
  Call APILog("SaveAndMerge", "Could not create a new model", -1)
 End if

 iResult = SCModel.SaveAs("Merge2")
 if (iResult <> 0) Then
  Call APILog("SaveAndMerge", "Could not save model: ", -1)
 End if

 iResult = SCApplication.OpenModel("Merge2", ModelId)
```

```
if (iResult <> 0) Then
  Call APILog("SaveAndMerge", "Could not open model: ", -1)
End if

'SCApplication.sleep(10000)

iResult = SCModel.ExportModel(""&strDirectory &"\Merge.smf")
if (iResult <> 0) Then
  Call APILog("SaveAndMerge", "Could not export model: ", -1)
End if

iResult = SCApplication.OpenModel("Merge1", ModelId)
if (iResult <> 0) Then
  Call APILog("SaveAndMerge", "Could not open model: ", -1)
End if

iResult = SCModel.MergeModel(""&strDirectory &"\Merge.smf")
if (iResult <> 0) Then
  Call APILog("SaveAndMerge", "Could not merge models: ", -1)
End if

if (number = 2) Then
  iResult = SCApplication.OpenModel("Merge1", ModelId)
  if (iResult <> 0) Then
    Call APILog("DeleteModels", "Could not open model: ", -1)
  End if

  iResult = SCModel.DeleteModel
  if (iResult <> 0) Then
    Call APILog("DeleteModels", "Could not delete model", -1)
  End if

  iResult = SCApplication.OpenModel("Merge2", ModelId)
  if (iResult <> 0) Then
    Call APILog("DeleteModels", "Could not open model: ", -1)
```

```
    End if

    iResult = SCModel.DeleteModel
    if (iResult <> 0) Then
      Call APILog("DeleteModels", "Could not delete model", -1)
    End if
  End if
End Sub
'*****************************************************
'*****************************************************
Sub DeleteModels()
  Dim iResult
  Dim ModelId

  iResult = SCApplication.OpenModel("Merge1", ModelId)
  if (iResult <> 0) Then
    Call APILog("DeleteModels", "Could not open model: ", -1)
  End if

  iResult = SCModel.DeleteModel
  if (iResult <> 0) Then
    Call APILog("DeleteModels", "Could not delete model", -1)
  End if

  iResult = SCApplication.OpenModel("Merge2", ModelId)
  if (iResult <> 0) Then
    Call APILog("DeleteModels", "Could not open model: ", -1)
  End if

  iResult = SCModel.DeleteModel
  if (iResult <> 0) Then
    Call APILog("DeleteModels", "Could not delete model", -1)
  End if

End Sub
```

```
'*****************************************************
'*****************************************************
Sub OpenModel(voModel, vsModelName)
  Dim iResult
  Dim oModel
  Dim Model

  Model = "Government"

  iResult = SCApplication.OpenModel("Government", oModel)
  if (iResult <> 0) Then
    Call APILog("OpenModel", "Could not open model using name ", -1)
  End if

  iResult = SCModel.CloseModel
  if (iResult <> 0) Then
    Call APILog("OpenModel", "Could not close model", -1)
  End if

  iResult = SCApplication.OpenModel(Model, oModel)
  if (iResult <> 0) Then
    Call APILog("OpenModel", "Could not open model using varial", -1)
  End if

  iResult = SCModel.CloseModel
  if (iResult <> 0) Then
    Call APILog("OpenModel", "Could not close model", -1)
  End if

  iResult = SCApplication.OpenModel(""&Model, oModel)
  if (iResult <> 0) Then
    Call APILog("OpenModel", "Could not open model using varial name", -1)
  End if

End Sub
```

```
'*****************************************************
'*****************************************************
Sub Calculations(bCalculation)

  if(bCalculation = False) Then
    iResult = SCModel.SetCalculation(0)
    if (iResult <> 0) Then
      Call APILog("Calculations", "Could not set calculation: ", -1)
    else
      Call APILog("Calculations", "Disabled calculation", 0)
    End if
    bCalculation = True
  else
    iResult = SCModel.SetCalculation(1)
    if (iResult <> 0) Then
      Call APILog("Calculations", "Could not set calculation: ", -1)
    else
      Call APILog("Calculations", "Enabled calculation", 0)
    End if
    iResult = SCModel.Recalculate(0, "")
    if (iResult <> 0) Then
      Call APILog("Calculations", "Could not recalculate model: ", -1)
    else
      Call APILog("Calculations", "Recalculated model", 0)
    End if
  End if

End Sub
'*********************************************************
'*********************************************************
Sub VarialTest()
  Dim iVarSCId
  Dim iVarMeaId
  Dim PropertyName
  Dim outResult
```

```
Dim Model
Dim CreateModel
Dim outModel
Dim OpenModel
Dim ModelName

CreateModel = "VarialTest Model"
ModelName = "Government"

Call APILog("VarialTest", "Starting to test different varials etc.", 0)

iResult = SCApplication.CreateModel("VarialTest Model", ""&ModelName, outModel)
if (iResult <> 0) Then
  Call APILog("VarialTest", "Could not create model", -1)
End if

'SCApplication.Sleep(10000)

iResult = SCApplication.OpenModel(CreateModel, OpenModel)
if (iResult <> 0) Then
  Call APILog("VarialTest", "Could not open model", -1)
End if

iResult = SCModel.CreateScorecard("VarialTest", "VART1", 0, iVarSCId)
if (iResult <> 0) Then
  Call APILog("VarialTest", "Could not create scorecard", -1)
End if

iResult=SCModel.CreateElement(0, "VarTestMeasure", "VAME1", iVarSCId, 0, iVarMeaId)
If (iResult <> 0) Then
  Call APILog("VarialTest", "Could not create element: ", -1)
End If

iResult = SCModel.SetProperty(""&iVarMeaId, "ACCUMULATIONRULE", 7)
If (iResult <> 0 ) Then
```

```
  Call APILog("VarialTest", "Could not set property 'ACCUMALATIONRULE': ", -1)
End If

PropertyName = "DESCRIPTION"

iResult = SCModel.SetProperty(iVarMeaId, PropertyName, "This is a description")
If (iResult <> 0 ) Then
  Call APILog("VarialTest", "Could not set property 'DESCRIPTION': ", -1)
End If

PropertyName = "MEASUREUNITID"

iResult = SCModel.SetProperty(iVarMeaId, ""&PropertyName, 272)
If (iResult <> 0 ) Then
  Call APILog("VarialTest", "Could not set property 'MEASUREUNITID': ", -1)
End If

iResult = SCModel.SetProperty(iVarMeaId, "PERIODLEVELID", 36)
If (iResult <> 0 ) Then
  Call APILog("VarialTest", "Could not set property 'PERIOIDLEVELID': ", -1)
End If

iResult = SCModel.SetProperty(iVarMeaId, "VALUESETTINGID", 24)
If (iResult <> 0 ) Then
  Call APILog("VarialTest", "Could not set property 'VALUESETTINGID': ", -1)
End If

iResult = SCModel.GetProperty(""&iVarMeaId, "ACCUMULATIONRULE", outResult)
if (iResult <> 0) Then
  Call APILog("VarialTest", "Could not get property 'ACCUMULATIONRULE'", -1)
End if

PropertyName = "DESCRIPTION"

iResult = SCModel.GetProperty(iVarMeaId, PropertyName, outResult)
```

```
  if (iResult <> 0) Then
    Call APILog("VarialTest", "Could not get property 'DESCRIPTION'", -1)
  End if

  PropertyName = "MEASUREUNITID"

  iResult = SCModel.GetProperty(iVarMeaId, ""&PropertyName, outResult)
  if (iResult <> 0) Then
    Call APILog("VarialTest", "Could not get property 'MEASUREUNITID'", -1)
  End if

  iResult = SCModel.DeleteElement(iVarSCId)
  if (iResult <> 0) Then
    Call APILog("VarialTest", "Could not delete scorecard", -1)
  End if

  iResult = SCModel.DeleteModel
  if (iResult <> 0) Then
    Call APILog("VarialTest", "Could not delete model", -1)
  End if

End Sub
'*********************************************************
'*********************************************************
Sub MiscTest()
  Dim oModel
  Dim iSCId
  Dim ElementId
  Dim Properties(4,1)
  Dim ObjectId
  Dim Rights()
  Dim bArchived
  Dim GotProperties()
  Dim Types()
  Dim TypeId
```

```
  Dim Values()
  Dim outFind
  Dim i

  Call APILog("MiscTest", "Starting to do misc tests", 0)

  Call ArrayOfProperties(Properties)

  Call APILog("MiscTest", "ArrayOfProperties called", 0)

  iResult = SCApplication.CreateModel("MiscTestModel", "Government", oModel)
  if (iResult <> 0) Then
    Call APILog("MiscTest", "Could not create new model: ", -1)
  End if

  Call APILog("MiscTest", "Model Created", 0)

'  iResult = SCModel.SetModelProperty("ModelArchived", TRUE)
'  if (iResult <> 0) Then
'    Call APILog("MiscTest", "Could not set model property 'ModelArchived' = TRUE: ", -1)
'  End if

  Call APILog("MiscTest", "Model Archived", 0)

'  iResult = SCModel.SetModelProperty("ModelArchived", FALSE)
'  if (iResult <> 0) Then
'    Call APILog("MiscTest", "Could not set model property 'ModelArchived' = FALSE: ", -1)
'  End if

  Call APILog("MiscTest", "Model unarchived", 0)

  iResult = SCModel.GetModelProperty("ModelArchived", bArchived)
  if (iResult <> 0) Then
    Call APILog("MiscTest", "Coudl not get model property: ", -1)
  End if
```

```
Call APILog("MiscTest", "Got archived status", 0)

iResult = SCModel.GetActive("MODEL", oModel)
if (iResult <> 0) Then
  Call APILog("MiscTest", "Could not get active model: ", -1)
End if

Call APILog("MiscTest", "Got active model", 0)

iResult = SCModel.CreateScorecard("MiscTestScorecard", "MTSC1", 0, iSCId)
if (iResult <> 0) Then
  Call APILog("MiscTest", "Could not create a new scorecard: ", -1)
End if

Call APILog("MiscTest", "Created SC 'MiscTestScorecard'", 0)

'iResult = SCModel.Find("ELEMENTTYPES=SCORECARD;", "(NAME = ""Misc-
TestScorecard"") AND (SYMBOL = ""MTSC1"")", "", outFind)
'if (iResult <> 0) Then
'  Call APILog("MiscTest", "Could not Find Scorecard id: ", -1)
'End if

'if IsEmpty(outFind) Then
'  Call APILog("MiscTest", "Could not FIND Scorecard id: ", -1)
'else
'  Call APILog("MiscTest", "Scorecard Id found. Comparing...", 0)
'  for i = lbound(outFind) to uBound(outFind)
'    if (""&outFind(i) <> ""&iSCId) Then
'      Call APILog("MiscTest", "Scorecard id's does not match: ", -1)
'    else
'      Call APILog("MiscTest", "Found scorecard id matches to created scorecard.", 0)
'    End if
'  Next
'End if
```

```
  iResult = SCModel.CreateElement(0, "MiscTestElement", "MiscMEA1", iSCId, 0, Elemen-
tId)
  if (iResult <> 0) Then
    Call APILog("MiscTest", "Could not create a new element: ", -1)
  End if

  Call APILog("MiscTest", "Created Element 'MiscTestElement'", 0)

  iResult = SCModel.SetProperties(ElementId, Properties)
  if (iResult <> 0) Then
    Call APILog("MiscTest", "Could not set properties for element: ", -1)
  End if

  Call APILog("MiscTest", "Set properties to the element", 0)

  iResult = SCModel.GetProperties(ElementId, GotProperties)
  if (iResult <> 0) Then
    Call APILog("MiscTest", "Could not get properties", -1)
  End if

  Call APILog("MiscTest", "get properties", 0)

  iResult = SCModel.CreateObject("INFORMATIONITEM", "Jep Jep", "", ObjectId)
  if (iResult <> 0) Then
    Call APILog("MiscTest", "Could not create object", -1)
  End if

  Call APILog("MiscTest", "Created information item", 0)

  iResult = SCModel.SetObjectRights(ObjectId, "qpr", 3)
  if (iResult <> 0) Then
    Call APILog("MiscTest", "Could not set object rights: ", -1)
  End if
```

```
Call APILog("MiscTest", "Set rights to information item", 0)

iResult = SCModel.GetObjectRights(ObjectId, Rights)
if (iResult <> 0) Then
  Call APILog("MiscTest", "Could not get object rights", -1)
End if

Call APILog("MiscTest", "Get object rights", 0)

iResult = SCModel.GetTypes(Types)
if (iResult <> 0) Then
  Call APILog("MiscTest", "Could not get types", -1)
End if

Call APILog("MiscTest", "Got types", 0)

iResult = SCModel.GetTypeIdByName("Element", TypeId)
if (iResult <> 0) Then
  Call APILog("MiscTest", "Could not get type id by name: ", -1)
End if

Call APILog("MiscTest", "Got type id by name", 0)

iResult = SCModel.GetValues(0, "", 0, Values)
if (iResult <> 0) Then
  Call APILog("MiscTest", "Could not get values: ", -1)
End if

Call APILog("MiscTest", "Got values", 0)

iResult = SCModel.DeleteModel
if (iResult <> 0) Then
  Call APILog("MiscTest", "Could not delete model: ", -1)
End if
```

```
  Call APILog("MiscTest", "Deleted model", 0)

  SCApplication.Sleep(10000)

End Sub
'*******************************************************
'*******************************************************
Sub CreateSCElements()
  Dim RootSCId
  Dim TopElementId
  Dim iSCId
  Dim ElementRights
  Dim values(6,2)
  Dim nimi
  Dim Array(0,2)
  Dim z
  Dim i
  Dim ReferenceId
  Dim ObjectId
  Dim DeleteObjectId

  i = 0
  z = 0

  call GetValuesFrom(values)

  iResult=SCModel.CreateScorecard("Scorecard " & SCName, "SC"&SCName, 0, iSCId)
  If (iResult <> 0) Then
    Call APILog("CreateSCElements", "Could not create scorecard: ", -1)
  End If

  iResult = SCModel.SetObjectRights(iSCId, "ViewUser", y)
  if (iResult <> 0) Then
    Call APILog("CreateSCElements", "Could not set scorecard rights: ", -1)
  End if
```

```
  iResult = SCModel.CreateObject("INFORMATIONITEM", "This is an information item.",
"", ObjectId)
  if (iResult <> 0) Then
    Call APILog("CCreateSCElements", "Could not create object", -1)
  End if

  iResult = SCModel.AttachObject(ObjectId, iSCId)
  if (iResult <> 0) Then
    Call APILog("CreateSCElements", "Could not attach object to scorecard", -1)
  End if

  do while i < 4
    Dim PerspectiveId

    if (i=0) Then
      ElementRights = "NONE"
    elseif (i=1) Then
      ElementRights = "VIEW"
    elseif (i=2) Then
      ElementRights = "UPDATE"
    else
      ElementRights = "ADMIN"
    End if

    iResult=SCModel.CreateElement(0, "UserRights " &SCName&ElementRights, "Measure"
&SCName &ElementRights, iSCId, 0, PerspectiveId)
    If (iResult <> 0) Then
      Call APILog("CreateSCElements", "Could not create element: ", -1)
    End If

    iResult = SCModel.SetProperty(""&PerspectiveId, "ACCUMULATIONRULE", 7)
    If (iResult <> 0 ) Then
      Call APILog("CreateSCElements", "Could not set property 'ACCUMALATIONRULE':
", -1)
```

```
End If

iResult = SCModel.SetProperty(PerspectiveId, "DESCRIPTION", "This is a description")
If (iResult <> 0 ) Then
  Call APILog("CreateSCElements", "Could not set property 'DESCRIPTION': ", -1)
End If

iResult = SCModel.SetProperty(PerspectiveId, "MEASUREUNITID", 272)
If (iResult <> 0 ) Then
  Call APILog("CreateSCElements", "Could not set property 'MEASUREUNITID': ", -1)
End If

iResult = SCModel.SetProperty(PerspectiveId, "PERIODLEVELID", 36)
If (iResult <> 0 ) Then
  Call APILog("CreateSCElements", "Could not set property 'PERIOIDLEVELID': ", -1)
End If

iResult = SCModel.SetProperty(PerspectiveId, "VALUESETTINGID", 24)
If (iResult <> 0 ) Then
  Call APILog("CreateSCElements", "Could not set property 'VALUESETTINGID': ", -1)
End If

iResult=SCModel.SetValues(PerspectiveId, values)
If (iResult <> 0) Then
  Call APILog("CreateSCElements", "Could not set values for element: ", -1)
End If

'   Dim StatArray1
'   Dim StatArray2(0,2)
'   Dim StatArray3

'   iResult = SCModel.GetStatuses(PerspectiveId, "ACT", -1, StatArray1)
'   if (iResult <> 0) Then
'     Call APILog("CreateSCElements", "Could not get status for series: ", -1)
'   End if
```

```
'    StatArray2(0,0) = "ACT"
'    StatArray2(0,1) = "$"&StatArray1(0,3)
'    StatArray2(0,2) = "Accepted"

'    iResult = SCModel.SetStatuses(PerspectiveId, StatArray2)
'    if (iResult <> 0) Then
'      Call APILog("CreateSCElements", "Could not set status for series: ", -1)
'    End if

'    iResult = SCModel.GetStatuses(PerspectiveId, "ACT", -1, StatArray3)
'    if (iResult <> 0) Then
'      Call APILog("CreateSCElements", "Could not get status for series 2: ", -1)
'    End if

'    if (""&StatArray2(0,2) = ""&StatArray3(0,13)) Then
'      Call APILog("CreateSCElements", "Succesfully set status for series", 0)
'    Else
'      Call APILog("CreateSCElements", "Error when trying to set status for measure", -1)
'    End if

    Call SetMeaFormula(PerspectiveId)

    iResult = SCModel.SetObjectRights(PerspectiveId, "ViewUser", z)
    if (iResult <> 0) Then
      Call APILog("CreateSCElements", "Could not set object rights: ", -1)
    End if

    iResult = SCModel.GetGraph(PerspectiveId, ""&strDirectory &"\trend"&i &z &".jpg",
"GRAPHTYPE=TREND;PERIOD=-1;SERIES=-1")
'    if (iResult <> 0) Then
'      Call APILog("CreateSCElements", "Could not get graph: ", -1)
'    End if
```

```
   iResult = SCModel.GetGraph(PerspectiveId, ""&strDirectory &"\template"&i &z &".jpg",
"WIDTH=800; HEIGHT=400;GRAPHTYPE=CHART;PERIOD=-1;SERIES=-1; TEM-
PLATE=Month Template;")
'    if (iResult <> 0) Then
'      Call APILog("CreateSCElements", "Could not get graph template: ", -1)
'    End if

   'iResult = SCModel.GetGraph(PerspectiveId, ""&strDirectory &"\chart"&i &z&".jpg",
"GRAPHTYPE=CHART;PERIOD=-4;SERIES=-1")
   'if (iResult <> 0) Then
   '  Call APILog("CreateSCElements", "Could not get element chart graph: ", -1)
   'End if

   iResult = SCModel.CreateReferenceElement(PerspectiveId, iSCId, 0, ReferenceId)
   if (iResult <> 0) Then
     Call APILog("CreateSCElements", "Could not create reference element: ", -1)
   End if

   z = z + 1
   i = i + 1
  loop

    Dim StatArray1
   Dim StatArray2(0,2)
   Dim StatArray3

   iResult = SCModel.GetStatuses(PerspectiveId, "ACT", -1, StatArray1)
   if (iResult <> 0) Then
     Call APILog("CreateSCElements", "Could not get status for series: ", -1)
   End if

   StatArray2(0,0) = "ACT"
   StatArray2(0,1) = "$"&StatArray1(0,3)
   StatArray2(0,2) = "Accepted"
```

```
  iResult = SCModel.SetStatuses(PerspectiveId, StatArray2)
  if (iResult <> 0) Then
    Call APILog("CreateSCElements", "Could not set status for series: ", -1)
  End if

  iResult = SCModel.GetStatuses(PerspectiveId, "ACT", -1, StatArray3)
  if (iResult <> 0) Then
    Call APILog("CreateSCElements", "Could not get status for series 2: ", -1)
  End if

  if (""&StatArray2(0,2) = ""&StatArray3(0,13)) Then
    Call APILog("CreateSCElements", "Succesfully set status for series", 0)
  Else
    Call APILog("CreateSCElements", "Error when trying to set status for measure", -1)
  End if

 iResult = SCModel.GetObjectIdByName("INFORMATIONITEM", "This is an informa-
tion item.", DeleteObjectId)
 if (iResult <> 0) Then
   Call APILog("CreateSCElements", "Could not get object id by name: ", -1)
 End if

 iResult = SCModel.DetachObject(DeleteObjectId, iSCId)
 if (iResult <> 0) Then
   Call APILog("CreateSCElements", "Could not detach object to scorecard", -1)
 End if

 iResult = SCModel.DeleteObject(DeleteObjectId)
 if (iResult <> 0) Then
   Call APILog("CreateSCElements", "Could not delete object", -1)
 End if

 y = y + 1

 Call APILog("CreateSCElements", "Created: Scorecard "& SCName, 0)
```

```
End Sub
'*****************************************
'*****************************************
Sub Modify()
  Dim RootSCId
  Dim TopElementId
  Dim iSCId
  Dim ElementRights
  Dim nimi
  Dim z
  Dim i
  Dim name
  Dim values2(6,2)
  Dim iCopyId
  Dim iActiveId
  Dim oModel
  Dim pass
  Dim fail

  pass = 0
  fail = -1

  i = 0
  z = 0

  Call APILog("Modify", "Starting to modify objects", 0)

  call GetValuesFrom2(values2)

  do while i < 4
    if (i=0) Then
      SCName = "NONE"
    elseif (i=1) Then
      SCName = "VIEW"
    elseif (i=2) Then
```

```
    SCName = "UPDATE"
   elseif (i=3) Then
    SCName = "ADMIN"
   End if
   do while z < 4
    if (z=0) Then
     ElementRights = "NONE"
    elseif (z=1) Then
     ElementRights = "VIEW"
    elseif (z=2) Then
     ElementRights = "UPDATE"
    elseif (z=3) Then
     ElementRights = "ADMIN"
    End if

    if (ElementRights = "NONE") Then
     pass = - 1
     fail = 0
    Else
     pass = 0
     fail = -1
    End if

    Call APILog("Modify", "Scorecard: "&SCName &ElementRights, 0)

    iResult = SCModel.GetElementId("SC"&SCName, "Meas-
ure"&SCName&ElementRights, iSCId)
    if (iResult <> 0) Then
     Call APILog("Modify", "Could not get element id: ", fail)
    else
     Call APILog("Modify", "Get element id", pass)
    End if

    iResult = SCModel.GetGraph(iSCId, ""&strDirectory &"trend"&i &z &".jpg",
"GRAPHTYPE=TREND;PERIOD=-1;SERIES=-1")
```

```
'     if (iResult <> 0) Then
'       Call APILog("Modify", "Could not get graph: ", pass)
'     else
'       Call APILog("Modify", "User got graph from element", fail)
'     End if

      iResult = SCModel.SetObjectRights(iSCId, ""&ViewUser, 3)
      if (iResult <> 0) Then
       if(ViewUser = "ViewUser") Then
         fail = 0
       End if
       Call APILog("Modify", "Could not set object rights: ", fail)
      else
       if(ViewUser = "ViewUser") Then
         pass = -1
       End if
       Call APILog("Modify", "Set object rights", pass)
      End if

      iResult = SCModel.RemoveObjectRights(iSCId, ""&stUser)
      if (iResult <> 0) Then
       if(ViewUser = "ViewUser") Then
         fail = 0
       End if
       Call APILog("Modify", "Could not remove object rights: ", fail)
      else
       if(ViewUser = "ViewUser") Then
         pass = -1
       End if
       Call APILog("Modify", "Removed object rights", pass)
      End if

      iResult = SCModel.GetProperty(iSCId, "NAME", name)
      if (iResult <> 0) Then
```

```
      if(ElementRights = "VIEW" OR ElementRights = "UPDATE" OR ElementRights =
"ADMIN") Then
        fail = -1
      else
        fail = 0
      End if
      Call APILog("Modify", "Could not get element name: ", fail)
    else
      if(ElementRights = "VIEW" OR ElementRights = "UPDATE" OR ElementRights =
"ADMIN") Then
        pass = 0
      else
        pass = -1
      End if
      Call APILog("Modify", "Get property 'NAME'", pass)
    End if

    iResult = SCModel.SetProperty(iSCId, "NAME", "Changed "&name)
    if (iResult <> 0) Then
      if(ElementRights = "UPDATE" OR ElementRights = "ADMIN") Then
        pass = 0
        fail = -1
      else
        pass = -1
        fail = 0
      End if
      Call APILog("Modify", "Could not set Element name: ", fail)
    else
      Call APILog("Modify", "Set new Element name", pass)
    End if

    iResult = SCModel.SetProperty(iSCId, "ACCUMULATIONRULE", 4)
    If (iResult <> 0 ) Then
      if(ElementRights = "UPDATE" OR ElementRights = "ADMIN") Then
        pass = 0
```

```
    fail = -1
  else
    pass = -1
    fail = 0
  End if
  Call APILog("Modify", "Could not set property 'ACCUMALATIONRULE': ", fail)
else
  Call APILog("Modify", "Set property 'ACCUMALATIONRULE'", pass)
End If

iResult = SCModel.SetProperty(iSCId, "DESCRIPTION", "CHANGED!!!!!!")
If (iResult <> 0 ) Then
  if(ElementRights = "UPDATE" OR ElementRights = "ADMIN") Then
    pass = 0
    fail = -1
  else
    pass = -1
    fail = 0
  End if
  Call APILog("Modify", "Could not set property 'DESCRIPTION': ", fail)
else
  Call APILog("Modify", "Set property 'DESCRIPTION'", pass)
End If

iResult = SCModel.SetProperty(iSCId, "MEASUREUNITID", 102)
If (iResult <> 0 ) Then
  if(ElementRights = "UPDATE" OR ElementRights = "ADMIN") Then
    pass = 0
    fail = -1
  else
    pass = -1
    fail = 0
  End if
  Call APILog("Modify", "Could not set property 'MEASUREUNITID': ", fail)
else
```

```
  Call APILog("Modify", "Set property 'MEASUREUNITID'", pass)
End If

iResult = SCModel.SetProperty(iSCId, "PERIODLEVELID", 35)
If (iResult <> 0 ) Then
  if(ElementRights = "UPDATE" OR ElementRights = "ADMIN") Then
    pass = 0
    fail = -1
  elseif(ElementRights = "VIEW" OR ElementRights = "NONE") Then
    pass = -1
    fail = 0
  End if
  Call APILog("Modify", "Could not set property 'PERIOIDLEVELID': ", fail)
else
  Call APILog("Modify", "Set property 'PERIODLEVELID'", pass)
End If

iResult = SCModel.SetProperty(iSCId, "VALUESETTINGID", 24)
If (iResult <> 0 ) Then
  if(ElementRights = "UPDATE" OR ElementRights = "ADMIN") Then
    pass = 0
    fail = -1
  else
    pass = -1
    fail = 0
  End if
  Call APILog("Modify", "Could not set property 'VALUESETTINGID': ", fail)
else
  Call APILog("Modify", "Set property 'VALUESETTINGID'", pass)
End If

iResult=SCModel.SetValues(iSCId, values2)
If (iResult <> 0) Then
  Call APILog("Modify", "Could not set values for element: ", fail)
else
```

```
      Call APILog("Modify", "Set new values for element", pass)
    End If

    Call SetMeaFormula2(iSCId, ElementRights)

    z = z + 1
  loop
  i = i + 1
  z = 0
loop

Dim TestSC
dim o
Dim qCount

qCount = 0

iResult = SCModel.GetScorecards(TestSC)
if (iResult <> 0) Then
  Call APILog("Modify", "Could not get scorecards: ", -1)
else
  for o = lbound(TestSC, 1) to uBound(TestSC ,1)
    Dim outFindList
    Dim q
    Dim outName123

    iResult = SCMo-
del.Find("SCORECARDIDS="&TestSC(o,1)&";ELEMENTTYPES=ELEMENT", "", "",
outFindList)
    if (iResult <> 0) Then
      Call APILog("Modify", "Could not find any elements or scorecards: " -1)
    else
      for q = lbound(outFindList) to uBound(outFindList)
        qCount = qCount + 1
      Next
```

```
    End if

    if (qCount = 7) Then
      Call APILog("Modify", "Correct amount of elements found", 0)
    elseif(qCount < 7) Then
      Call APILog("Modify", "User did not find enough scorecards/elements. Minor user
rights issue", -1)
    else
      Call APILog("Modify", "User found too many scorecards/elements. MAJOR ISSUE!!!!",
-1)
    End if

    qCount = 0

  Next
 End if

 iResult = SCModel.ExportModel("c:\Export_"&SCName &".smf")
 if (iResult <> 0) Then
  Call APILog("Modify", "Could not export model: ", fail)
 else
  Call APILog("Modify", "Exported model", pass)
 End if

 iResult = SCModel.SaveAs(""&Viewuser &"_newModel")
 if (iResult <> 0) Then
  Call APILog("Modify", "Could not save model to server: ", fail)
 else
  Call APILog("Modify", "Saved model as server model", pass)
 End if

 iResult = SCModel.Recalculate(0,"")
 if (iResult <> 0) Then
  Call APILog("Modify", "Could not recalculate model: ", fail)
 else
```

```
  Call APILog("Modify", "Recalculated model", pass)
 End if

End Sub
'*****************************************
'*****************************************
Sub GetScorecards(iSCNoneId, iSCViewId, iSCUpdateId, iSCAdminId)
 Dim Scorecards
 Dim i

 Call APILog("GetScorecards", "Getting scorecard id's", 0)

 iResult = SCModel.GetScorecards(Scorecards)
 if (iResult <> 0) Then
  Call APILog("GetScorecards", "Could not get scorecards", -1)
 else
  Call APILog("GetScorecards", "Got scorecards", 0)
 End if

 if (IsEmpty(Scorecards)=True) Then

 Else
  for i = lbound(Scorecards, 1) to ubound(Scorecards, 1)
   if (Scorecards(i,0) = "Scorecard NONE") Then
    iSCNoneId = Scorecards (i,1)
   elseif (Scorecards(i,0) = "Scorecard VIEW") Then
    iSCViewId = Scorecards (i,1)
   elseif (Scorecards(i,0) = "Scorecard UPDATE") Then
    iSCUpdateId = Scorecards (i,1)
   elseif (Scorecards (i,0) = "Scorecard ADMIN") Then
    iSCAdminId = Scorecards (i,1)
   End if
  next
 End if
End Sub
```

```
'*******************************************
'*******************************************
Sub CopyScorecards()
  Dim x
  Dim iCopyId
  Dim NewId
  Dim iSCNoneId
  DIM iSCViewId
  Dim iSCUpdateId
  Dim iSCAdminId

  Call APILog("CopyScorecards", "Here it comes", 0)

  Call GetScorecards(iSCNoneId, iSCViewId, iSCUpdateId, iSCAdminId)

  for x = 0 to 3
    if (x = 0) Then
      iCopyId = iSCNoneId
    elseif (x = 1) Then
      iCopyId = iSCViewId
    elseif (x = 2) Then
      iCopyId = iSCUpdateId
    else
      iCopyId = iSCAdminId
    End if

    iResult = SCModel.CopyScorecard(iCopyId, iCopyId, NewId)
    if (iResult <> 0) Then
      Call APILog("CopyScorecards", "Could not copy scorecard: ", 0)
    else
      Call APILog("CopyScorecards", "Copied scorecards", -1)
    End if
  next
End Sub
'*******************************************
```

```
'*****************************************
Sub DeleteScorecards(user)
  Dim iSCNoneId
  DIM iSCViewId
  Dim iSCUpdateId
  Dim iSCAdminId
  Dim iDeleteId
  Dim x
  Dim pass
  Dim fail

  Call APILog("DeleteScorecards", "Here it comes", 0)

  Call GetScorecards(iSCNoneId, iSCViewId, iSCUpdateId, iSCAdminId)

  for x = 0 to 3
   if (x = 0) Then
    iDeleteId = iSCNoneId
   elseif (x = 1) Then
    iDeleteId = iSCViewId
   elseif (x = 2) Then
    iDeleteId = iSCUpdateId
   else
    iDeleteId = iSCAdminId
   End if

   iResult = SCModel.DeleteElement(iDeleteId)
   if (iResult <> 0) Then
    if (user = "qpr")Then
     fail = -1
    else
     fail = 0
    End if
    Call APILog("DeleteScorecards", "Could not delete scorecard: ", fail)
   else
```

```
    if (user = "qpr")Then
      pass = 0
    else
      pass = -1
    End if
    Call APILog("DeleteScorecards", "Deleted scorecard", pass)
  End if
 next
End Sub
'*******************************************
'*******************************************
Sub DeleteModel()
 Dim oModel
 Dim pass
 Dim fail

 iResult = SCApplication.OpenModel(""&Viewuser &"_newModel", oModel)
 if (iResult <> 0) Then
  if (SCApplication.GetErrorMessage(iResult) = "") Then
   fail = -1
  else
   fail = 0
  End if
  Call APILog("Modify", "Could not open model: ", fail)
 else
  Call APILog("Modify", "Opened model: "&Viewuser &"_newModel", 0)
 End if

 iResult = SCModel.DeleteModel
 if (iResult <> 0) Then
  if (SCApplication.GetErrorMessage(iResult) = "") Then
   fail = -1
  else
   fail = 0
  End if
```

```
    Call APILog("Modify", "Could not delete model: ", fail)
  else
    Call APILog("Modify", "Deleted model", pass)
  End if
End Sub
'*********************************************
'*********************************************
Sub CreateModel()
  Dim oModel

  Call APILog("External - CreateModel", "Creating a new model", 0)

  iResult = SCApplication.Createmodel("SetGetPropertyTest", "Government", oModel)
  if (iResult <> 0) Then
    Call APILog("External - CreateModel", "Could not create a new model: ", -1)
  End if
End Sub
'*************************************************
'*************************************************
Sub CreateSCElement()
  Dim AlertElementId
  Dim AlertSeriesId
  Dim ChildElementId
  Dim ChildElementId2

  Call APILog("External - CreateSCElements", "Creating new scorecards", 0)

  iResult = SCModel.CreateScorecard("PropertyTest", "PTSC", 0, iSCId)
  if (iResult <> 0) Then
    Call APILog("External - CreateSCElements", "Could not create a new scorecard: ", -1)
  End if

  iResult = SCModel.CreateScorecard("ChildPropertyTest", "CPTSC", 0, iChildSCId)
  if (iResult <> 0) Then
    Call APILog("External - CreateSCElements", "Could not create a new scorecard: ", -1)
```

```
  End if

  iResult = SCModel.GetProperty(iChildSCId, "TOPELEMENTID", TopElementId)
  if (iResult <> 0) Then
    Call APILog("External - CreateSCElements", "Could not get Top element id: ", -1)
  End if

  iResult = SCModel.CreateScorecard("PropertiesTest", "PTsSC", 0, PropSCId)
  if (iResult <> 0) Then
    Call APILog("External - CreateSCElements", "Could not create a new scorecard: ", -1)
  End if

  Call APILog("External - CreateSCElements", "Creating a new element", 0)

  iResult = SCModel.CreateElement(0, "PropertyTestElement", "PTELE", iSCId, 0, Elemen-
tId)
  if (iResult <> 0) Then
    Call APILog("External - CreateSCElements", "Could not create a new element: ", -1)
  End if

  iResult = SCModel.CreateElement(0, "ChildElementTest", "CET", iSCId, ElementId, Chil-
dElementId)
  if (iResult <> 0) Then
    Call APILog("CreateSCElements", "Could not create child element: ", -1)
  End if

  iResult = SCModel.CreateElement(0, "ChildElementTest 2", "CET2", iSCId, ElementId,
ChildElementId2)
  if (iResult <> 0) Then
    Call APILog("CreateSCElements", "Could not create child element: ", -1)
  End if

  iResult = SCModel.CreateElement(0, "PropertiesTestElement", "PTsELE", iSCId, 0, PropE-
lemId)
  if (iResult <> 0) Then
```

```
  Call APILog("External - CreateSCElements", "Could not create a new element: ", -1)
 End if

 Call APILog("External - CreateSCElements", "Creating an information item", 0)

 iResult = SCModel.CreateObject("INFORMATIONITEM", "InfoItemOriginal", "", InfoI-
temId)
 if (iResult <> 0) Then
  Call APILog("External - CreateSCElements", "Could not create information item: ", -1)
 End if

 iResult = SCModel.AttachObject(InfoItemId, ElementId)
 if (iResult <> 0) Then
  Call APILog("External - CreateSCElements", "Could not attach object to element: ", -1)
 End if

 iResult = SCModel.GetElementId("SC669", "MEA671", AlertElementId)
 if (iResult <> 0) Then
  Call APILog("External - CreateSCElements", "Could not get element Id: ", -1)
 End if

 iResult = SCModel.CreateObject(-18, "", "ELEMEN-
TID="&AlertElementId&";SERIES=Act;", AlertId)
 if (iResult <> 0) Then
  Call APILog("External - CreateSCElements", "Could not create object 'Alert': ", -1)
 End if
End Sub
'*************************************************
'*************************************************
Sub Scorecard()
 Dim values(6,1)
 Dim Inheritance(15,1)
 Dim i
 Dim pass
 Dim outArray(15,1)
```

```
Dim x
Dim sCase
Dim PropertiesArray(5,1)
Dim outValue
Dim PropertyArray(18,1)

pass = True
sCase = "Scorecard"

Call InheritanceOptions(Inheritance)
Call ScorecardValues(values, Inheritance)

Call SettingGetting(iSCId, values, PropertiesArray, pass, sCase, 2)

for i = 3 to 4
  iResult = SCModel.SetProperty(iChildSCId, values(i,0), values(i,1))
  if (iResult <> 0) Then
    Call APILog("External - Scorecard", "Could not set property: ", -1)
  End if

  iResult = SCModel.GetProperty(iChildSCId, values(i,0), outValue)
  if (iResult <> 0) Then
    Call APILog("External - Scorecard", "Could not get property: ", -1)
  End if

  PropertiesArray(i,0) = values(i,0)

  Call Compare(outValue, values(i,0), values(i,1), pass, sCase)
next

iResult = SCModel.SetProperty(iChildSCId, "INHERITANCEOPTIONS", Inheritance)
if (iResult <> 0) Then
  Call APILog("External - Scorecard", "Could not set property: ", -1)
End if
```

```
  iResult = SCModel.GetProperty(iChildSCId, "INHERITANCEOPTIONS", outArray)
  if (iResult <> 0) Then
    Call APILog("External - Scorecard", "Could not get property", -1)
  End if

  PropertiesArray(5,1) = "InheritanceOptions"

'  for x = 0 to 15
'    if(outArray(x,0) = Inheritance(x,0) AND outArray(x,1) = TRUE) Then
'    else
'      pass = False
'      Call APILog("External - Scorecard", "Property: "&values(5,0) &", Value:
"&Inheritance(x,0)&":"&Inheritance(x,1), -1)
'      Call APILog("External - Scorecard", "Property: "&values(5,0) &", Value:
"&outArray(x,0)&":"&outArray(x,1), -1)
'    End if
'  next

  Call Result(sCase, pass)

  values(1,1) = "PRSC"
  pass = True

  Call Compareproperties(PropSCId, values, PropertiesArray, sCase, pass, 4)

  Call AllScorecard(PropertyArray)

  Call GetPropertiesEX(iChildSCId, PropertyArray, sCase, pass)

  Call BuildArrayAndCompare(PropSCId, PropertyArray, 16, sCase, pass)

  Call Result("Scorecard/Properties", pass)
End Sub
'************************************************
'************************************************
```

```
Sub Element()
  Dim values(13,1)
  Dim i
  Dim pass
  Dim GetElemSeries(4,5)
  Dim SetElemSeries(4,5)
  Dim GetDrillDown
  Dim SetDrillDown
  Dim sCase
  Dim PropertiesArray(10,1)
  Dim outProperties()
  Dim PropertyArray(28,1)
  Dim x
  Dim ChildIds
  Dim ChildId
  Dim z
  Dim ParentId

  pass = True
  sCase = "Element"

  iResult = SCModel.GetProperty(ElementId, "CHILDMEASUREIDS", ChildIds)
  if (iResult <> 0) Then
    Call APILog(""&sCase, "Could not get CHILDMEASUREIDS: ", -1)
    pass = FALSE
  End if

  for z = lbound(ChildIds, 1) to uBound(ChildIds, 1)
    ChildId = ""&ChildIds(z)

    iResult = SCModel.GetProperty(ChildId, "PARENTELEMENTID", ParentId)
    if (iResult <> 0) Then
      Call APILog(""&sCase, "Could not get 'PARENTELEMENTID': ", -1)
      pass = FALSE
    end if
```

```
  if(""&ParentId = ""&ElementId) Then
    Call APILog("External - "&sCase, "Child-parent connection was correct:", 0)
  else
    Call APILog(""&sCase, "Child-parent connection was NOT correct:", -1)
  End if
next

Call ElementValues(values)

Call SettingGetting(ElementId, values, PropertiesArray, pass, sCase, 10)

values(0,1) = "JARGON"
values(1,1) = "ZARGON"

iResult = SCModel.GetProperty(ElementId, "ELEMENTSERIES", GetElemSeries)
if(iResult <> 0) Then
  Call APILog("External - "&sCase, "Could not get ElementSeries: ", -1)
End if

for i = 0 to 4
  for x = 0 to 5
    SetElemSeries(i,x) = GetElemSeries(i,x)
  Next
Next

SetElemSeries(0,1) = "LABEL"
SetElemSeries(0,4) = 1
SetElemSeries(0,5) = "1+2+3+4"

values(11,1) = SetElemSeries

iResult = SCModel.SetProperty(ElementId, "ELEMENTSERIES", SetElemSeries)
if (iResult <> 0) Then
  Call APILog("External - "&sCase, "Could not set ElementSeries: ", -1)
```

```
End if

iResult = SCModel.GetProperty(ElementId, "ELEMENTSERIES", GetElemSeries)
if(iResult <> 0) Then
  Call APILog("External - "&sCase, "Could not get ElementSeries: ", -1)
End if

for i = 0 to 4
  for x = 0 to 5
    if(SetElemSeries(i,x) = GetElemSeries(i,x)) Then
    Else
      Call APILog("External - "&sCase, "SetProperty Elementseries did not work: ", -1)
    End if
  Next
Next

iResult = SCModel.GetProperty(ElementId, "DrillDownOptions", GetDrillDown)
if (iResult <> 0) Then
  Call APILog("External - "&sCase, "Could not get DrillDownOptions", -1)
End if

SetDrillDown = GetDrillDown

SetDrillDown(0,1) = False
SetDrillDown(3,1) = True

values(12,1) = SetDrillDown

iResult = SCModel.SetProperty(ElementId, "DrillDownOptions", SetDrillDown)
if (iResult <> 0) Then
  Call APILog("External - "&sCase, "Could not set DrillDownOptions", -1)
End if

iResult = SCModel.RefreshDrillDown(0)
if (iResult <> 0) Then
```

```
   Call APILog("External - "&sCase, "Could not refresh Drill Down for the whole model: ", -
1)
  End if

  iResult = SCModel.GetProperty(ElementId, "DrillDownOptions", GetDrillDown)
  if (iResult <> 0) Then
   Call APILog("External - "&sCase, "Could not get DrillDownOptions", -1)
  End if

  for i = lbound(GetDrillDown, 1) to uBound(GetDrillDown, 1)
   if (""&GetDrillDown(i,1) = ""&SetDrillDown(i,1)) Then
   else
    Call APILog("External - "&sCase, ""&GetDrillDown(i,0)&": "&GetDrillDown(i,1), -1)
    Call APILog("External - "&sCase, ""&SetDrillDown(i,0)&": "&SetDrillDown(i,1), -1)
    pass = False
   End if
  Next

  Call Compareproperties(ElementId, values, PropertiesArray, sCase, pass, 10)

  Call AllMeasure(PropertyArray)

  'Call GetPropertiesEX(ElementId, PropertyArray, sCase, pass)

  Call BuildArrayAndCompare(ElementId, PropertyArray, 16, sCase, pass)

  Call Result("Element/Properties", pass)
End Sub
'************************************************
'************************************************
Sub PeriodLevel()
  Dim iPeriodLevelId
  Dim values(5,1)
  Dim i
  Dim pass
```

```
  Dim sCase
  Dim PropertiesArray(5,1)
  Dim PropertyArray(14,1)

  pass = True
  sCase = "PeriodLevel"

  Call GetPerLevProp(values)

  iResult = SCModel.GetProperty(ElementId, "PERIODLEVELID", iPeriodLevelId)
  if (iResult <> 0) Then
    Call APILog("External - PeriodLevelId", "Could not get property 'PERIODLEVELID': ",
-1)
  End if

  Call SettingGetting(iPeriodLevelId, values, PropertiesArray, pass, sCase, 5)

  pass = True

  values(0,1) = "PeriodLevel"
  values(1,1) = "NEW"

  Call Compareproperties(iPeriodLevelId, values, PropertiesArray, sCase, pass, 5)

  Call AllPeriodLevel(PropertyArray)

  Call GetPropertiesEX(iPeriodLevelId, PropertyArray, sCase, pass)

  Call BuildArrayAndCompare(iPeriodLevelId, PropertyArray, 13, sCase, pass)

  Call Result("Element/Properties", pass)
End Sub
'*************************************************
'*************************************************
Sub Range()
```

```
  Dim outElement()
  Dim values(3,1)
  Dim pass
  Dim sCase
  Dim PropertiesArray(3,1)
  Dim PropertyArray(11,1)

  pass = True
  sCase = "Range"

  Call GettingElements("RANGE", outElement, sCase)

  RangeId = outElement(1,1)

  Call RangeValues(values)

  Call SettingGetting(RangeId, values, PropertiesArray, pass, sCase, 3)

  pass = True

  values(0,1) = "NewName"
  values(1,1) = "NeNa"

  Call Compareproperties(RangeId, values, PropertiesArray, sCase, pass, 3)

  Call AllRange(PropertyArray)

  Call GetPropertiesEX(RangeId, PropertyArray, sCase, pass)

  Call BuildArrayAndCompare(RangeId, PropertyArray, 11, sCase, pass)

  Call Result("Range/Properties", pass)
End Sub
'*************************************************
'*************************************************
```

```
Sub ChartTemplate()
  Dim outElement()
  Dim ChartId
  Dim values(3,1)
  Dim pass
  Dim sCase
  Dim PropertiesArray(3,1)
  Dim PropertyArray(10,1)

  pass = True
  sCase = "ChartTemplate"

  Call GettingElements("CHARTTEMPLATE", outElement, sCase)

  ChartId = outElement(1,1)

  Call ChartValues(values)

  Call SettingGetting(ChartId, values, PropertiesArray, pass, sCase, 3)

  pass = True

  values(0,1) = "NewChart"
  values(1,1) = "NC"

  Call Compareproperties(ChartId, values, PropertiesArray, sCase, pass, 3)

  Call AllChartTemplate(PropertyArray)

  Call GetPropertiesEX(ChartId, PropertyArray, sCase, pass)

  Call BuildArrayAndCompare(ChartId, PropertyArray, 10, sCase, pass)

  Call Result("Chart/Properties", pass)
End Sub
```

```
'************************************************
'************************************************
Sub Series()
  Dim outElement()
  Dim values(5,1)
  Dim pass
  Dim sCase
  Dim PropertiesArray(5,1)
  Dim PropertyArray(14,1)

  pass = True
  sCase = "Series"

  Call SeriesValues(values)

  Call GettingElements("SERIES", outElement, sCase)

  SeriesId = outElement(1,1)

  Call SettingGetting(SeriesId, values, PropertiesArray, pass, sCase, 5)

  pass = true

  values(0,1) = "NewNAME"
  values(1,1) = "NeNA"

  Call Compareproperties(SeriesId, values, PropertiesArray, sCase, pass, 5)

  Call AllSeries(PropertyArray)

  Call GetPropertiesEX(SeriesId, PropertyArray, sCase, pass)

  Call BuildArrayAndCompare(SeriesId, PropertyArray, 14, scase, pass)

  Call Result("Series/Properties", pass)
```

```
End Sub
'************************************************
'************************************************
Sub ValueSetting()
  Dim outElement()
  Dim valueSettingId
  Dim values(5,1)
  Dim i
  Dim pass
  Dim sCase
  Dim PropertiesArray(5,1)
  Dim PropertyArray(15,1)

  pass = True
  sCase = "ValueSetting"

  Call ValueSettingvalues(values)

  Call GettingElements("VALUESETTING", outElement, sCase)

  valueSettingId = outElement(5,1)

  Call SettingGetting(ValueSettingId, values, PropertiesArray, pass, sCase, 5)

  pass = True

  values(0,1) = "ValSet"
  values(1,1) = "VS"

  Call Compareproperties(ValueSettingId, values, PropertiesArray, sCase, pass, 5)

  Call AllValueSetting(PropertyArray)

  Call GetPropertiesEX(ValueSettingId, PropertyArray, sCase, pass)
```

```
  Call BuildArrayAndCompare(ValueSettingId, PropertyArray, 13, sCase, pass)

  Call Result("ValueSetting/Properties", pass)
End Sub
'*************************************************
'*************************************************
Sub InformationItem()
  Dim sCase
  Dim pass
  Dim values(3,1)
  Dim PropertiesArray(3,1)
  Dim outProperties()
  Dim PropertyArray(11,1)

  sCase = "InformationItem"
  pass = True

  Call InfoItemValues(values)

  Call SettingGetting(InfoItemId, values, PropertiesArray, pass, sCase, 3)

  pass = True

  values(0,1) = "NewInfo"
  values(1,1) = "NeIn"

  Call Compareproperties(InfoItemId, values, PropertiesArray, sCase, pass, 3)

  Call AllInfoItem(PropertyArray)

  Call GetPropertiesEX(InfoItemId, PropertyArray, sCase, pass)

  Call BuildArrayAndCompare(InfoItemId, PropertyArray, 11, scase, pass)

  Call Result("InformationItem/Properties", pass)
```

```
End Sub
'************************************************
'************************************************
Sub MeasureUnit()
  Dim MeaUnitId
  Dim outElement()
  Dim sCase
  Dim pass
  Dim values(8,1)
  Dim PropertiesArray(8,1)
  Dim PropertyArray(16,1)

  pass = True
  sCase = "MeasureUnit"

  Call GettingElements("UNIT", outElement, sCase)

  MeaUnitId = outElement(1,1)

  Call MeasureUnitValues(values)

  Call SettingGetting(MeaUnitId, values, PropertiesArray, pass, sCase, 8)

  pass = True

  values(0,1) = "NewMea"
  values(1,1) = "NEME"

  Call Compareproperties(MeaUnitId, values, PropertiesArray, sCase, pass, 8)

  Call AllMeasureUnit(PropertyArray)

  Call GetPropertiesEX(MeaUnitId, PropertyArray, sCase, pass)

  Call BuildArrayAndCompare(MeaUnitId, PropertyArray, 16, scase, pass)
```

```
  Call Result("MeasureUnit/Properties", pass)
End Sub
'*************************************************
'*************************************************
Sub Alert()
  Dim outElement()
  Dim sCase
  Dim pass
  Dim values(5,1)
  'Dim AlertId
  Dim PropertiesArray(5,1)
  Dim PropertyArray(8,1)

  sCase = "Alert"
  pass = True

  'Call GettingElements("ALERT", outElement, sCase)

  'AlertId = outElement(0,1)

  iResult = SCModel.GetProperty(AlertId, "Recipients", RecArray)
  if (iResult <> 0) Then
    Call APILog("External - "&sCase, "Could not get property 'Recipients': ", -1)
  End if

  Call AlertValues(values)

  Call SettingGetting(AlertId, values, PropertiesArray, pass, sCase, 4)

  pass = True

  Call CompareProperties(AlertId, values, PropertiesArray, sCase, pass, 4)

  Call AllAlert(PropertyArray)
```

```
  Call GetPropertiesEX(AlertId, PropertyArray, sCase, pass)

  Call BuildArrayAndCompare(AlertId, PropertyArray, 7, sCase, pass)

  Call Result("Alert/Properties", pass)

End Sub
'************************************************
'************************************************
Sub LinkedData()
  Dim outElement()
  Dim sCase
  Dim pass
  Dim values(2,1)
  Dim LinkedDataId
  Dim PropertiesArray(5,1)
  Dim PropertyArray(11,1)

  sCase = "LinkedData"
  pass = True

  Call GettingElements("LINKEDDATA", outElement, sCase)

  if (IsEmpty(outElement)=TRUE) Then
    Call APILog("LinkedData", "Could not find linked data", -1)
    pass = False
    Call Result("LinkedDataId/Properties", pass)
  Else

    LinkedDataId = outElement(0,1)

    Call LinkedDataValues(values)

    Call SettingGetting(LinkedDataId, values, PropertiesArray, pass, sCase, 2)
```

```
    pass = True

    values(0,1) = "NewName"
    values(1,1) = "New Description"

    Call Compareproperties(LinkedDataId, values, PropertiesArray, sCase, pass, 2)

    Call AllLinkedData(PropertyArray)

    Call GetPropertiesEX(LinkedDataId, PropertyArray, sCase, pass)

    Call BuildArrayAndCompare(LinkedDataId, PropertyArray, 11, scase, pass)

    Call Result("LinkedDataId/Properties", pass)
  End if
End Sub
'*************************************************
'*************************************************
Sub SettingGetting(Id, values, PropertiesArray, pass, sCase, number)
  Dim i
  Dim outValue

  for i = 0 to number
    iResult = SCModel.SetProperty(Id, values(i,0), values(i,1))
    if (iResult <> 0) Then
      Call APILog("External - "&sCase, "Could not set property "&i&": ", -1)
    End if

    iResult = SCModel.GetProperty(Id, values(i,0), outValue)
    if (iResult <> 0) Then
      Call APILog("External - "&sCase, "Could not get property: ", -1)
    End if

    Call Compare(outValue, values(i,0), values(i,1), pass, sCase)
```

```
  PropertiesArray(i,0) = values(i,0)
 next

 Call Result(sCase, pass)
End Sub
'************************************************
'************************************************
Sub Compareproperties(Id, values, PropertiesArray, sCase, pass, number)
 Dim outValue
 Dim i
 Dim outArray
 Dim Help
 Dim x
 Dim Help2
 Dim outRoles(0,2)

 pass = True

 iResult = SCModel.Setproperties(Id, values)
 if (iResult <> 0) Then
  Call APILog("External - "&sCase, "Could not set properties: ", -1)
 End if

 for i = 0 to number
  iResult = SCModel.Getproperty(Id, PropertiesArray(i,0), outValue)
  if (iResult <> 0) Then
   Call APILog("External - "&sCase, "Could not get property: ", -1)
  End if

  PropertiesArray(i,1) = outValue
 next

 for i = 0 to number
  if(""&PropertiesArray(i,1) = ""&values(i,1)) Then
```

```
    else
      pass = False
      Call APILog("External - "&sCase, "Property: "&values(i,0) &", Value: "&values(i,1), -1)
      Call APILog("External - "&sCase, "Property: "&PropertiesArray(i,0) &", Value:
"&PropertiesArray(i,1), -1)
    End if
  next

  if(sCase = "Scorecard") Then
    iResult = SCModel.GetProperty(id, "InheritanceOptions", outArray)
    if (iResult <> 0) Then
      Call APILog("External - "&sCase, "Could not get InheritanceOptions from scorecard: ", -
1)
    End if

    Help = values(5,1)

    for i = 0 to 15
      if(outArray(i,0) = Help(i,0) AND outArray(i,1)=True) Then
      else
        pass = False
        Call APILog("External - Scorecard", "Comparing InheritanceOptions did not work, or
there were some other errors: ", -1)
      End if
    next

    iResult = SCModel.GetProperty(Id, "Owner", outValue)

    if (""&outValue <> "1") Then
      Call APILog("External - Scorecard", "Owner did not work: ", -1)
    End if
  End if

  if(sCase = "Element") Then
    iResult = SCModel.GetProperty(Id, "ElementSeries", outArray)
```

```
if (iResult <> 0) Then
  Call APILog("External - "&sCase, "Could not get ElementSeries: ", -1)
End if

Help = values(11,1)

for i = 0 to 4
  for x = 0 to 5
    if(Help(i,x) = outArray(i,x)) Then
    Else
      Call APILog("External - "&sCase, "SetProperties Elementseries did not work: " -1)
    End if
  Next
Next

iResult = SCModel.GetProperty(Id, "DrillDownOptions", outArray)
if (iResult <> 0) Then
  Call APILog("External - "&sCase, "Could not get DrillDownOptions: ", -1)
End if

for i = lbound(outArray,1) to uBound(outArray,1)
  if (""&outArray(i,1) = ""&values(12,1)(i,1)) Then
  else
    Call APILog("External - "&sCase, ""&outArray(i,0)&": "&outArray(i,1), -1)
    Call APILog("External - "&sCase, ""&values(12,1)(i,0)&": "&values(12,1)(i,1), -1)
    pass = false
  End if
Next

iResult = SCModel.GetProperty(Id, "Roles", outRoles)
if (iResult <> 0) Then
  Call APILog(""&sCase, "Could not get Roles: ", -1)
End if

for i = 0 to 2
```

```
      if (""&outRoles(0,i) = ""&values(13,1)(0,i)) Then
      else
        Call APILog("External - "&sCase, ""&outRoles(i,0), -1)
        Call APILog("External - "&sCase, ""&values(13,1)(0,i), -1)
        pass = false
      End if
    Next
  End if

  if(sCase = "Alert") Then
    iResult = SCModel.GetProperty(Alertid, "Recipients", outArray)
    if (iResult <> 0) Then
      Call APILog("External - "&sCase, "Could not get property 'Recipients': ", -1)
    End if

    Help = values(5,1)

    for i = ubound(outArray,1) to lbound(outArray,1)
      if(Help(i) = RecArray(i)) Then
      else
        Call APILog("External - "&sCase, "Problems with 'recipients': ", -1)
      End if
    Next
  End if
End Sub
'*************************************************
'*************************************************
Sub BuildArrayAndCompare(Id, PropertyArray, number, scase, pass)
  Dim outProperties()
  Dim i
  Dim Help1
  Dim Help2
  Dim x
  Dim y
```

```
  iResult = SCModel.GetProperties(Id, outProperties)
   if (iResult <> 0) Then
    Call APILog("External - "&sCase, "Could not get properties: ", -1)
   End if

  for i = 0 to number
   iResult = SCModel.GetProperty(Id, PropertyArray(i,0), PropertyArray(i,1))
   if (iResult <> 0) Then
    Call APILog("External - "&sCase, "Could not get "&PropertyArray(i,0)&" value for array:
", -1)
   End if

   if(""&outProperties(i,1) = ""&PropertyArray(i,1)) Then
   Else
    Call APILog("External - "&sCase, ""&outProperties(i,0) &": "&outProperties(i,1), -1)
    Call APILog("External - "&sCase, ""&PropertyArray(i,0) &": "&PropertyArray(i,1), -1)
    pass = False
   End if
  Next

  if(sCase = "Scorecard") Then
   iResult = SCModel.GetProperty(Id, PropertyArray(17,0), PropertyArray(17,1))
   if (iResult <> 0) Then
    Call APILog("External - "&sCase, "Could not get SC InheritanceOptions: ", -1)
   End if

   Help1 = PropertyArray(17,1)
   Help2 = outProperties(17,1)

   for i = 0 to 15
    if(Help1(i,1) = Help2(i,1)) Then
    Else
     Call APILog("External - "&sCase, "InheritanceOptions has problems: ", -1)
     pass = False
    End if
```

```
    next

    iResult = SCModel.GetProperty(Id, PropertyArray(18,0), PropertyArray(18,1))
    if (iResult <> 0) Then
      Call APILog("External - "&sCase, "Could not get "&PropertyArray(18,0)&" value for
array: ", -1)
    End if

    if(""&outProperties(18,1) = ""&PropertyArray(18,1)) Then
    Else
      Call APILog("External - "&sCase, ""&outProperties(18,0) &": "&outProperties(18,1), -1)
      Call APILog("External - "&sCase, ""&PropertyArray(18,0) &": "&PropertyArray(18,1), -1)
      pass = False
    End if
  End if

  if(sCase = "Element") Then
    for i = 17 to 20
      if(i = 20) Then
        y = 5
      else
        y = 2
      End if

      iResult = SCModel.GetProperty(Id, PropertyArray(i,0), PropertyArray(i,1))
      if (iResult <> 0) Then
        if(iResult = 2013) Then
          Call APILog("External - "&sCase, "Could not get "&PropertyArray(i,0)&" for Measure:
", 0)
        else
          Call APILog("External - "&sCase, "Could not get "&PropertyArray(i,0)&" for Measure:
", -1)
          y = 0
        End if
      else
```

```
      Help1 = PropertyArray(i,1)
      Help2 = outProperties(i,1)

      for x = 0 to y
       if(""&outProperties(x,1) = ""&PropertyArray(x,1)) Then
       Else
         Call APILog("External - "&sCase, ""&outProperties(i,0) &": "&outProperties(i,1), -1)
         Call APILog("External - "&sCase, ""&PropertyArray(i,0) &": "&PropertyArray(i,1), -
1)
         pass = False
       End if
      next
     End if
   Next

   for i = 21 to 25
    iResult = SCModel.GetProperty(Id, PropertyArray(i,0), PropertyArray(i,1))
    if (iResult <> 0) Then
     Call APILog("External - "&sCase, "Could not get "&PropertyArray(i,0)&" value for ar-
ray: ", -1)
    End if

    if(""&outProperties(i,1) = ""&PropertyArray(i,1)) Then
    Else
     Call APILog("External - "&sCase, ""&outProperties(i,0) &": "&outProperties(i,1), -1)
     Call APILog("External - "&sCase, ""&PropertyArray(i,0) &": "&PropertyArray(i,1), -1)
     pass = False
    End if
   Next

   for i = 26 to 28
    if (i = 26) Then
     iResult = SCModel.GetProperty(Id, PropertyArray(i,0), PropertyArray(i,1))
     if (iResult <> 0) Then
      if(iResult = 2013) Then
```

```
          Call APILog("External - "&sCase, "Could not get "&PropertyArray(i,0)&" for Meas-
ure: ", 0)
         else
          Call APILog("External - "&sCase, "Could not get "&PropertyArray(i,0)&" for Meas-
ure: ", -1)
         End if
        else
         for x = lbound(PropertyArray(i,1)) to uBound(PropertyArray(i,1))
         if (""&PropertyArray(i,1)(x) = ""&outProperties(i,1)(x)) Then
         else
          Call APILog("External - "&sCase, "Child Elements: "&PropertiesArray(i,1)(x), -1)
          Call APILog("External - "&sCase, "Child Elements: "&outProperties(i,1)(x), -1)
          pass = False
         End if
        Next
        End if
      End if

      if (i = 27) Then
       iResult = SCModel.GetProperty(Id, PropertyArray(i,0), PropertyArray(i,1))
       if (iResult <> 0) Then
         Call APILog("External - "&sCase, "Could not get "&PropertyArray(i,0)&" value for
array: ", -1)
        End if

        for x = lbound(PropertyArray(i,1)) to uBound(PropertyArray(i,1))
         if (""&PropertyArray(i,1)(x,0) = ""&outProperties(i,1)(x,0)) Then
         else
          Call APILog("External - "&sCase, "Roles: "&PropertiesArray(i,1)(x,0), -1)
          Call APILog("External - "&sCase, "Roles: "&outProperties(i,1)(x,0), -1)
          pass = False
         End if
        Next
      End if
```

```
      if (i = 28) Then
        iResult = SCModel.GetProperty(Id, PropertyArray(i,0), PropertyArray(i,1))
        if (iResult <> 0) Then
          Call APILog("External - "&sCase, "Could not get "&PropertyArray(i,0)&" value for
array: ", -1)
        end if

        for x = lbound(PropertyArray(i,1)) to uBound(PropertyArray(i,1))
          if (""&PropertyArray(i,1)(x,1) = ""&outProperties(i,1)(x,1)) Then
          else
            Call APILog("External - "&sCase, ""&PropertyArray(i,1)(x,0)&":
"&PropertyArray(i,1)(x,1), -1)
            Call APILog("External - "&sCase, ""&outProperties(i,1)(x,0)&":
"&outProperties(i,1)(x,1), -1)
            pass = False
          End if
        Next
      End if
    Next
  End if

  if(sCase = "PeriodLevel") Then
    iResult = SCModel.GetProperty(Id, PropertyArray(14,0), PropertyArray(14,1))
    if (iResult <> 0) Then
      Call APILog("External - "&sCase, "Could not get Period for PeriodLevel: ", -1)
    End if

    Help1 = PropertyArray(14,1)
    Help2 = outProperties(14,1)

    for i = 0 to 4
      if(Help1(i,1) = Help2(i,1)) Then
      Else
        Call APILog("External - "&sCase, "Period has problems: ", -1)
        pass = False
```

```
    End if
   next
  End if

  if(sCase = "ValueSetting") Then
   for x = 14 to 15
    iResult = SCModel.GetProperty(Id, PropertyArray(x,0), PropertyArray(x,1))
    if (iResult <> 0) Then
     Call APILog("External - "&sCase, "Could not get "&PropertyArray(x,0)&" for ValueSet-
ting: ", -1)
    End if

    Help1 = PropertyArray(x,1)
    Help2 = outProperties(x,1)

    for i = 0 to 2
     if(Help1(i,1) = Help2(i,1)) Then
     Else
      Call APILog("External - "&sCase, "Period or Series has problems: ", -1)
      pass = False
     End if
    next
   Next
  End if

  if(sCase = "Alert") Then
   iResult = SCModel.GetProperty(Id, PropertyArray(5,0), PropertyArray(5,1))
   if (iResult <> 0) Then
    Call APILog("External - "&sCase, "Could not get "&PropertyArray(5,0)&" for Alert: ", -1)
   End if

   Help1 = PropertyArray(5,1)
   Help2 = outProperties(5,1)

   if (Help1 = Help2) Then
```

```
    Else
      Call APILog("External - "&sCase, "'Recipients does not match: ", -1)
    End if
  End if
End Sub
'*************************************************
'*************************************************
Sub GettingElements(value, outElement, sCase)

  iResult = SCModel.GetElements(value, outElement)
  if (iResult <> 0) Then
    Call APILog("External - "&sCase, "Could not get "&sCase &" - elements: ", -1)
  End if

End Sub
'*************************************************
'*************************************************
Sub GetPropertiesEX(Id, Properties, sCase, pass)
  Dim outValues
  Dim i, num
  Dim PropString
  Dim Help
  Dim outProperty
  Dim x, y, z
  Dim outProperties

  PropString = ""

  for i = lBound(Properties, 1) to uBound(Properties, 1)
    Propstring = Propstring + ""&Properties(i,0)&","
  Next

  num = Len(Propstring)

  PropString = Left(PropString, num-1)
```

```
 iResult = SCModel.GetPropertiesEX(Id, Propstring, outValues)
 if (iResult <> 0) Then
  Call APILog("External - GetPropertiesEX: "&sCase, "Could not get properties: ", -1)
 End if

 for i = lbound(outValues, 2) to uBound(outValues, 2)
  if (IsArray(outValues(0,i))=FALSE) Then
   iResult = SCModel.GetProperty(Id, ""&Properties(i,0), outProperty)
   if (iResult <> 0) Then
    if (iResult = 2013) Then
     Call APILog("External - GetPropertiesEX: "&sCase, "Can't compare
'"&Properties(i,0)&"': ", 0)
    else
     Call APILog("External - GetPropertiesEX: "&sCase, "Could not get compare property:
", -1)
    End if
   Else
    Call Compare(outProperty, Properties(i,0), outvalues(0,i), pass, sCase)
   End if
  Else
   if (sCase = "Alert") Then
    for x = lbound(outValues(0,i), 1) to uBound(outValues(0,i), 1)
     iResult = SCModel.GetProperty(Id, ""&Properties(i,0), outProperties)
     if (iResult <> 0) Then
      Call APILog("External - GetPropertiesEX: "&sCase, "Could not get property
'"&Properties(i,0)&"': ", -1)
     End if

     if(""&outValues(0,i)(x)=""&outProperties(x)) Then
     else
      Call APILog("GetPropertiesEX: "&sCase, "Different values:
"&outProperties(x)&"/"&outValues(0,i)(x), -1)
     End if
    Next
```

```
    else
      for x = lbound(outValues(0,i), 1) to uBound(outValues(0,i), 1)
       iResult = SCModel.GetProperty(Id, ""&Properties(i,0), outProperties)
       if (iResult <> 0) Then
         Call APILog("External - GetPropertiesEX: "&sCase, "Could not get properties array:
", -1)
       End if

       if (""&outProperties(x,1) = ""&outValues(0,i)(x,1)) Then
       else
         Call APILog("GetPropertiesEX "&sCase, "Different values:
"&outProperties(x,1)&"/"&outValues(0,i)(x,1), -1)
       End if
      Next
    End if
  End if
 Next

 if (pass = TRUE) Then
  Call APILog("External - "&sCase, "GetPropertiesEX worked OK", 0)
 else
  Call APILog("External - "&sCase, "Getproperties did not work. Check log for more de-
tails!", -1)
 End if

End Sub
'*************************************************
'*************************************************
Function ScorecardValues(values, Inheritance)
 values(0,0) = "NAME"
 values(0,1) = "Changed Scorecard name"
 values(1,0) = "SYMBOL"
 values(1,1) = "CSCN"
 values(2,0) = "DESCRIPTION"
 values(2,1) = "This is a description"
```

```
  values(3,0) = "PARENTSCORECARDID"
  values(3,1) = ""&iSCId
  values(4,0) = "BASESCORECARD"
  values(4,1) = ""&iSCId
  values(5,0) = "INHERITANCEOPTIONS"
  values(5,1) = Inheritance
  values(6,0) = "OWNER"
  values(6,1) = "1"
End Function
'*************************************************
'*************************************************
Function InheritanceOptions(Inheritance)
  Inheritance(0,0) ="Name"
  Inheritance(0,1) = 1
  Inheritance(1,0) = "Description"
  Inheritance(1,1) = 1
  Inheritance(2,0) = "ElementType"
  Inheritance(2,1) = 1
  Inheritance(3,0) = "MeasureUnit"
  Inheritance(3,1) = 1
  Inheritance(4,0) = "Roles"
  Inheritance(4,1) = 1
  Inheritance(5,0) = "ValueSettings"
  Inheritance(5,1) = 1
  Inheritance(6,0) = "PeriodLevel"
  Inheritance(6,1) = 1
  Inheritance(7,0) = "AccumulationRule"
  Inheritance(7,1) = 1
  Inheritance(8,0) = "IndicatorProperties"
  Inheritance(8,1) = 1
  Inheritance(9,0) = "ChartProperties"
  Inheritance(9,1) = 1
  Inheritance(10,0) = "ViewSettings"
  Inheritance(10,1) = 1
  Inheritance(11,0) = "SeriesSettings"
```

```
  Inheritance(11,1) = 1
  Inheritance(12,0) = "Values"
  Inheritance(12,1) = 1
  Inheritance(13,0) = "ScorecardStructure"
  Inheritance(13,1) = 1
  Inheritance(14,0) = "ScorecardStructure_noref"
  Inheritance(14,1) = 1
  Inheritance(15,0) = "ScorecardStructure_addonly"
  Inheritance(15,1) = 1
End Function
'************************************************
'************************************************
Function ElementValues(values)
  Dim aRoles(0,2)

  Call RoleValues(aRoles)

  values(0,0) = "NAME"
  values(0,1) = "Changed Element name"
  values(1,0) = "SYMBOL"
  values(1,1) = "CELEN"
  values(2,0) = "DESCRIPTION"
  values(2,1) = "This is a description for the element"
  values(3,0) = "PARENTELEMENTID"
  values(3,1) = TopElementId
  values(4,0) = "MEASUREUNITID"
  values(4,1) = "272"
  values(5,0) = "INCHARGE"
  values(5,1) = "1"
  values(6,0) = "ACCUMULATIONRULE"
  values(6,1) = "1"
  values(7,0) = "VALUESETTINGID"
  values(7,1) = "24"
  values(8,0) = "PERIODLEVELID"
  values(8,1) = "36"
```

```
  values(9,0) = "CHARTTEMPLATE"
  values(9,1) = "Month Template"
  values(10,0) = "CHARTTEMPLATEID"
  values(10,1) = "149"
  values(11,0) = "ELEMENTSERIES"
  values(11,1) = ""
  values(12,0) = "DRILLDOWNOPTIONS"
  values(12,1) = ""
  values(13,0) = "ROLES"
  values(13,1) = aRoles
End Function
'************************************************
'************************************************
Function GetPerLevProp(values)
  values(0,0) = "NAME"
  values(0,1) = "PerLev"
  values(1,0) = "SYMBOL"
  values(1,1) = "PeLe"
  values(2,0) = "DESCRIPTION"
  values(2,1) = "This is a description"
  values(3,0) = "PERIODLEVELPREFIX"
  values(3,1) = "What?"
  values(4,0) = "PERIODLEVELTYPE"
  values(4,1) = 100
  values(5,0) = "PERIODLEVELACCUMULATION"
  values(5,1) = 1
End Function
'************************************************
'************************************************
Function RangeValues(values)
  values(0,0) = "NAME"
  values(0,1) = "New Name"
  values(1,0) = "SYMBOL"
  values(1,1) = "NN"
  values(2,0) = "DESCRIPTION"
```

```
  values(2,1) = "This is still a description"
  values(3,0) = "RANGEVALUE"
  values(3,1) = 3
End Function
'*************************************************
'*************************************************
Function ChartValues(values)
  values(0,0) = "NAME"
  values(0,1) = "New Chart"
  values(1,0) = "SYMBOL"
  values(1,1) = "NC"
  values(2,0) = "DESCRIPTION"
  values(2,1) = "This is yet another description"
  values(3,0) = "AUTHOR"
  values(3,1) = 1
End Function
'*************************************************
'*************************************************
Function SeriesValues(values)
  values(0,0) = "NAME"
  values(0,1) = "SeriesName"
  values(1,0) = "SYMBOL"
  values(1,1) = "SN"
  values(2,0) = "DESCRIPTION"
  values(2,1) = "This is a description"
  values(3,0) = "REVERSEDTREND"
  values(3,1) = "True"
  values(4,0) = "TRENDTYPE"
  values(4,1) = "0"
  values(5,0) = "DEFAULTFORMULA"
  values(5,1) = "5+2"
End Function
'*************************************************
'*************************************************
Function ValueSettingValues(values)
```

```
  values(0,0) = "NAME"
  values(0,1) = "ValueName"
  values(1,0) = "SYMBOL"
  values(1,1) = "VN"
  values(2,0) = "DESCRIPTION"
  values(2,1) = "Guess what this is"
  values(3,0) = "DEFAULTSERIESID"
  values(3,1) = SeriesId
  values(4,0) = "FORCEMIN"
  values(4,1) = False
  values(5,0) = "FORCEMAX"
  values(5,1) = True
End Function
'*************************************************
'*************************************************
Function InfoItemValues(values)
  values(0,0) = "NAME"
  values(0,1) = "InfoItem"
  values(1,0) = "SYMBOL"
  values(1,1) = "II"
  values(2,0) = "DESCRIPTION"
  values(2,1) = "INFO: This is a description!"
  values(3,0) = "LOCATION"
  values(3,1) = "0"
End Function
'*************************************************
'*************************************************
Function MeasureUnitValues(values)
  values(0,0) = "NAME"
  values(0,1) = "MeasureUnit"
  values(1,0) = "SYMBOL"
  values(1,1) = "MeaU"
  values(2,0) = "DESCRIPTION"
  values(2,1) = "MeaUnit Description"
  values(3,0) = "UNITSYMBOL"
```

```
  values(3,1) = "Z"
  values(4,0) = "NUMBEROFDECIMALS"
  values(4,1) = "3"
  values(5,0) = "DECIMALSEPARATOR"
  values(5,1) = ","
  values(6,0) = "DIGITSINGROUP"
  values(6,1) = "3"
  values(7,0) = "DIGITSEPARATOR"
  values(7,1) = "."
  values(8,0) = "UNITBEFOREVALUE"
  values(8,1) = "1"
End Function
'************************************************
'************************************************
Function AllScorecard(values)
  values(0,0) = "Name"
  values(1,0) = "Symbol"
  values(2,0) = "ElementTypeId"
  values(3,0) = "Description"
  values(4,0) = "Author"
  values(5,0) = "ScorecardId"
  values(6,0) = "CreatedDate"
  values(7,0) = "ModifiedDate"
  values(8,0) = "ReferenceId"
  values(9,0) = "SourceModelId"
  values(10,0) = "SourceObjectId"
  values(11,0) = "ParentScorecardId"
  values(12,0) = "TopElementId"
  values(13,0) = "Order"
  values(14,0) = "Level"
  values(15,0) = "Index"
  values(16,0) = "BaseScorecard"
  values(17,0) = "InheritanceOptions"
  values(18,0) = "Owner"
End Function
```

```
'************************************************
'************************************************
Function AllMeasure(values)
  values(0,0) = "Name"
  values(1,0) = "Symbol"
  values(2,0) = "ElementTypeId"
  values(3,0) = "Description"
  values(4,0) = "Author"
  values(5,0) = "ScorecardId"
  values(6,0) = "CreatedDate"
  values(7,0) = "ModifiedDate"
  values(8,0) = "ReferenceId"
  values(9,0) = "SourceModelId"
  values(10,0) = "SourceObjectId"
  values(11,0) = "ParentElementId"
  values(12,0) = "MeasureUnitId"
  values(13,0) = "InCharge"
  values(14,0) = "AccumulationRule"
  values(15,0) = "ValueSettingId"
  values(16,0) = "PeriodLevelId"
  values(17,0) = "InformationItem"
  values(18,0) = "Integration"
  values(19,0) = "LinkedData"
  values(20,0) = "ElementSeries"
  values(21,0) = "Order"
  values(22,0) = "Level"
  values(23,0) = "Index"
  values(24,0) = "ChartTemplate"
  values(25,0) = "ChartTemplateId"
  values(26,0) = "ChildMeasureIds"
  values(27,0) = "Roles"
  values(28,0) = "DrillDownOptions"
End Function
'************************************************
'************************************************
```

```
Function AllPeriodLevel(values)
  values(0,0) = "Name"
  values(1,0) = "Symbol"
  values(2,0) = "ElementTypeId"
  values(3,0) = "Description"
  values(4,0) = "Author"
  values(5,0) = "ScorecardId"
  values(6,0) = "CreatedDate"
  values(7,0) = "ModifiedDate"
  values(8,0) = "ReferenceId"
  values(9,0) = "SourceModelId"
  values(10,0) = "SourceObjectId"
  values(11,0) = "PeriodLevelPrefix"
  values(12,0) = "PeriodLevelType"
  values(13,0) = "PeriodLevelAccumulation"
  values(14,0) = "Period"
End Function
'*************************************************
'*************************************************
Function AllRange(values)
  values(0,0) = "Name"
  values(1,0) = "Symbol"
  values(2,0) = "ElementTypeId"
  values(3,0) = "Description"
  values(4,0) = "Author"
  values(5,0) = "ScorecardId"
  values(6,0) = "CreatedDate"
  values(7,0) = "ModifiedDate"
  values(8,0) = "ReferenceId"
  values(9,0) = "SourceModelId"
  values(10,0) = "SourceObjectId"
  values(11,0) = "RangeValue"
End Function
'*************************************************
'*************************************************
```

```
Function AllChartTemplate(values)
  values(0,0) = "Name"
  values(1,0) = "Symbol"
  values(2,0) = "ElementTypeId"
  values(3,0) = "Description"
  values(4,0) = "Author"
  values(5,0) = "ScorecardId"
  values(6,0) = "CreatedDate"
  values(7,0) = "ModifiedDate"
  values(8,0) = "ReferenceId"
  values(9,0) = "SourceModelId"
  values(10,0) = "SourceObjectId"
End Function
'************************************************
'************************************************
Function AllSeries(values)
  values(0,0) = "Name"
  values(1,0) = "Symbol"
  values(2,0) = "ElementTypeId"
  values(3,0) = "Description"
  values(4,0) = "Author"
  values(5,0) = "ScorecardId"
  values(6,0) = "CreatedDate"
  values(7,0) = "ModifiedDate"
  values(8,0) = "ReferenceId"
  values(9,0) = "SourceModelId"
  values(10,0) = "SourceObjectId"
  values(11,0) = "SeriesType"
  values(12,0) = "DefaultFormula"
  values(13,0) = "ReversedTrend"
  values(14,0) = "TrendType"
End Function
'************************************************
'************************************************
Function AllValueSetting(values)
```

```
  values(0,0) = "Name"
  values(1,0) = "Symbol"
  values(2,0) = "ElementTypeId"
  values(3,0) = "Description"
  values(4,0) = "Author"
  values(5,0) = "ScorecardId"
  values(6,0) = "CreatedDate"
  values(7,0) = "ModifiedDate"
  values(8,0) = "ReferenceId"
  values(9,0) = "SourceModelId"
  values(10,0) = "SourceObjectId"
  values(11,0) = "DefaultSeriesId"
  values(12,0) = "ForceMin"
  values(13,0) = "ForceMax"
  values(14,0) = "Range"
  values(15,0) = "Series"
End Function
'*************************************************
'*************************************************
Function AllInfoItem(values)
  values(0,0) = "Name"
  values(1,0) = "SYMBOL"
  values(2,0) = "ELEMENTTYPEID"
  values(3,0) = "DESCRIPTION"
  values(4,0) = "AUTHOR"
  values(5,0) = "SCORECARDID"
  values(6,0) = "CREATEDDATE"
  values(7,0) = "MODIFIEDDATE"
  values(8,0) = "REFERENCEID"
  values(9,0) = "SOURCEMODELID"
  values(10,0) = "SOURCEOBJECTID"
  values(11,0) = "LOCATION"
End Function
'*************************************************
'*************************************************
```

```
Function AllMeasureUnit(values)
  values(0,0) = "NAME"
  values(1,0) = "SYMBOL"
  values(2,0) = "ELEMENTTYPEID"
  values(3,0) = "DESCRIPTION"
  values(4,0) = "AUTHOR"
  values(5,0) = "SCORECARDID"
  values(6,0) = "CREATEDDATE"
  values(7,0) = "MODIFIEDDATE"
  values(8,0) = "REFERENCEID"
  values(9,0) = "SOURCEMODELID"
  values(10,0) = "SOURCEOBJECTID"
  values(11,0) = "UNITSYMBOL"
  values(12,0) = "NUMBEROFDECIMALS"
  values(13,0) = "DECIMALSEPARATOR"
  values(14,0) = "DIGITSINGROUP"
  values(15,0) = "DIGITSEPARATOR"
  values(16,0) = "UNITBEFOREVALUE"
End Function
'*************************************************
'*************************************************
Function AllAlert(values)
  values(0,0) = "AlertType"
  values(1,0) = "ElementTypeId"
  values(2,0) = "AlertRangeId"
  values(3,0) = "AlertDelay"
  values(4,0) = "Delayed"
  values(5,0) = "AlertURL"
  values(6,0) = "LastAlertedDate"
  values(7,0) = "LastAlertedPeriodId"
  values(8,0) = "Recipients"
End Function
'*************************************************
'*************************************************
Function LinkedDataValues(values)
```

```
  values(0,0) = "Name"
  values(0,1) = "LD name"
  values(1,0) = "Description"
  values(1,1) = "LD Description"
  values(2,0) = "Status"
  values(2,1) = 1
End Function
'*************************************************
'*************************************************
Function AlertValues(values)
  values(0,0) = "AlertType"
  values(0,1) = "0"
  values(1,0) = "AlertRangeId"
  values(1,1) = "15"
  values(2,0) = "Delayed"
  values(2,1) = "True"
  values(3,0) = "AlertDelay"
  values(3,1) = "2"
  values(4,0) = "AlertURL"
  values(4,1) = "http://www.qpr.com"
  values(5,0) = "Recipients"
  values(5,1) = RecArray
End Function
'*************************************************
'*************************************************
Function AllLinkedData(values)
  values(0,0) = "Name"
  values(1,0) = "Symbol"
  values(2,0) = "ElementTypeId"
  values(3,0) = "Description"
  values(4,0) = "Author"
  values(5,0) = "ScorecardId"
  values(6,0) = "CreatedDate"
  values(7,0) = "ModifiedDate"
  values(8,0) = "ReferenceId"
```

```
  values(9,0) = "SourceModelId"
  values(10,0) = "SourceObjectId"
  values(11,0) = "Status"
End Function
'*************************************************
'*************************************************
Function RoleValues(Roles)
  Roles(0,0) = "1"
  Roles(0,1) = "In Charge"
  Roles(0,2) = "1"
End Function
'*************************************************
'*************************************************
Function Compare(outvalue, property, values, pass, sCase)
  if (""&outValue = ""&values) Then
  else
    pass = False
    Call APILog("External - "&sCase, "Property: "&property &", Value: "&values, -1)
    Call APILog("External - "&sCase, "Property: "&property &", Value: "&outValue, -1)
  End if
End Function
'*************************************************
'*************************************************
Function Result(sCase, pass)
  if (pass = False) Then
    Call APILog("External - "&sCase, "Set/GetProperty did not work properly. Identificate!", -
1)
  else
    Call APILog("External - "&sCase, "Set/GetProperty worked OK", 0)
  End if
End Function
'*************************************************
'*************************************************
'*********************************************
'*********************************************
```

```
Function GetValuesFrom(values)
    Values(0,0) = "ACT"                 'seriessymbol
    Values(0,1) = "1.1.2002"            'date
    Values(0,2) = "1,4"                             'value

    Values(1,0) = "ACT"
    Values(1,1) = "5.7.2002"
    Values(1,2) = "2,66"

    Values(2,0) = "ACT"
    Values(2,1) = "1.1.2003"
    Values(2,2) = "3,4"

    Values(3,0) = "ACT"
    Values(3,1) = "1.12.2003"
    Values(3,2) = "7,9"

    Values(4,0) = "ACT"
    Values(4,1) = "1.1.2004"
    Values(4,2) = "14,7"

    Values(5,0) = "ACT"
    Values(5,1) = "1.12.2004"
    Values(5,2) = "22,7"

    Values(6,0) = "Text"
    Values(6,1) = "1.12.2004"
    Values(6,2) = "Moro"
End Function
'***********************************************
'*********************************************
Function GetValuesFrom2(values2)
    Values2(0,0) = "ACT"                'seriessymbol
    Values2(0,1) = "1.1.2002"                       'date
    Values2(0,2) = "666"                            'value
```

```
    Values2(1,0) = "ACT"
    Values2(1,1) = "5.7.2002"
    Values2(1,2) = "666"

    Values2(2,0) = "ACT"
    Values2(2,1) = "1.1.2003"
    Values2(2,2) = "666"

    Values2(3,0) = "ACT"
    Values2(3,1) = "1.12.2003"
    Values2(3,2) = "666"

    Values2(4,0) = "ACT"
    Values2(4,1) = "1.1.2004"
    Values2(4,2) = "666"

    Values2(5,0) = "ACT"
    Values2(5,1) = "1.12.2004"
    Values2(5,2) = "666"

    Values2(6,0) = "Text"
    Values2(6,1) = "1.12.2004"
    Values2(6,2) = "Changed"
End Function
'************************************************
'*******************************************
Function ArrayOfProperties(Properties)
    Properties(0,0) = "DESCRIPTION"
    Properties(0,1) = "This tests set properties"

    Properties(1,0) = "MEASUREUNITID"
    Properties(1,1) = "272"

    Properties(2,0) = "ACCUMULATIONRULE"
```

```
    Properties(2,1) = "7"

    Properties(3,0) = "VALUESETTINGID"
    Properties(3,1) = "24"

    Properties(4,0) = "PERIODLEVELID"
    Properties(4,1) = "36"
End Function
'************************************************
'************************************************
Function SetMeaFormula(PerspectiveId)
  Dim fail

  if (ViewUser = "ViewUser") Then
    fail = 0
  Else
    fail = -1
  End if

  iResult = SCModel.SetFormula(PerspectiveId, "UALA", "AVERAGE_(ACT()+1)")
  If (iResult <> 0) Then
    Call APILog("SetMeaFormula", "Could not set formula for 'UALA': ", fail)
  End If

  iResult = SCModel.SetFormula(PerspectiveId, "UTAR", "AVERAGE_(ACT()-1)")
  If (iResult <> 0) Then
    Call APILog("SetMeaFormula", "Could not set formula for 'UTAR': ", fail)
  End If

  iResult = SCModel.SetFormula(PerspectiveId, "LTAR", "AVERAGE_(ACT()-2)")
  If (iResult <> 0) Then
    Call APILog("SetMeaFormula", "Could not set formula for 'LTAR': ", fail)
  End If

  iResult = SCModel.SetFormula(PerspectiveId, "LALA", "AVERAGE_(ACT()-3)")
```

```
  If (iResult <> 0) Then
    Call APILog("SetMeaFormula", "Could not set formula for 'LALA': ", fail)
  End If
End Function
'************************************************
'************************************************
Function SetMeaFormula2(iSCId, ElementRights)
  Dim fail

  if (ViewUser = "ViewUser") Then
    fail = 0
  Else
    fail = -1
  End if

  iResult = SCModel.SetFormula(iSCId, "UALA", "AVERAGE_(ACT()+100)")
  If (iResult <> 0) Then
    Call APILog("SetMeaFormula2", "Could not set formula for 'UALA': ", fail)
  End If

  iResult = SCModel.SetFormula(iSCId, "UTAR", "AVERAGE_(ACT()-100)")
  If (iResult <> 0) Then
    Call APILog("SetMeaFormula2", "Could not set formula for 'UTAR': ", fail)
  End If

  iResult = SCModel.SetFormula(iSCId, "LTAR", "AVERAGE_(ACT()-100)")
  If (iResult <> 0) Then
    Call APILog("SetMeaFormula2", "Could not set formula for 'LTAR': ", fail)
  End If

  iResult = SCModel.SetFormula(iSCId, "LALA", "AVERAGE_(ACT()-300)")
  If (iResult <> 0) Then
    Call APILog("SetMeaFormula2", "Could not set formula for 'LALA': ", fail)
  End If
End Function
```

```
'************************************************
'************************************************
Sub APILog(log1, log2, error)
  Dim objFSO, objFolder, objShell, objTextFile
  Dim dDate
  Dim dTime
  Dim strFile
  Dim objFSO2, objFolder2, objShell2, strDirectory2

  strDirectory2 = "C:\Regression"

  if (""&log2 = "Starting to run SC_Regression.smf") Then
    Set objFSO2 = CreateObject("Scripting.FileSystemObject")
    if objFSO2.FolderExists(strDirectory2) Then
      Set objFolder2 = objFSO2.GetFolder(strDirectory2)
      Call APILog("APILog", "Folder has already been created", 0)
    else
      Set objFolder2 = objFSO2.CreateFolder(strDirectory2)
      Call APILog("APILog", "Created folder : "&strDirectory2, 0)
    End if
  End if

  if (error = -1 OR error = "") Then
    completed = False
  End if

  strFile = "APILog.txt"

  dTime = Time
  dDate = Date

  Set objFSO = CreateObject("Scripting.FileSystemObject")

  If objFSO.FileExists(""&strFile) = 0 Then
    Set objTextFile = objFSO.CreateTextFile(""&strFile)
```

```
    objTextFile.WriteLine(""&dTime &"                    "&dDate &"  "&log1 &" "&log2 +
SCApplication.GetErrorMessage(iResult))
    if (completed = False) Then
      objTextFile.WriteLine(""&dTime &"                  "&dDate &"  "&log1 &" Er-
ror:"&log2 + SCApplication.GetErrorMessage(iResult))
      SCApplication.Quit
    End if
    objTextFile.Close
  Else
    Set objTextFile = objFSO.OpenTextFile(""&strFile, 8, True)
    objTextFile.WriteLine(""&dTime &"                    "&dDate &"  "&log1 &" "&log2 +
SCApplication.GetErrorMessage(iResult))
    if (completed = False) Then
      objTextFile.WriteLine(""&dTime &"                  "&dDate &"  "&log1 &" Er-
ror:"&log2 + SCApplication.GetErrorMessage(iResult))
      SCApplication.Quit
    End if
    objTextFile.Close
  End If

  if (log2 = "SC_Advanced complete") Then
    if (completed = False) Then
      Set objTextFile = objFSO.OpenTextFile(""&strFile, 8, True)
      objTextFile.WriteLine(""&dTime &"                  "&dDate &"  Error: Errors occurred
while executing script")
      objTextFile.Close
    else
      Set objTextFile = objFSO.OpenTextFile(""&strFile, 8, True)
      objTextFile.WriteLine(""&dTime &"                  "&dDate &"  All OK!")
      objTextFile.Close
    End if
  End if
End Sub
'*************************************************************************
'*************************************************************************
```

CART - Määrityskuvasto

Juho Vuorio

Määrityskuvasto v.1.0

2010

Versio	Päivämäärä	Muutokset
0.1	**1.6.2009**	
0.11	**20.6.2009**	**Käyttötapaukset päivitetty**
0.2	**21.6.2009**	**Kuvat lisätty, Kappale 4 lisätty**
0.21	**23.6.2009**	**Lähdeluettelo ja sanasto lisätty**
0.22	**24.6.2009**	**Delphi Unit testing project lisätty. Intermediate tasot lisätty testeihin. Kaikki kuvat päivitetty**
1.0	**10.4.2010**	**Määrityskuvasto päivitetty uusien raportointiohjeiden mukaiseksi**

Tiivistelmä

Kyseisessä määrityskuvastossa käydään läpi Continuous Automated Regression Testing tool – ohjelman määritykset. Kuvasto pitää sisällään järjestelmän ympäristön, käyttötapaukset sekä yhteenvedon ohjelman käyttöoikeuksista.

Sisällys

1 Johdanto

Määrityskuvastossa kuvataan QPR Softwarelle kehitettävän automaattisen regressiotestaukseen tarkoitettavaa järjestelmää. Testeusjärjestelmän avulla, ohjelmistokehittäjät voivat testata ohjelman toimivuuden helposti ja nopeasti.

2 Järjestemän ympäristö

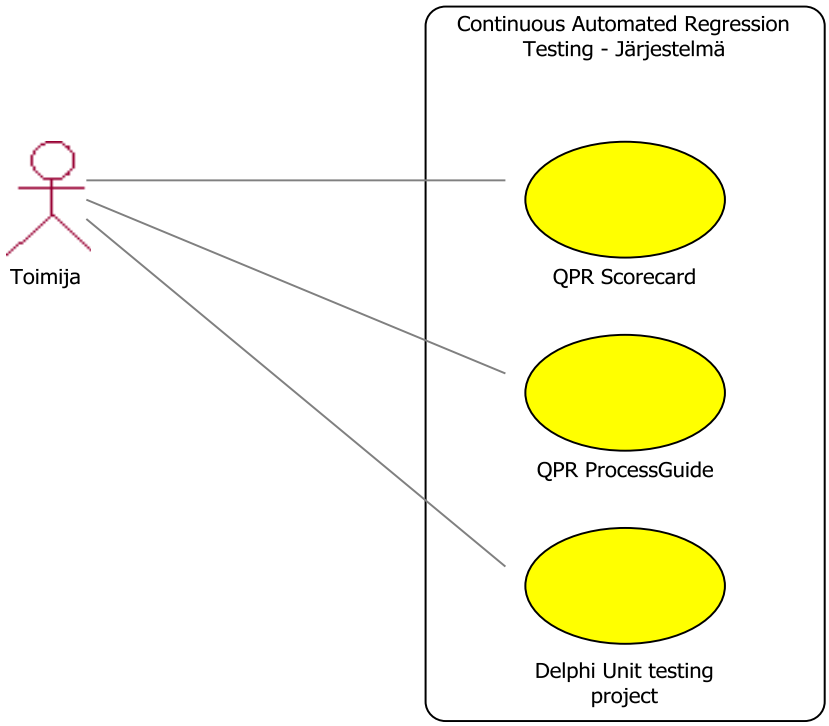
2.1 Yleiskuvaus

Continuous Automated Regression Testing tool – järjestelmä mahdollistaa helpon ja nopean tavan testata QPR Processguide ja QPR scorecard – ohjelmien toiminnallisuutta. Järjestelmällä voidaan suorittaa joko kevyemmät, keskitason tai raskaammat regressiotestit. Vaihtoehdot ovat PG Smoke, SC Smoke, PG Intermediate, SC Intermediate, PG Advanced ja SC Advanced.

Smoke – testit ovat kevyitä ja nopeita testejä, jotka varmistavat vain ohjelman toiminnallisuuden päällisin puolin. Intermediate – testit varmistavat ohjelman perustoiminnallisuudet. Advanced testit käyttävät ohjelmien ominaisuuksia monipuolisesti.

Delphi Unit testing project – on vanha yksikkötestaus projekti, jota on tarkoitus käyttää CART:n käyttöliittymän kautta.

CART on suunnattu QPR Software Oyj:n kehittäjiem sekä testaajien käytettäväksi



Kuva 1. Käyttötapauskartta

2.2 Palvelukuvaukset

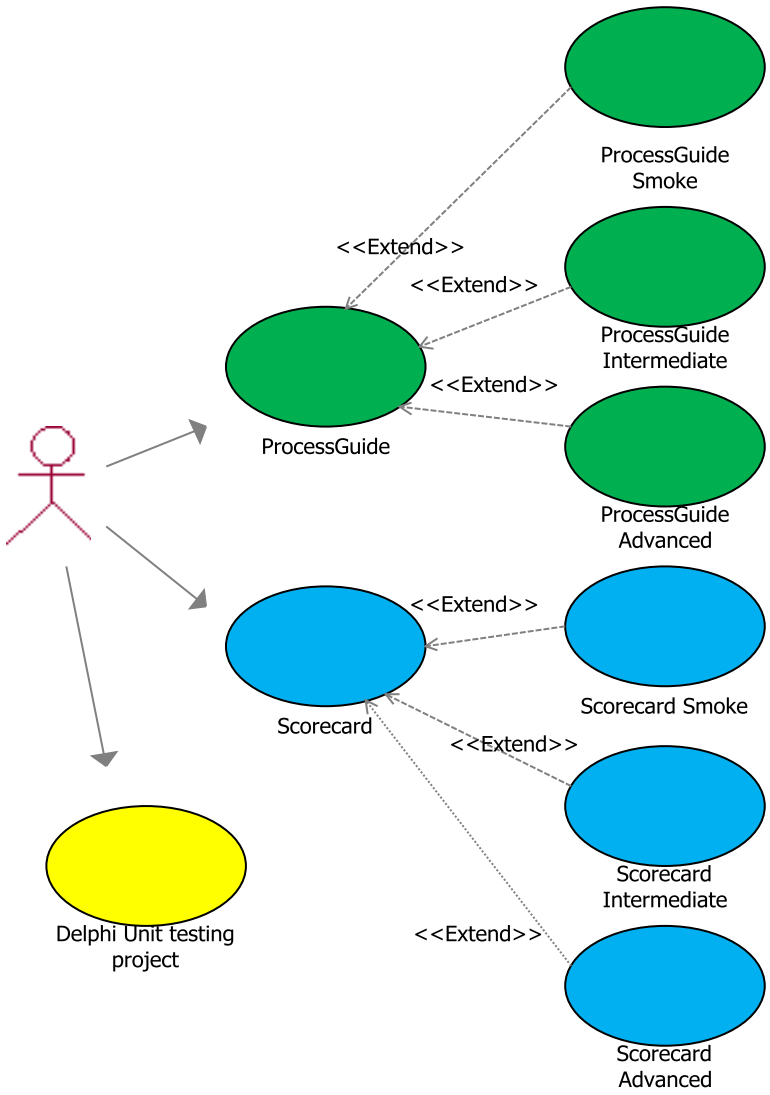
Järjestelmää käyttävät pääsääntöisesti ohjelmistokehittäjät ja tarvittaessa myös ohjelmistotestaajat.

2.3 Toimijakuvaukset

Ohjelmistokehittäjillä ja testaajilla on oikeus kaikkiin ohjelman toimintoihin. Testaaja päivittää tarvittaessa testitapauksia ja testiaineistoa.

3 Käyttötapaukset

3.1 Palveluiden väliset riippuvuudet



kuva 2. Riippuvuuskaavio

3.2 QPR Processguide

Toimija: Kehittäjä tai testaaja

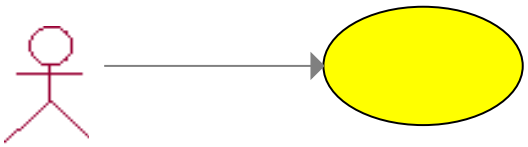
Lopputulos: Järjestelmä on ajanut valitut testit ja ilmoittanut testien tulokset

1. Järjestelmä näyttää aloitusvalikon
2. Toimija valitsee haluamansa toiminnon seuraavista vaihtoehdoista: Processguide Smoke, Processguide Intermediate, Processguide Advanced

V2.1. Toimija valitsee Processguide Smoke -> Osakäyttötapaus Processguide Smoke

V2.2. Toimija valitsee Processguide Intermediate -> Osakäyttätapaus Processguide Intermediate

V2.3. Toimija valitsee Processguide Advanced -> Osakäyttötapaus Processguide Advanced



Kuva 3. QPR Processguide

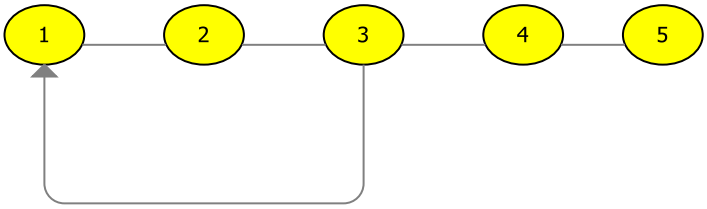
3.2.1 Processguide Smoke

Esiehto:	Yhteys tietokantaan luotu. QPR Processguide on käännetty onnistuneesti
Toimija:	Kehittäjä tai testaaja
Lopputulos:	Processguide Smoke testit on ajettu ja testien tulos ilmoitettu
Suoritustiheys:	Tarvittaessa

1. Järjestelmä näyttää aloitusvalikon
2. Toimija valitsee ”Processguide” ja ”Smoke”
3. Toimija pyytää järjestelmää suorittamaan testit

   V3. Toimija ei pyydä järjestelmää suorittamaan testejä

   P3.1 Järjestelmä ilmoittaa virheestä
4. Järjestelmä ilmoittaa testituloksen.
5. Järjestelmä siirtyy alkutilaan

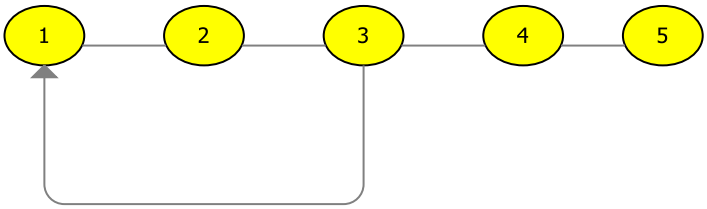


Kuva 4. Processguide Smoke

3.2.2 Processguide Intermediate

Esiehto:	Yhteys tietokantaan luotu. QPR Processguide on käännetty onnistuneesti
Toimija:	Kehittäjä tai testaaja
Lopputulos:	Processguide Intermediate testit on ajettu ja testien tulos ilmoitettu
Suoritustiheys:	Tarvittaessa

1. Järjestelmä näyttää aloitusvalikon
2. Toimija valitsee ”Processguide” ja ”Intermediate”
3. Toimija pyytää järjestelmää suorittamaan testit
   V3. Toimija ei pyydä järjestelmää suorittamaan testejä
   P3.1 Järjestelmä ilmoittaa virheestä
4. Järjestelmä ilmoittaa testituloksen
5. Järjestelmä siirtyy alkutilaan



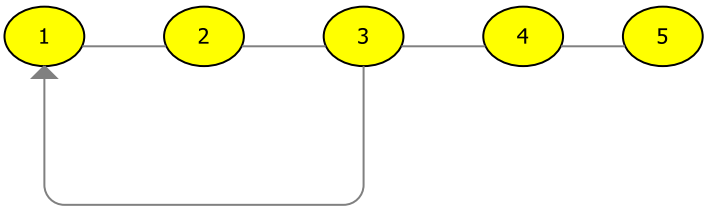
Kuva 5. Processguide Intermediate

3.2.3 Processguide Advanced

Esiehto:	Yhteys tietokantaan luotu. QPR Processguide on käännetty onnistuneesti
Toimija:	Kehittäjä tai testaaja
Lopputulos:	Processguide Advanced testit on ajettu ja testien tulos ilmoitettu.
Suoritustiheys:	Tarvittaessa

1. Järjestelmä näyttää Aloitusvalikon
2. Toimija valitsee ”Processguide” ja ”Advanced”
3. Toimija pyytää järjestelmää suorittamaan testit
   V3. Toimija ei pyydä järjestelmää suorittamaan testejä
   P3.1 Järjestelmä ilmoittaa virheestä
4. Järjestelmä ilmoittaa testituloksen

5. Järjestelmä siirtyy alkutilaan



kuva 6. Processguide Advanced

3.3 QPR Scorecard

Toimija: Kehittäja tai testaaja

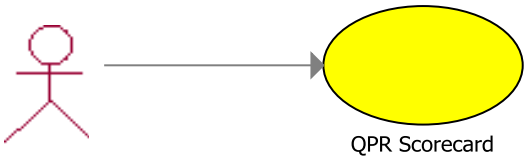
Lopputulos: Järjestelmä on ajanut valitut testit ja ilmoittanut testien tulokset

1. Järjestelmä näyttää aloitusvalikon
2. Toimija valitsee haluamansa toiminnon seuraavista: Scorecard Smoke, Scorecard Intermediate, Scorecard Advanced

   V2.1. Toimija valitsee Scorecard Smoke -> Osakäyttötapaus Scorecard Smoke

   V2.2. Toimija valitsee Scorecard Intermediate -> Osakäyttötapaus Scorecard Intermediate

   V.2.3. Toimija valitsee Scorecard Advanced -> Osakäyttötapaus Scorecard Advanced



Kuva 7. QPR Scorecard

3.3.1 Scorecard Smoke

Esiehto: Yhteystietokantaan luotu. QPR Scorecard on käännetty onnistuneesti.

Toimija: Kehittäjä tai testaaja

Lopputulos: Scorecard Smoke testit on suoritettu ja testien tulos ilmoitettu.

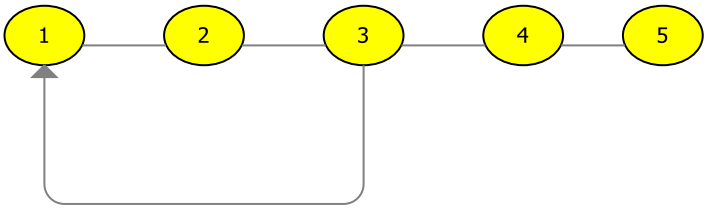
Suoritustiheys: Tarvittaessa

1. Järjestelmä näyttää aloitusvalikon

2. Toimija valitsee ”Scorecard” ja ”Smoke”
3. Toimija pyytää järjestelmää suorittamaan testit

   V3. Toimija ei pyydä järjestelmää suorittamaan testejä

   P3.1. Järjestelmä ilmoittaa virheestä
4. Järjestelmä ilmoittaa testituloksen
5. Järjestelmä siirtyy alkutilaan



kuva 8. Scorecard Smoke

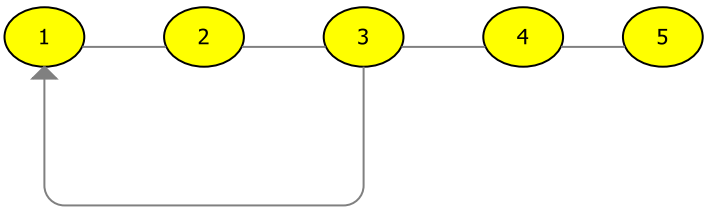
3.3.2 Scorecard Intermediate

Esiehto:	Yhteystietokantaan luotu. QPR Scorecard on käännetty onnistuneesti.
Toimija:	Kehittäjä tai testaaja
Lopputulos:	Scorecard Intermediate testit on suoritettu ja testien tulos ilmoitettu.
Suoritustiheys:	Tarvittaessa

1. Järjestelmä näyttää aloitusvalikon
2. Toimija valitsee ”Scorecard” ja ”Intermediate”
3. Toimija pyytää järjestelmää suorittamaan testit

   V3. Toimija ei pyydä järjestelmää suorittamaan testejä

   P3.1. Järjestelmä ilmoittaa virheestä
4. Järjestelmä ilmoittaa testituloksen
5. Järjestelmä siirtyy alkutilaan

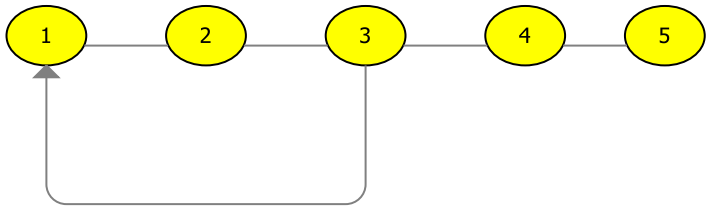


kuva 9. Scorecard Intermediate

3.3.3 Scorecard Smoke

Esiehto:	Yhteystietokantaan luotu. QPR Scorecard on käännetty onnistuneesti.
Toimija:	Kehittäjä tai testaaja
Lopputulos:	Scorecard Advanced testit on suoritettu ja testien tulos ilmoitettu.
Suoritustiheys:	Tarvittaessa

1. Järjestelmä näyttää aloitusvalikon
2. Toimija valitsee ”Scorecard” ja ”Smoke”
3. Toimija pyytää järjestelmää suorittamaan testit
   V3. Toimija ei pyydä järjestelmää suorittamaan testejä
   P3.1. Järjestelmä ilmoittaa virheestä
4. Järjestelmä ilmoittaa testituloksen
5. Järjestelmä siirtyy alkutilaan

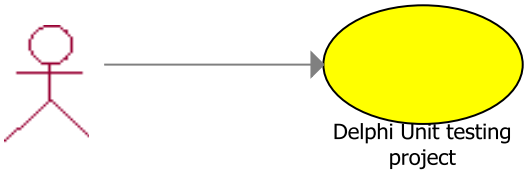


kuva 10. Scorecard Advanced

3.4 Delphi Unit testing project

Toimija:	Kehittäjä
Lopputulos:	Järjestelmä on ajanut valitut testit ja ilmoittanut testien tulokset

1. Järjestelmä näyttää aloitusvalikon
2. Toimija valitsee haluamansa toiminnon seuraavista: Delphi Unit testing project
   V2.1. Toimija valitsee Delphi Unit testing project -> Osakäyttötapaus Delphi Unit testing project.



kuva 11. Delphi Unit testing project

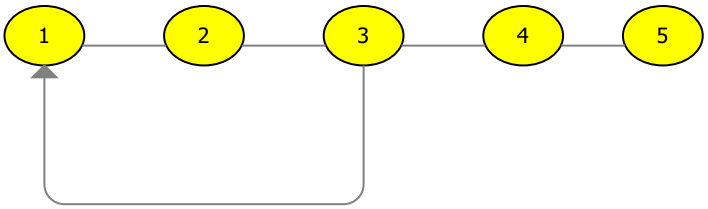
3.4.1 Delphi Unit testing project

Esiehto:	Lähdekoodi on käännetty onnistuneesti
Toimija:	Kehittäjä
Lopputulos:	Delphi Unit testing project testit ovat suoritettu ja testien tulos raportoitu
Suoritustiheys:	Tarvittaessa

1. Järjestelmä näyttää aloitusvalikon
2. Toimija valitsee ”Delphi Unit testing project”
3. Toimija pyytää järjestelmää suorittamaan testit

   V3. Toimija ei pyydä järjestelmää suorittamaan testejä

   P3.1 Järjestelmä ilmoittaa virheestä
4. Järjestelmä ilmoittaa testituloksen
5. Järjestelmä siirtyy alkutilaan



kuva 12. Delphi Unit testing project

4 Yhteenveto käyttöoikeuksista

Järjestelmällä on kaksi päätoimijaa: Kehittäjä ja testaaja. Kummallakin toimijalla on täydet oikeudet suorittaa järjestelmän testejä. Testaaja on ainoa toimija, jolla on oikeudet päivittää testitapauksia tai testimateriaalia.

Lähteet

Kurssimateriaali kurssilta SYS48D

Vanhanen, Ulla 2005: Luentotiivistelmä, Käyttötapaukset
http://myy.helia.fi/~ict2td005/maaritys/m-luento/kayttotapaukset.pdf. Luettu: 24.6.2009

CART - Projektisuunnitelma

Juho Vuorio

Projektisuunnitelma

2010

Versio	Päiväys	Muutos
0.1	**28.05.2009**	**Projektisuunnitelman ensimmäinen versio**
0.2	**29.5.2009**	**Projektin nimi käännetty suomeksi** **1.2 Mittareita lisätty projektin tavoitteeseen** **1.3 Järjestelmän hyväksymiskriteereitä tarkennettu** **1.4 Lisätty jatkotoimenpiteet** **1.8.1 Työn kustannukset osio päivitetty** **2.4 Käytettävät standardit päivitetty** **Liite 2. Riskitaulukko päivitetty**
1.0	**11.04.2010**	**Raportti muokattu uuden raportointiohjeen mukaiseksi**

Tiivistelmä

CART Projektin tarkoituksena on luoda testausjärjestelmä kehittäjille regressiotestaukseen QPR Software Oyj:n käyttöön.

Projektin tavoitteena on luoda ohjelman lisäksi myös seuraavat dokumentit:

- Suunnittelukuvasto
- Käyttöohjeet (englanti ja suomi)
- Testausdokumentaatio
- Opinnäytetyön loppuraportti

Tässä dokumentissa käydään läpi myös mahdolliset riskit, kustannukset sekä käytettävät standardit

Sisällys

1 Projektin määritys

1.1 Projektin tausta

Projektin tarkoituksena on luoda testausjärjestelmä kehittäjille regressiotestaukseen QPR Software Oyj:n käyttöön. Projektissa luotavan ohjelmiston tavoitteena vähentää korjausten ja uusien toiminnallisuuksien regressiovaikutusta ohjelmistoon. Jos mahdolliset regressiovaikutukset saadaan kiinni jo kehitysvaiheessa, säästetään huomattavasti aikaa ja resursseja.

Projektin avulla on tarkoitus täydentää automaattisten testien määrää QPR Software Oyj:ssa. Tällä hetkellä testauksessa käytetään seuraavia ohjelmia/järjestelmiä: FATE (Fully Automated Test Engine), Silk Performer sekä TestComplete. Mikään olemassa olevista järjestelmistä/ohjelmista ei ole tarkoitettu kehittäjille.

Työn tilaaja QPR Software Oyj toimittaa tarvittavat työkalut työn toteuttamiseen. Työn toimittaja toimittaa QPR Softwarelle toimivan testausjärjestelmän ja tarvittavan dokumentaation.

1.2 Projektin tehtävä

Projektin tehtävä on luoda regressiotestaus järjestelmä joka on suunnattu kehittäjille ja mitata järjestelmän vaikutusta ohjelmiston laatuun. Mittareina on testausjärjestelmän löytämät virheet suhteessa kaikkiin löydettyihin virheisiin. Toinen mittari vertailee historiatiedon pohjalta löydettyjen bugien määrää suhteessa aikaan ennen järjestelmän käyttöönottoa.

1.3 Tavoitteet ja lopputulokset

Projektin päätavoitteena on luoda toimiva ja ylläpidettävä testausjärjestelmä, joka on helppokäyttöinen ja hyvin dokumentoitu. Muina tavoittena on mitata järjestelmän vaikutusta laatuun sekä saada tarvittavat raportit luotua.

Projektissa luotavat dokumetit:

- Suunnittelukuvasto

- Käyttöohjeet (englanti ja suomi)
- Testausdokumentaatio
- Opinnäytetyön loppuraportti

Suunnittelukuvaston pohjalta luodaan varsinainen järjestelmä, joka sisältää testauksessa käytettävän ohjelman sekä testimateriaalin. Testausdokumentaation avulla varmistetaan ohjelman varsinainen toiminta.

Tavoite on saavutettu, kun kehittäjät hyväksyvät järjestelmän ja järjestelmä on toiminnassa sekä kaikki tarvittavat dokumentit on luotu. Järjestelmän hyväksymisen perustoiminnallisuudet ovat: 1. Järjestelmä asentuu ympäristöön. 2. Järjestelmä lähtee käyntiin. 3. Järjestelmä ajaa toivotut testit. 4. Järjestelmä palauttaa oikeat arvot tuloksesta.

Oppimistavoitteena on oppia hallitsemaan, suunnittelemaan ja toteuttamaan kokonainen projekti. Myös erinäisten raporttien ja ohjeiden tekemisen parempi hallitseminen on tärkeänä tavoitteena.

1.4 Tehtävän rajaus

Tämän tehtävän tarkoituksena ei ole päivittää jo olemassa olevia regressiotestaus – dokumentteja. Tuotettava ohjelmisto perustuu QPR ProcessGuide ja QPR Scorecard ohjelmien API rajapintaan, joten tehtävä ei kata muiden osien (QPR User Managment Server sekä QPR Portal) tai muiden tekniikoiden (Web Service) testaamista. Tehtävä kattaa vain yhden QPR version testaamiseen tarkoitetun ohjelmiston.

Projektin jälkeisenä aikana jatkotoimenpiteinä on nyt rajattujen ohjelmien ja tekniikoiden liittäminen / hyödyntäminen järjestelmässä.

1.5 Tilanneanalyysi ja riskit

Projektin lähtökohta on erinomainen. Projektin toteuttajalla on hyvä tuntemus testattavista ohjelmista sekä testien toteuttamiseen käytettävästä API rajapinnasta. Toteuttajan tietämys testauksesta on myös eduksi projektille.

Projektin suurimpiin riskeihin kuuluvat toteuttajan sairastuminen, muut kiireet sekä ongelmat ohjelmistossa. Tarkemmat analyysit riskeitä, katso Liite 2.

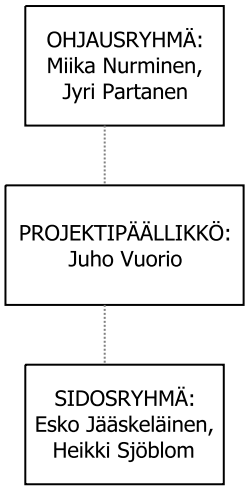
1.6 Projektiorganisaatio

Projektin organisaatio koostuu varsinaisesti Ohjausryhmästä sekä projektipäälliköstä. Sidosryhmänä toimii QPR Software Oyj:n kehittäjän sekä testaajat.

Ohjausryhmä ohjaa ja seuraa projektin edistymistä. Kaikki projektiin tehtävät muutokset on hyväksytettävä ohjausryhmällä.

Projektipäällikkö vastaa projektin aikataulun pitävyydestä. Projektipäällikkö myös vastaa projektin suunnittelusta ja toteutuksesta sekä projektin raportoinnista.

Sidosryhmä korjaa projektissa mahdollisesti löytyvät ohjelmistovirheet (QPR:n tuotteista). Testaajat auttavat kyseisten virheiden mahdollisessa testauksessa.



Kuva 1. Projektiorganisaatio

1.7 Ympäristö

Ohessa kuvataan toteutuksen sidosryhmät ja toteutuksessa tarvittavat ohjelmat ja resurssit.

1.7.1 Tuloksen sidosryhmät

Projektin pääsidosryhmät ovat henkilöiden osalta kehittäjät sekä testaajat, järjestelmien osalta testattavat ohjelmistot QPR ProcessGuide sekä QPR Scorecard.

Kehittäjät ovat tuotetun järjestelmän pääkäyttäjiä, joiden on tarkoitus käyttää ohjelmaa päivittäin.

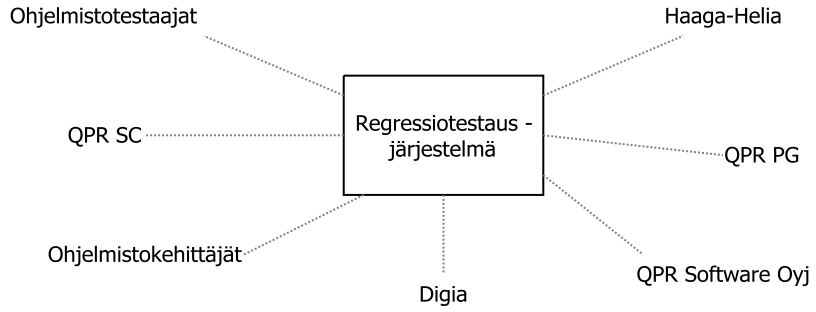
Testaajien tarkoitus on ylläpitää ja päivittää testitapauksia, kun testausjärjestelmä on käytössä ja jo toimitettu

QPR SC ja QPR PG ovat testattavat ohjelmistot.

Haaga-Helia hyväksyy opinnäytetyön loppuraportin. Oikeudet loppuraportin käytön suhteen määritelty muualla.

QPR Software Oyj omistaa oikeudet kehitettävään järjestelmään sekä sen dokumentaatioon

Digia on QPR Software Oyj:n alihankinta tiimi, jossa on kehittäjiä sekä testaajia. Heillä on samat työnkuvat kuin QPR Sortware Oyj:n työntekijöillä projektissa.



Kuva 2. Sidosryhmät

1.7.2 Toteutusympäristö

Määrityksessä käytettävät ohjelmistot

- Microsoft Office – tuotteet
- QPR ProcessGuide

Järjestelmän luomisessa käytettävät ohjelmistot

- Microsoft Office – tuotteet
- Visual Studio
- Autohotkey
- QPR API
- Ontime

Testausympäristö

- Yksikkö, integraatio sekä järjestelmätestauksessa käytetään järjestelmän kehitysympäristöä
- Hyväksymistestaus suoritetaan kehittäjän työympäristössä

QPR Software Oyj on antanut projektin käyttöön tarvittavat ohjelmistot sekä koneet.

1.8 Projektin budjetti

Projekti ei vaadi erillistä budjettia. Hankintoja ei tarvitse tehdä.

1.8.1 Työn kustannukset

Työn kustannukset pitävät sisällään QPR:n työntekijöiden työajalla tekemän työn palkkakustannukset. Muita kustannuksia ei projekti vaadi

1.8.2 Hankinnat ja muut kustannukset

Projekti toteuttamiseen tarvittavat työvälineet ja tilat on jo olemassa. Uusia hankintoja ei tarvita.

1.9 Projektin aikataulu

Projekti koostuu määrityksestä, toteuttamisesta, seurannasta ja raportoinnista. Katso Liite 3 tarkemmasta aikataulutuksesta. Projekti alkaa 1.6 ja loppuu 1.10.

2 Työsuunnitelma

2.1 Vaiheistus

Projekti koostuu järjestelmän määrittelystä sekä toteutuksesta. Ohessa kuvattu ohjauspisteet projektille.

Taulukko 1. Projektin ohjauspisteet ja ohjauskokousaikataulu

Ohjauspiste	Tavoite kokoukselle	kokouspvm
1.	Määrittelykuvaston hyväksyminen Työsuunnitelman hyväksyminen	26.6
2.	Järjestelmän hyväksyminen	
Päättökokous	Loppuraportin hyväksyminen Projektin päättäminen	

2.2 Tehtävät, työmäärät ja lopputulokset

Jokaiselle tehtävälle on määritelty aloituskriteeri ja lopputulos, jotta työn aikataulussa pysyminen voidaan todentaa. Työmäärät on määritelty tunteina. Katso Liite 3.

2.3 Ajoitus

Projekti tehdään 1.6 – 1.10 välisenä aikana. Tästä ajasta tarkoitus on käyttää kuukausi seurantaan/ylläpitoon, jotta saadaan aikaiseksi vertailukohta ohjelman hyödyllisyydestä.

Liitteessa 3 on kuvattu tarkemmin ajoitus.

2.4 Työmenetelmät ja standardit

Projektin työmenetelmissä käytetään perinteistä vesiputous - mallia.

Testausdokumentaatiossa käytetään IEEE 829 Standard for Software Test Documentation – standardia. Testauksessa käytetään ISO/IEC 9126-1:2001, Software Product Quality – standardia. Standardit noudattavat ISEB (Information Systems Examination Board) suosituksia.

2.5 Projektihallinnolliset menettelytavat

Ohjauskäytännöt

- Projektin pääsääntöinen ohjaus suoritetaan erillisissä ohjauskokouksissa. Ohjaus voi tapahtua myös esimerkiksi puhelimen ja sähköpostin välityksellä

Raportointikäytännöt

- Haaga-Helian standardien mukaisesti

Tiedottamiskäytännöt

- Tiedottaminen tapahtuu sähköpostin ja puhelimen välityksellä. QPR sisäinen tiedottaminen tapahtuu QPR:n ohjeiden mukaisesti

Muutoshallinta

- Mahdollisista muutoksista on välittömästi tiedotettava ohjausryhmää ja mahdollisesti kutsua kokoon ohjausryhmän kokous

3 Laatusuunnitelma

3.1 Laadulliset tavoitteet

Projektin tuloksen laadullisina tavoitteina on se, että se täyttää asiakkaan vaatimukset luotettavuuden sekä käytettävyyden osalta. Ohjelmiston on myös oltava yhdenmukainen tuotettujen dokumenttien kanssa. Käyttöohjeet seuraavat QPR:n dokumenttien raportointi – käytäntöä. Muut dokumentit noudattavat Haaga-Helian raportointi käytäntöjä.

Projektin onnistumisen kannalta on ehdottoman tärkeää, että projekti pysyy aikataulussa. Projektin on tuotettava sovitut tulokset sovitussa aikataulussa sekä dokumentaatiot ohjauskokouksiin, jossa voidaan todentaa aikataulun pitävyys.

3.2 Laadunvarmistusmenettelyt ja vastuut

Laadunvarmistusmenettelyinä käytetään dokumenttien katselmointia ja ohjelmiston testausta. Katselmoinnit ja testaukset on esitetty ajoitussuunnitelmassa (liite 3) ja tarkennettu testaussuunnitelmassa. Kustakin katselmointi – ja systeemitestaustilaisuudesta laaditaan pöytäkirja ja mahdolliset virheraportit.

Kaikki projektin tuottamat luovutettavat dokumentit katselmoidaan. Katselmointien ja testausten avulla löytyneet virheet, epäjohdonmukaisuudet sekä muut puutteet, jotka eivät vastaa edellistä hyväksyttyä tulosta, korjataan.

Käytettyä työmäärää seurataan suhteessa suunniteltuun. Ylitykset ja alitukset selvitetään edistymisen seurannan yhteydessä.

3.3 Dokumentointi – ja versionhallintamenettelyt

Varsinaisen opinnäytetyön raportointi perustuu 1.9.2007 annettuun raportointiohjeeseen. Kaikkien dokumenttien versiointi perustuu liitteessä 1 annettuihin ohjeisiin. Myös tuotetun ohjelmiston versiointi perustuu samaan liitteessä 1 olevaan ohjeeseen.

Liite 1. Dokumenttien versiointiohje

Kaikki projektin tuottamat kirjalliset dokumentit, ohjelmisto- ja testitiedostot versioidaan tämän versionhallintaohjeen mukaisesti, ellei aloituskokouksessa ole muunlaisesta käytännöstä sovittu. Versionhallintamerkinnät tarkistetaan ao. tuloksen katselmuksen tai muun tarkistuksen yhteydessä tarkistuspisteessä.

Jokaisen kirjallisen dokumentin kansilehdelle laitetaan versiomerkinnät. Kursiivilla on oheisessa esimerkissä merkitty muuttuvat osat. Dokumentti tarkastetaan vaiheen lopussa olevassa katselmuksessa ja hyväksytään sitä seuraavassa johtoryhmäkokouksessa.

Versio:	*iso.pieni*	*luonnos/ehdotus/hyväksytty*
Tekijä:	*nimi*	Pvm: *pp.kk.vvvv*
Tarkastanut:	*nimi*	Pvm: *pp.kk.vvvv*
Hyväksynyt:	*nimi*	Pvm: *pp.kk.vvvv*

iso.pieni	Kun dokumenttia tehdään ensimmäistä kertaa, niin sille annetaan versionumero 0.1. Kun dokumentti tai tuote on hyväksytty katselmuksessa, niin sen versionumeroksi tulee 1.0. Jos muutosmenettelyn kautta jo kertaalleen hyväksyttyä dokumenttia tai tuotteen osaa muutetaan, niin muutoksen aikana sen versionumero on 1.1. Kun ko. osa hyväksytään, niin versionumeroksi tulee 2.0. Versionumeron ensimmäinen numero siis ilmaisee, montako kertaa ko. osa on hyväksytty.
luonnos	Dokumenttia tehdään vielä
ehdotus	Dokumentti on katselmoitu
hyväksytty	Dokumentti on hyväksytty johtoryhmäkokouksessa
nimi	Kuka on tehnyt, tarkastanut (katselmuskokouksen siht.), hyväksynyt (johtoryhm siht..) Tarkastusmerkinnät tehdään katselmuksen jälkeen Hyväksymismerkinnät johtoryhmäkokouksen jälkeen
pvm	Tapahtumapäivämäärä

Kunkin versioitavan dokumentin osalta tulee ylläpitää versiohistoriaa. Versiohistoria sijoitetaan kansilehden ja sisällysluettelon väliin. Versionumeroinnin muuttuessa kirjataan versiohistoriaan, mikä taho päätti muutoksesta, minkä tyyppisestä muutoksesta on kyse ja mitä muutettiin (kohta, joka muuttui). Lisäksi kirjataan päivämäärä, jolloin muutos dokumenttiin tehtiin ja muutoksen kirjaaja.

Liite 2. Riskien kartoitus ja analysointi

*) P = pieni, K = kohtuullinen, S = suuri/huomattava.

Riski	**Todennäköisyys *)**	**Seurausvaikutus *)**	**Syyt**	**Syiden ennaltaehkäisy, suojaustoimenpiteet**	**Seurauksiin varautuminen, varmistamistoimenpiteet**
Projektin jäsenten sairastuminen	K	S	Sairastuminen	Vältellä sairastumiseen johtavia tilanteita	Ei ole
Mahdolliset virheet käytettävässä API -rajapinnassa	K	P/K/S	Laajassa ohjelmistossa	Vanhojen testaus – skriptien ylläpitö / tarkistus	Mahdolliset workaroundit, vikojen korjaaminen niiden selvitessä
Aikataulun pettäminen	P	S	Liialliset kiireet, muut tekijät (kuten virheet ohjelmistossa)	Aikataulussa pysyminen alusta alkaen. Mahdollisten virheiden korjaaminen alusta nopeasti	Ohjauskokouksissa läpikäynti

Työkalusta liian staattinen	P/K	K	Ei ole otettu huomioon jatkokehityksen tarvetta	Otetaan tarpeet huomioon alusta alkaen	Katselmoinnit
Ylläpidettävyys	P	S	Järjestelmästä luodaan liian monimutkainen ja dokumentaatio on puutteellinen	Pidetään järjestelmä mahdollisimman yksinkertaisena ja dokumentoidaan toiminnallisuus huolellisesti	Katselmoinnit

Liite 3. Projektin tehtävät, työmäärät ja ajoitus

teht. no.	tehtävä	lopputulos	aloituskriteeri	tuntia	vastuu	Viikko																			
						22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	
				405																					
1	**Projektin käynnistys**	hyväksytty projektisuunnitelma	toimeksianto	**15**	OR																				
1.1	Aloituskokouksen valmistelu	esityslista ja muu materiaali jaettu		10																					
1.2	Aloituskokous	käynnistetty projekti	p-suunnitelma valmis	2																					
1.3	Pöytäkirjan laatiminen	pöytäkirja jaettu		3																					
2	**Toiminnan määrityksen tarkistus**	kehittämisen lähtökohta sovittu	projekti on käynnistetty	**5**																					
2.1	Toiminnallisen määrit. tarkistus	muutoslista esitetty		3																					
2.2	Tavoitteen jäsentäminen	oma tehtäväkokonaisuus on selvitetty		2																					

3	**Järjestelmän määritys**	määrityskuvasto	lähtökohta selvillä	**45**																				
3.1	Käyttötapausten määritys	atk-systeemin yleiskuvaus, käyttötapaukset		10																				
3.2	Säilytettävien tietojen määritys	luokkamalli, tilakaaviot		5																				
3.3	Käyttötilanteiden ja niiden tietotarpeiden määritys	käyttötapaus- ja käyttötilanne- sekä luokkakuvaukset		5																				
3.4	Määrityskuvaston viimeistely	ohjaustiedot, käsittelysäännöt		10																				
3.5	Katselmointi	korjattu määrityskuvasto	valmis määrityskuvasto	15																				
4	**Suunnittelu**	suunnittelukuvasto	hyväksytty määrityskuvasto	**80**																				
4.1	Testauksen suunnittelu	testaussuunnitelma		10																				
4.2	Tietokannan suunnittelu	luokat		10																				
4.3	Käyttöliittymän suunnittelu	näytöt, navigointi		15																				

4.4	Paketoinnin suunnittelu	paketit, ohjaustiedot		10																					
4.5	Käytön suunnittelu	viestiyhteyskaaviot		10																					
4.6	Katselmointi	testauspöytäkirja	valmis suunnittelukuvasto	20																					
5	**Toteutus**	valmis ohjelmisto	suunnittelukuvasto	**160**																					
5.1	Tietokannan toteutus			10																					
5.2	Käyttöliittymän toteutus			10																					
5.3	Sovellusohjelmointi			100																					
5.4	Testaus	testauspöytäkirja		40																					
6	**Systeemitestaus**	testattu, dokumentoitu ohjelmisto	valmis atk-järjestelmä	**35**																					
6.1	Testauksen suunnittelu	testaussuunnitelma, testiaineisto		5																					
6.2	Testaus	testauspöytäkirja, -raportti	asennettu atk-järjestemä	20																					
6.3	Korjaaminen	korjattu atk-järjestelmä		10																					

7	**Projektin ohjaus**	hallittu edistyminen	projekti on käynnistetty	**25**	PP																			
7.1	Edistymisen raportointi ja seuranta	edistymisraportti	seurantajakso umpeutuu	10																				
7.2	Ohjausryhmän kokoukset	esityslista, materiaali jaettu	sovittu kokousaika	10																				
7.3	Pöytäkirjan laatiminen	pöytäkirja	kokous on pidetty	5																				
8	**Projektin päättäminen**	hyväksytty tulos ja päätetty projekti	projektin tulos on valmis	**40**	OR																			
8.1	Loppuraportin laatiminen	projektin loppuraportti		20																				
8.2	Päättökokouksen valmistelu	esityslista, materiaali jaettu		5																				
8.3	Päättökokous	päätetty projekti		10																				
8.4	Pöytäkirjan laatiminen	pöytäkirja		5																				