Appendix A: Water quality data for the SALCA Regions

Parameter	Unit	SALCA Region 1		SALCA Region 2		SALCA Region 3		SALCA Region 4	
		Average	Maximum	Average	Maximum	Average	Maximum	Average	Maximum
Organic carbon	mg/l	20.489	427.200	4.794	63.680	5.588	385.100	6.667	139.460
Berellium	mg/l	0.001	0.010	0.002	0.061	0.001	0.029	0.003	0.153
Vanadium	mg/l	0.006	0.097	0.003	0.067	0.004	0.166	0.009	0.466
Chromium	mg/l	0.010	0.203	0.004	0.093	0.009	0.423	0.012	3.017
Manganese	mg/l	0.073	10.780	0.048	3.564	0.137	17.880	0.557	149.260
Cobalt	mg/l	0.026	0.169	0.012	0.092	0.020	0.495	0.025	0.807
Nickel	mg/l	0.018	0.150	0.008	0.045	0.017	0.772	0.048	6.220
Copper	mg/l	0.031	1.319	0.002	0.015	0.004	0.098	0.015	1.365
Zinc	mg/l	0.041	0.837	0.010	0.438	0.021	1.148	0.093	8.730
Arsenic	mg/l	0.050	0.050	0.025	0.050	0.069	0.653	0.071	0.100
Zirconium	mg/l	0.022	0.174	0.016	0.772	0.018	1.420	0.029	1.087
Molybdenum	mg/l	0.012	0.291	0.005	0.099	0.009	0.265	0.012	0.742
Cadmium	mg/l	0.004	0.017	0.001	0.002	0.005	0.088	0.004	0.043
Barium	mg/l	0.018	0.151	0.029	0.254	0.039	0.670	0.041	2.510
Mercury	mg/l	0.002	0.026	0.001	0.002	0.011	0.037	0.007	0.100
Lead	mg/l	0.053	2.173	0.018	0.144	0.059	3.371	0.033	0.412
Elec. conductivity	mg/l	260.176	8620.000	54.212	3150.000	37.054	2170.000	58.194	2040.000
TSS	mg/l	1771.418	107480.00	388.092	22508.000	271.357	4319.000	387.976	8347.000
Hydrogen ions	mg/l	0.024	181.970	0.009	8709.636	0.010	42657.952	0.006	46773.514
Sodium	mg/l	491.233	32686.700	76.606	6948.900	23.501	387.000	40.694	2309.400
Magnesium	mg/l	69.212	5493.900	15.289	850.900	16.398	326.500	21.006	383.000
Calcium	mg/l	48.358	1058.400	19.679	470.200	25.342	338.400	38.961	726.800
Iron	mg/l	0.291	3.160	0.248	3.520	0.268	4.470	0.239	31.050
Chloride	mg/l	861.488	59370.400	92.250	12417.500	21.874	771.400	41.545	4273.300
Nitrates	mg/l	0.312	40.907	0.457	54.578	0.856	716.103	0.876	795.987
Sulphates	mg/l	174.819	8935.400	42.435	5248.200	58.065	2088.400	81.854	3216.000
Phosphates	mg/l	0.082	28.730	0.055	6.845	0.097	18.613	0.186	21.700
Total alkalinity	mg/l	112.242	936.500	112.387	1067.000	97.692	469.900	132.643	1719.000
Silicone	mg/l	3.094	26.490	6.902	19.340	6.087	32.320	5.540	80.570
Potassium	mg/l	13.956	1308.340	2.721	223.830	3.399	96.900	4.405	318.400
Ammonium	mg/l	0.104	54.904	0.092	71.253	0.168	25.318	0.223	48.000
Kjeldahl nitrogen	mg/l	9.029	17.508	0.896	45.914	0.612	37.058	1.315	55.666
Total phosphorus	mg/l	21.925	43.810	0.135	5.950	0.051	2.910	0.396	25.540

Appendix B: South African land cover data

Land cover type [ha]	SALCA Region 1	SALCA Region 2	SALCA Region 3	SALCA Region 4
Barren rock	49496.76	20303.31	8883.67	192754.39
Forest and Woodland	93847.34	1155677.17	6136116.78	282597.78
Herbland	5005.99	73.32	6260.17	239380.94
Shrubland and low Fynbos	16581459.57	1743868.95	4996264.46	22737784.38
Thicket & bushland (etc.)	1757850.57	3440923.52	2381485.83	9625435.37
Unimproved grassland	399533.99	7814047.44	1917463.78	16731606.08
Total: natural or semi-natural	**18887194.22**	**14174893.71**	**15446474.69**	**49809558.94**
Cultivated: permanent-commercial dryland	50458.81	21977.04	52770.16	927.25
Cultivated: permanent-commercial irrigated	258116.03	8176.18	79055.88	34929
Cultivated: permanent-commercial sugarcane	0	462979.01	46273.42	15.54
Cultivated: temporary-commercial dryland	2003585.32	366981.38	1497345.65	5897440.78
Cultivated: temporary-commercial irrigated	239020.45	228974.34	370207.63	248167.29
Cultivated: temporary-semi-commercial/subsistence dryland	484	1916307.87	1044123.09	961994.25
Degraded: Herbland	63.52	75.12	0	0
Degraded: forest and woodland	0	158242.99	901579.81	395.24
Degraded: forest and woodland	426929.48	54748.33	36.19	81393.87
Degraded: thicket & bushland(etc)	29699.15	375510.49	695230.94	1204496.11
Degraded: unimproved grassland	138.54	1594204.1	44743.05	1127543.54
Dongas & sheet erosion scars	107229.11	45743.43	10544.41	24315.77
Forest plantations	142250.6	1119188.05	588491.37	62446.35
Improved grassland	61302.08	36536.16	25438.9	17369.41
Mines & quarries	1023207.59	9023.2	51628.97	78535.77
Urban / built-up land: commercial	5455.7	5176.22	8946.53	15670.69
Urban / built-up land: industrial / transport	18171.98	12507.88	218906.31	30287
Urban / built-up land: residential	105281.15	341314.13	186844.3	251008.58
Urban / built-up land: residential (small holdings: bushland)	943.63	15665.99	57658.27	4764.11
Urban / built-up land: residential (small holdings: grassland)	43.07	2867.03	4245.23	70132.43
Urban / built-up land: residential (small holdings: shrubland)	11248.1	296.66	40161.92	717.59
Urban / built-up land: residential (small holdings: woodland)	17089.62	43.38	14784.88	198.13
Total: transformed land	**4500717.93**	**6776539.01**	**5939016.91**	**10112748.7**
Water bodies	60875.63	135617.04	31901.45	189803.6
Wetlands	23028.84	94365.87	3821.39	458122.12
Total: water-covered land	**83904.47**	**229982.91**	**35722.84**	**647925.72**

Appendix C: South African conserved vegetation data

Vegetation type [ha]	SALCA Region 1	SALCA Region 2	SALCA Region 3	SALCA Region 4
Forest and woodland	**52475.46**	**47316.23**	**26141.13**	**310.26**
Afromontane Forest	49641.66	25191.82	26141.13	310.26
Coastal Forest	2833.80	6361.56		
Sand Forest		15762.85		
Herbland	**29187.98**	**0.00**	**0.00**	**2312.02**
Bushmanland	18380.33			2312.02
West Coast Renosterveld	10807.65			
Shrubland and low Fynbos	**859498.92**	**378623.95**	**261579.67**	**948508.07**
Central Lower Karoo	0.00			
Central Mountain Renosterveld	27627.19			
Coastal Bushveld / Grassland		166681.24		
Eastern Mixed Nama Karoo	35137.48	29044.75		58106.05
Escarpment Mountain Renosterveld				269.34
Great Nama Karoo	792.69			
Kalahari Mountain Bushveld	3122.01			387.17
Kalahari Plains Thorn Bushveld			438.21	24067.89
Kalahari Plateau Bushveld				0.00
Karroid Kalahari Bushveld				2423.77
Little Succulent Karoo	20862.76			
Lowland Succulent Karoo	23965.40	0.00		1903.52
Mopane Shrubveld			261141.45	
Mountain Fynbos	603749.63			
Natal Lowveld Bushveld		182897.96		
North-western Mountain Renosterveld	0.00			
Orange River Nama Karoo				79124.85
Sand Plain Fynbos	5526.06			
Shrubby Kalahari Dune Bushveld				728151.59
South and South-west Coast Renosterveld	20080.76			
Strandveld Succulent Karoo	1519.15			0.00
Upland Succulent Karoo	117094.62			52998.53
Upper Nama Karoo	21.16			1075.35
Thicket & bushland (etc.)	**252780.86**	**533060.73**	**2461482.41**	**320497.42**
Clay Thorn Bushveld			15088.10	
Coast-Hinterland Bushveld		36181.81		
Dune Thicket	51470.05	1587.05		
Eastern Thorn Bushveld	798.23	3418.58		

Vegetation type [ha]	SALCA Region 1	SALCA Region 2	SALCA Region 3	SALCA Region 4
Grassy Fynbos	101636.99	4107.73		
Kimberly Thorn Bushveld				84440.19
Laterite Fynbos	289.62			
Lebombo Arid Mountain Bushveld		51935.13	162233.02	
Limestone Fynbos	29616.22			
Mesic succulent Thicket	10346.28			
Mixed Bushveld			217940.89	8506.34
Mixed Lowveld Bushveld		159598.12	500922.16	
Mopane Bushveld			776299.43	
Natal Central Bushveld		26733.66		
Sour Lowveld Bushveld		33908.37	220710.53	
Soutpansberg Arid Mountain Bushveld			60282.32	
Spekboom Succulent Thicket	8812.89			
Subarid Thorn Bushveld		1670.68		
Subhumid Lowveld Bushveld		29216.39		
Sweet Bushveld			38643.97	
Sweet Lowveld Bushveld		127419.65	363757.20	
Thorny Kalahari Dune Bushveld				227550.88
Valley Thicket	0.00	39742.55		
Waterberg Moist Mountain Bushveld			105604.78	
Xeric Succulent Thicket	49810.58	17541.01		
Unimproved grassland	**631.67**	**455159.10**	**211648.21**	**127602.58**
Alti Mountain Grassland		131796.83		9545.91
Afro Mountain Grassland				
Coastal Grassland	243.10	3046.81		
Dry Clay Highveld Grassland				0.00
Dry Sandy HighveldGrassland				18203.76
Moist Clay Highveld Grassland				0.00
Moist Cold Highveld Grassland				14239.04
Moist Cool Highveld Grassland			3361.88	8138.25
Moist Sandy Highveld Grassland		1853.83	6685.09	1839.93
Moist Upland Grassland		110473.67		137.08
North-eastern Mountain Grassland		180511.74	189164.37	3.03
Rocky Highveld Grassland			12436.87	20795.60
Short Mistbelt Grassland		11390.38		
South-eastern Mountain Grassland	388.58	3785.11		3083.58
Wet Cold Highveld Grassland		12300.72		51616.39
Total conserved	**1194574.90**	**1414160.01**	**2960851.42**	**1399230.34**

Appendix D: Current and target values for the RII calculations

Midpoint category and resource group impact	Measurement units	Ambient annual values	SALCA Region 1	SALCA Region 2	SALCA Region 3	SALCA Region 4
Water use (ground and surface water reserves) (WU – water resources)	kg of available reserves[a]	**Current [t]**	1694×10^6	6598×10^6	2407×10^6	2562×10^6
		Target [t]	18×10^6	1123×10^6	1184×10^6	550×10^6
Eutrophication potential (EP – water resources)	kg PO_4^{3-} equivalence	**Current [t]**	462.5	1346.2	740.0	1560.6
		Target [t]	69.4	201.9	111.0	117.1
Acidification potential (AP – air resources)	kg SO_2 equivalence	**Current [kg]**	306.7	560.2	636.7	573.2
		Target [kg]	233.5	550.7	646.4	521.7
Acidification potential (AP – water resources)	kg H_2SO_4 equivalence	**Current [kg]**	5692.6	5239.7	3626.0	2412.5
		Target [kg]	7166.5	21108.4	11603.2	12235.1
Acidification potential (AP – land resources)	kg H_2SO_4 equivalence	**Current [kg]**	5692.6	5239.7	3626.0	2412.5
		Target [kg]	7166.5	21108.4	11603.2	12235.1
Ozone creation potential (OCP – air resources)	kg O_3 formed	**Current [kg]**	466.2	1064.3	1209.7	1089.2
		Target [kg]	1167.3	2753.5	3232.1	2608.5
Ozone depletion potential (ODP – air resources)	kg CFC-11 equivalence	**Current [t]**	3754.8	3377.7	3405.0	9659.7
		Target [t]	2346.8	2117.9	2135.1	6057.0
Global warming potential (GWP – air resources)	kg CO_2 equivalence	**Current [Mt]**	1668.7	1505.9	1518.0	4306.5
		Target [Mt]	1600.5	1444.4	1456.1	4130.9
Human toxicity potential (HTP – air resources)	kg Pb equivalence	**Current [t]**	3.7	9.8	9.2	8.3
		Target [t]	2.9	6.9	8.1	6.5
Human toxicity potential (HTP – water resources)	kg Pb equivalence	**Current [kg]**	245125	242316	436600	257499
		Target [kg]	925	2692.4	1480	1560.6
Human toxicity potential (HTP – land resources)	kg Pb equivalence	**Current [t]**	5750	5189	5231	14840
		Target [t]	2168	1957	1973	5597

Midpoint category and resource group impact	Measurement units	Ambient annual values	SALCA Region 1	SALCA Region 2	SALCA Region 3	SALCA Region 4
Aquatic toxicity potential (ATP – water resources)	kg Pb equivalence	**Current [kg]**	245125	242316	436600	257499
		Target [kg]	925	2692.4	1480	1560.6
Terrestrial toxicity potential (TTP – land resources)	kg Pb equivalence	**Current [t]**	5750	5189	5231	14840
		Target [t]	2168	1957	1973	5597
Occupied land use (OLU – land resources)	m^2.a near-natural[a]	**Current [ha]**	1.997×10^7	1.494×10^7	1.556×10^7	5.062×10^7
		Target [ha]	2.015×10^7	1.500×10^7	1.518×10^7	5.152×10^7
Transformed land use (TLU – land resources)	m^2 non-natural[a]	**Current [ha]**	3.453×10^6	6.229×10^6	5.889×10^6	9.852×10^6
		Target [ha]	3.279×10^6	6.170×10^6	6.270×10^6	8.953×10^6
Mineral depletion (MD – mined resources)	kg Pt equivalence[a]	**Current [Mt]**	35529	35529	35529	35529
		Target [Mt]	16025	16025	16025	16025
Energy depletion (ED – mined resources)	kg coal equivalence[a]	**Current [Mt]**	51813	51813	51813	51813
		Target [Mt]	24171	24171	24171	24171

a The measurement units for resource use midpoint categories are per person, e.g. kg of available water reserves per person, and therefore corrected on the population in the specific region. The following population numbers are used:

SALCA Region 1 - 7528298.0
SALCA Region 2 - 11568283.5
SALCA Region 3 - 18433327.0
SALCA Region 4 - 3053664.5
South Africa - 40583573.0

Appendix E: Landfill site calculations for average waste treatment

Estimation of land usage requirements

MRD for an average medium South African landfill site	=	325	t/day
Mass of waste treated annually	=	84500000	kg/year
Approximate operating life	=	20	years
Approximate annual growth	=	2	%
IRD for an average medium South African landfill site	=	218.7	t/day
Average density of waste (general landfill sites)	=	0.975	t/m^3
Landfill site airspace required (260 days operating)	=	72905.2	m^3
Typical thickness (general landfill sites)	=	2.5	m
Area required (general landfill sites)	=	29162.1	m^2
	=	2.9	ha

Estimation of underground water affected by typical landfill site operations

Assumed ratio of substances (underground water body)	=	0.66	ha/kt
	=	1.52	kt/ha
Weight of underground water body	=	4.419	kt/year
	=	5.229×10^{-8}	kt/kg waste

Source: **Minimum requirements for the handling, classification and disposal of hazardous waste, South African Department of Water Affairs and Forestry, Second Edition, Pretoria, 1998**

Appendix F: Life Cycle Inventories of key process parameters

Electricity usage (per MJ)

Substance	Compartment	Unit	Value
water (drinking, for process.)	raw	g	7.5200E+02
baryte	raw	mg	1.2400E+01
aluminium (Al)	raw	mg	2.1700E+01
bentonite	raw	mg	1.7000E+01
chromium (Cr)	raw	µg	1.1000E+03
coal (South African)	raw	g	1.2042E+02
cobalt (Co)	raw	ng	2.3100E+00
copper (Cu)	raw	mg	7.2700E+00
oil crude	raw	g	2.5791E+00
energy from hydro power	raw	kJ	1.6900E+01
energy from uranium	raw	J	5.6700E+01
natural gas	raw	kg	9.6432E+00
iron (Fe)	raw	mg	7.1700E+02
lead (Pb)	raw	µg	7.3900E+01
coal (soft, lignite)	raw	g	2.0640E+00
manganese (Mn)	raw	µg	6.6900E+02
marl	raw	g	3.4100E+00
methane	raw	g	1.0200E+00
molybdene (Mo)	raw	pg	9.1400E+02
nickel (Ni)	raw	µg	4.6600E+02
palladium (Pd)	raw	pg	2.6200E+01
platinum (Pt)	raw	pg	6.0200E+01
rhenium (Re)	raw	pg	1.3800E+01
rhodium (Rh)	raw	pg	2.0200E+01
rock salt	raw	mg	4.9400E+01
silver (Ag)	raw	µg	7.3400E+00
tin (Sn)	raw	µg	4.0800E+00
uranium (U)	raw	mg	1.3649E+01
wood	raw	g	1.5300E+00
zeolite	raw	µg	1.3700E+02
zinc (Zn)	raw	µg	5.6700E+01
1,2-dichloroethane	air	µg	2.7400E+00

Substance	Compartment	Unit	Value
acetaldehyde	air	µg	5.1600E+00
acetic acid	air	µg	2.3200E+01
acetone	air	µg	5.1000E+00
acrolein	air	ng	4.8400E+01
aluminium (Al)	air	mg	1.4700E+01
aldehydes	air	ng	5.8300E+02
alkanes	air	mg	1.0400E+00
alkenes	air	µg	9.2200E+02
ammonia (NH_3)	air	mg	2.4400E+00
arsenic (As)	air	µg	3.0500E+01
boron (B)	air	mg	1.3400E+00
barium (Ba)	air	µg	1.8100E+02
beryllium (Be)	air	µg	1.8400E+00
benzaldehyde	air	ng	1.6700E+01
benzene	air	µg	5.6800E+01
benzo(a)pyrene	air	ng	4.6700E+01
bromide (Br)	air	µg	5.8100E+02
butane	air	µg	4.2300E+02
butene	air	µg	7.4800E+00
calcium (Ca)	air	mg	2.2300E+00
cadmium (Cd(II))	air	µg	2.3300E+00
CFC-116	air	µg	1.5000E+00
CFC-14	air	µg	1.2000E+01
carbon monoxide (CO)	air	mg	5.8400E+01
carbon dioxide (CO_2)	air	g	2.1200E+02
cobalt (Co)	air	µg	8.7200E+00
chromium (Cr(III))	air	µg	7.2000E+01
copper (Cu(II))	air	µg	1.0600E+02
C_xH_y	air	µg	3.8100E+02
C_xH_y (aromatic)	air	µg	1.1200E+00
cyanide	air	ng	4.4300E+01
dioxins (unspesified)	air	pg	2.1500E+01
dust (PM10)	air	mg	8.8531E+01
ethane	air	µg	4.2300E+02
ethanol	air	µg	1.0000E+01

Substance	Compartment	Unit	Value
ethene	air	µg	8.2200E+01
ethylbenzene	air	µg	9.2200E+02
ethyne	air	µg	7.9400E+00
iron (Fe(III))	air	mg	6.2900E+00
formaldehyde	air	µg	2.8600E+02
hydrogen sulfide (H_2S)	air	µg	3.5900E+01
HALON-1301	air	ng	9.9700E+02
hydrogen chloride (HCl)	air	mg	1.2400E+02
heptane	air	µg	7.4800E+01
hexane	air	µg	1.5700E+02
hydrogen fluoride (HF)	air	mg	1.3200E+01
mercury (Hg(II))	air	µg	1.5900E+01
iodine (I)	air	µg	1.9400E+02
potassium (K)	air	mg	1.7700E+00
lanthanum (La)	air	µg	5.2500E+00
methane	air	g	1.7700E+00
methanol	air	µg	1.0800E+01
magnesium (Mg)	air	mg	4.7000E+00
manganese (Mn)	air	µg	5.2600E+01
molybdene (Mo)	air	µg	7.0100E+00
nitrous oxide (N_2O)	air	mg	2.7300E+00
sodium (Na)	air	mg	1.0500E+00
nickel (Ni)	air	µg	1.9900E+02
nitrogen oxides (NO_x as NO_2)	air	mg	8.8076E+02
Volatile Organic Compounds (VOC)	air	mg	4.5500E+01
phosphorous (P)	air	µg	1.6400E+02
Polycyclic Aromatic Hydrocarbons	air	µg	5.7400E+00
lead (Pb(II))	air	µg	8.2000E+01
pentane	air	mg	1.0400E+00
phenol	air	ng	3.5800E+02
propane	air	µg	5.2100E+02
propene	air	µg	9.1100E+01
propionic acid	air	ng	4.4100E+02
platinum (Pt)	air	pg	6.0800E+00
antinomy (Sb)	air	µg	3.5200E+00

Substance	Compartment	Unit	Value
scandium (Sc)	air	µg	2.2600E+00
selenium (Se)	air	µg	6.6400E+01
silicates	air	mg	2.2500E+01
tin (Sn)	air	µg	5.0400E+00
sulphur dioxide (SO_2)	air	g	1.9691E+00
soot	air	mg	1.7000E+00
strontium (Sr)	air	µg	2.2300E+02
thorium (Th)	air	µg	1.3000E+01
titanium (Ti)	air	µg	6.9200E+02
thallium (Tl)	air	ng	5.9500E+02
toluene	air	µg	5.1700E+02
uranium (U)	air	µg	5.8700E+00
vanadium (V)	air	µg	7.7400E+02
vinyl chloride	air	µg	1.5600E+00
xylene	air	mg	3.9700E+00
zinc (Zn(II))	air	µg	1.6800E+02
zirconium (Zr)	air	ng	2.5400E+01
silwer (Ag)	water	ng	1.2800E+02
aluminium (Al)	water	mg	4.0600E+02
alkanes	water	µg	2.7300E+01
alkenes	water	µg	2.4800E+00
Aromatic Organic Halogens (AOX)	water	ng	7.4300E+02
arsenic (As)	water	µg	8.2000E+02
boron (B)	water	µg	6.3600E+02
barium (Ba)	water	mg	3.2900E+01
baryte	water	mg	4.1100E+00
beryllium (Be)	water	ng	9.2800E+00
benzene	water	µg	2.7600E+01
Biological Oxygen Demand (BOD)	water	µg	6.6400E+01
calcium (Ca)	water	mg	3.6200E+02
cadmium (Cd(II))	water	µg	2.1200E+01
chlorobenzene	water	pg	6.2000E-01
chloride (Cl^-)	water	g	2.6400E+00
cobalt (Co)	water	µg	8.1100E+02
Chemical Oxygen Demand (COD)	water	mg	1.9300E+00

Substance	Compartment	Unit	Value
chromium (Cr(III))	water	mg	4.0600E+00
chromium (Cr(VI))	water	µg	1.1200E+00
caesium (Cs)	water	ng	2.0900E+02
copper (Cu(II))	water	mg	2.0300E+00
C_xH_y	water	mg	5.0100E+00
C_xH_y (aromatic)	water	µg	1.2700E+02
C_xH_y (chloro)	water	µg	1.0400E+00
cyanide	water	µg	7.7900E+00
dichloroethane	water	µg	1.3700E+00
dissolved substances	water	mg	1.7401E+02
ethylbenzene	water	µg	5.0200E+00
iron (Fe(III))	water	mg	1.2700E+02
fluoride (F^-)	water	µg	9.7600E+02
formaldehyde	water	ng	5.9800E+00
hydrogen sulfide (H_2S)	water	µg	1.2700E+00
mercury (Hg(II))	water	ng	5.7800E+02
iodine (I)	water	µg	2.0900E+01
potassium (K)	water	mg	1.2300E+02
methylene chloride	water	ng	2.3200E+02
magnesium (Mg)	water	mg	3.4400E+02
manganese (Mn)	water	mg	8.3600E+00
molybdene (Mo)	water	mg	1.0800E+00
nitrogen (N)	water	µg	9.0400E+02
sodium (Na)	water	mg	4.0500E+02
nickel (Ni)	water	mg	2.0500E+00
nitrate (NO_3^-)	water	mg	9.7800E+00
phosphorous (P)	water	ng	2.6000E+01
Polycyclic Aromatic Hydrocarbons	water	µg	2.7400E+00
lead (Pb(II))	water	mg	2.0700E+00
phenol	water	µg	3.9200E+01
phosphate (PO_4^{3-})	water	mg	2.4300E+01
sulphur (S)	water	µg	6.4000E+00
salt	water	mg	1.4700E+01
antinomy (Sb)	water	µg	6.7400E+00
selenium (Se)	water	mg	2.0300E+00

Substance	Compartment	Unit	Value
silicone (Si)	water	ng	9.8000E+01
tin (Sn)	water	µg	5.7300E+00
strontium (Sr)	water	mg	6.1100E+00
sulphate (SO_4^{2-})	water	g	1.8502E+00
Total Suspended Solids (TSS)	water	mg	1.8900E+01
titanium (Ti)	water	mg	2.4300E+01
Total Organic Carbon (TOC)	water	mg	2.4400E+00
toluene	water	µg	2.4900E+01
tributyltin	water	µg	9.1600E+00
trichloroethene	water	ng	2.3400E+02
vanadium (V)	water	mg	2.0400E+00
tungsten (W)	water	µg	4.6600E+00
xylene	water	µg	1.9800E+01
zinc (Zn(II))	water	mg	4.1000E+00
ash	solid	g	3.2000E+01
waste (inert)	solid	g	9.2800E+01
waste (not inert)	solid	g	5.2400E+00
heat losses to air	non material	MJ	1.7200E+00
heat losses to soil	non material	J	9.7500E+02
heat losses to water	non material	kJ	3.1100E+02
Occupy as existing severely industrialised land	non material	$mm^2.a$	3.4230E+03

Liquid fuel (diesel) usage (per kg)

LCI constituent	Medium	Unit	Value
Barite	raw	mg	10.4
aluminium (Al)	raw	mg	17
bentonite	raw	mg	8.98
chromium (Cr)	raw	µg	624
coal (South African)	raw	kg	1.86183
cobalt (Co)	raw	ng	1.93
copper (Cu)	raw	mg	6.03
oil crude	raw	g	702
energy from hydro power	raw	kJ	6.4
gas from oil production	raw	g	1.601846
iron (Fe)	raw	mg	600
lead (Pb)	raw	µg	61.2
coal (soft, lignite)	raw	g	1.44
manganese (Mn)	raw	µg	551
marl	raw	g	2.87
methane	raw	mg	865
molybdene (Mo)	raw	pg	759
natural gas	raw	g	6.19914402
nickel (Ni)	raw	µg	149
palladium (Pd)	raw	pg	22.1
platinum (Pt)	raw	pg	50.5
rhenium (Re)	raw	pg	11.6
rhodium (Rh)	raw	pg	17.1
rock salt	raw	mg	41.6
silver (Ag)	raw	µg	6.18
tin (Sn)	raw	µg	3.43
unspecified energy	raw	MJ	40.1
uranium (U)	raw	µg	98.1
water (drinking, for process.)	raw	kg	4.202
wood	raw	g	1.29
zeolite	raw	µg	115
zinc (Zn)	raw	µg	47.9
1,2-dichloroethane	air	µg	2.32
acetaldehyde	air	µg	4.35

LCI constituent	Medium	Unit	Value
acetic acid	air	µg	19.5
acetone	air	µg	4.29
acrolein	air	ng	40.8
aluminium (Al)	air	mg	12.4
aldehydes	air	mg	40
alkanes	air	µg	879
alkenes	air	µg	782
ammonia (NH_3)	air	mg	22.1
arsenic (As)	air	µg	25.8
boron (B)	air	mg	1.14
barium (Ba)	air	µg	153
beryllium (Be)	air	µg	1.56
benzaldehyde	air	ng	14.1
benzene	air	µg	47.7
benzo(a)pyrene	air	ng	39.4
bromide (Br)	air	µg	493
butane	air	µg	357
butene	air	µg	6.31
calcium (Ca)	air	mg	1.89
cadmium (Cd(II))	air	µg	1.97
CFC-116	air	µg	1.22
CFC-14	air	µg	9.74
carbon monoxide (CO)	air	mg	301
carbon dioxide (CO_2)	air	g	490
cobalt (Co)	air	µg	7.38
chromium (Cr(III))	air	µg	61.1
copper (Cu(II))	air	µg	90.1
CxHy	air	mg	60
CxHy (aromatic)	air	ng	938
cyanides	air	ng	37.3
dioxins (unspesified)	air	pg	18.2
dust (PM10)	air	mg	291.9
ethane	air	µg	354
ethanol	air	µg	8.37
ethene	air	µg	69.5

LCI constituent	Medium	Unit	Value
Ethylbenzene	air	µg	782
Ethyne	air	µg	6.73
iron (Fe(III))	air	mg	5.34
Formaldehyde	air	µg	242
hydrogen sulfide (H_2S)	air	g	4.09
HALON-1301	air	ng	842
hydrogen chloride (HCl)	air	mg	106
Heptane	air	µg	63.1
Hexane	air	µg	133
hydrogen fluoride (HF)	air	mg	11.2
mercury (Hg(II))	air	µg	13.5
iodine (I)	air	µg	165
potassium (K)	air	mg	1.51
lanthanum (La)	air	µg	4.45
Methane	air	g	1.5
Methanol	air	µg	9.12
magnesium (Mg)	air	mg	3.99
manganese (Mn)	air	µg	44.6
molybdene (Mo)	air	µg	5.94
nitrous oxide (N_2O)	air	mg	50.3
sodium (Na)	air	µg	894
nickel (Ni)	air	µg	168
nitrogen oxides (NO_x as NO_2)	air	g	2.643
Phosphorous (P)	air	µg	139
Polycyclic Aromatic Hydrocarbons	air	µg	4.86
lead (Pb(II))	air	µg	69.5
Pentane	air	µg	883
Phenol	air	ng	304
Propane	air	µg	440
Propene	air	µg	77.2
propionic acid	air	ng	360
platinum (Pt)	air	pg	5.11
antinomy (Sb)	air	µg	2.98
scandium (Sc)	air	µg	1.92
selenium (Se)	air	µg	56.4

LCI constituent	Medium	Unit	Value
Silicates	air	mg	19
tin (Sn)	air	µg	4.27
sulphur dioxide (SO_2)	air	g	5.52
strontium (Sr)	air	µg	189
thorium (Th)	air	µg	11.1
titanium (Ti)	air	µg	587
thallium (Tl)	air	ng	504
toluene	air	µg	438
uranium (U)	air	µg	4.98
vanadium (V)	air	µg	656
vinyl chloride	air	µg	1.32
Volatile Organic Compounds (VOC)	air	g	17.0383
xylene	air	mg	3.37
zinc (Zn(II))	air	µg	142
zirconium (Zr)	air	ng	21.3
silwer (Ag)	water	ng	107
aluminium (Al)	water	mg	344
alkanes	water	µg	23.1
alkenes	water	µg	2.09
Aromatic Organic Halogens (AOX)	water	ng	627
arsenic (As)	water	µg	696
boron (B)	water	µg	539
barium (Ba)	water	mg	27.9
baryte	water	mg	3.47
beryllium (Be)	water	ng	2.13
benzene	water	µg	23.3
Biological Oxygen Demand (BOD)	water	mg	6.06
calcium (Ca)	water	mg	307
cadmium (Cd(II))	water	µg	18
chlorobenzenes	water	pg	0.526
chloride (Cl^-)	water	g	2.24
cobalt (Co)	water	µg	688
Chemical Oxygen Demand (COD)	water	mg	19.6
chromium (Cr(III))	water	mg	3.45
chromium (Cr(VI))	water	ng	948

LCI constituent	Medium	Unit	Value
oil	water	mg	163
caesium (Cs)	water	ng	177
copper (Cu(II))	water	mg	1.72
CxHy	water	mg	4.23
CxHy (aromatic)	water	µg	107
CxHy (chloro)	water	ng	876
cyanide	water	µg	6.56
dichloroethane	water	µg	1.16
dissolved organics	water	µg	9.74
dissolved substances	water	g	12.8
ethylbenzene	water	µg	4.24
iron (Fe(III))	water	mg	107
fluoride (F^-)	water	µg	824
formaldehyde	water	ng	5.07
glutaraldehyde	water	ng	428
hydrogen sulfide (H_2S)	water	µg	1.07
mercury (Hg(II))	water	ng	490
iodine (I)	water	µg	17.6
potassium (K)	water	mg	104
methylene chloride	water	ng	195
magnesium (Mg)	water	mg	291
manganese (Mn)	water	mg	7.06
molybdene (Mo)	water	µg	910
nitrogen (N)	water	µg	713
sodium (Na)	water	mg	342
nickel (Ni)	water	mg	1.74
nitrate (NO_3^-)	water	mg	8.28
phosphorous (P)	water	ng	8.34
Polycyclic Aromatic Hydrocarbons	water	µg	2.32
lead (Pb(II))	water	mg	1.75
phenol	water	µg	33.1
phosphate	water	mg	20.6
sulphur (S)	water	µg	5.39
salt	water	mg	12.4
antinomy (Sb)	water	µg	5.72

LCI constituent	Medium	Unit	Value
selenium (Se)	water	mg	1.72
silicone (Si)	water	ng	82.7
tin (Sn)	water	µg	4.86
strontium (Sr)	water	mg	5.18
sulphate (SO_4^{2-})	water	g	1.54
Total Suspended Solids (TSS)	water	mg	20.8
titanium (Ti)	water	mg	20.6
Total Organic Carbon (TOC)	water	mg	2.02
toluene	water	µg	21
tributyltin	water	µg	7.77
trichloroethene	water	ng	198
vanadium (V)	water	mg	1.73
tungsten (W)	water	µg	3.95
xylene	water	µg	16.7
zinc (Zn(II))	water	mg	3.47
ash	solid	g	27.1
waste (inert)	solid	g	78.6
waste (not inert)	solid	g	13.8054
heat losses to air	non material	MJ	1.32
heat losses to soil	non material	J	797
heat losses to water	non material	kJ	270
Occupy as existing severely industrialised land	non material	$m^2.a$	0.001656426

Steam usage (per kg)

LCI constituent	Medium	Unit	Value
coal (South African)	raw	g	119
water (surface, for process.)	raw	g	423
carbon dioxide (CO_2)	air	g	295
CxHy	air	mg	4.79
dust (PM10)	air	mg	354
nitrogen oxides (NO_x as NO_2)	air	mg	2.17
sulphur dioxide (SO_2)	air	g	1.29
ash	solid	g	19.8
coal (South African)	raw	g	119

Landfill treatment of produced waste (per kg)

LCI constituent	Medium	Unit	Value
Acetaldehyde	water	µg	5.3
Acetone	water	µg	610
Acetonitrile	water	µg	100
acetylacetone	water	µg	2.9
acetyl chloride	water	µg	4.2
Acrolein	water	µg	0.008
acrylic acid	water	µg	13
Acrylonitrile	water	µg	1.4
Alcohol	water	µg	700
Aldrin	water	µg	0.0004
aluminium (Al)	water	µg	0.39
aluminium chloride	water	µg	10
aluminium phosphide	water	µg	0.1
aminodimethylbenzenes	water	µg	1
aminophenols	water	µg	0.2
ammonia (NH_3)	water	µg	0.0024
ammonium chloride	water	µg	10.9
amyl acetate	water	µg	6.5
amyl alcohol	water	µg	47
aniline	water	µg	13.4
aniline hydrochloride	water	µg	0.55
anisole	water	µg	1
antimony (Sb)	water	µg	0.07
antimony chloride	water	µg	1
arsenic (As)	water	µg	0.43
arsenic acid	water	µg	0.43
arsenic trioxide	water	µg	0.43
barium (Ba)	water	µg	7.8
barium nitrate	water	µg	0.1
benzaldehyde	water	µg	0.11
benzene	water	µg	2.2
benzidine	water	µg	0.1
benzine	water	µg	0.22
benzo(a)pyrene	water	µg	0.005

LCI constituent	Medium	Unit	Value
benzonitrile	water	µg	7.8
benzyl chloride	water	µg	0.6
bromobenzene	water	µg	0.946
bromoxynile	water	µg	0.015
brucine	water	µg	2
butanols	water	µg	430
butyl chlorides	water	µg	9.7
butyl ethers	water	µg	5.2
butyl mercaptans	water	µg	0.55
butyronitrile	water	µg	0.1
cacodylic acid	water	µg	100
cadmium (Cd(II))	water	µg	0.031
cadmium chloride	water	µg	0.031
calcium cyanide	water	µg	0.05
calcium fluoride	water	µg	1
calcium hypochlorite	water	µg	0.18
carbaryl	water	µg	0.06
carbofuran	water	µg	0.028
carbon disulphide	water	µg	13.5
carbon tetrachloride	water	µg	0.1
caustic soda	water	µg	3.3
chlordane	water	µg	0.007
chlordimeform	water	µg	1.17
chloroacetic acid	water	µg	0.1
chlorobenzene	water	µg	4.5
chloroform	water	µg	0.1
chloronitroanilines	water	µg	1
chloronitrobenzene	water	µg	0.12
chlorophenols	water	µg	0.19
chlorotoluenes	water	µg	0.59
chlorpyrifos	water	µg	0.0003
chromium (Cr(III))	water	µg	4.7
chromium (Cr(VI))	water	µg	0.02
cobalt (Co)	water	µg	6.9
copper (Cu(II))	water	µg	0.1

LCI constituent	Medium	Unit	Value
copper sulphate	water	µg	0.01
cresols	water	µg	0.4
cresylic acid	water	µg	0.4
crotonaldehyde	water	µg	0.13
cyanamide	water	µg	0.1
cyanide	water	µg	0.0053
cyclohexane	water	µg	9.3
cyclohexanol	water	µg	72
cyclohexanone	water	µg	52.7
cymenes	water	µg	1
DDT	water	µg	0.0007
decanol	water	µg	2.3
decalin	water	µg	1
demeton	water	µg	0.01
diaminobenzenes	water	µg	0.574
diacetone alcohol	water	µg	42
dichlofenthion	water	µg	0.1
dichloroacetic acid	water	µg	1
dichloroanilines	water	µg	0.7
1,2-dichlorobenzene	water	µg	1.7
1,3-dichlorobenzene	water	µg	0.78
1,4-dichlorobenzene	water	µg	3.37
1,1-dichloroethane	water	µg	48
1,2-dichloroethane	water	µg	6.5
dichloromethane	water	µg	14
dichlorophenols	water	µg	0.5
dichlorotoluene	water	µg	0.58
dichlorvos	water	µg	0.09
dicrotophos	water	µg	20
dieldrin	water	µg	0.0006
diethylaminoethanol	water	µg	1
diethyl ether	water	µg	260
diethyl ketone	water	µg	154
diethyl phthalate	water	µg	3
dimetan	water	µg	0.1

LCI constituent	Medium	Unit	Value
dimethylamine	water	µg	4
dimethyl nitrosamine	water	µg	8.8
dimethyl sulfoxide	water	µg	10
dimetilan	water	µg	0.1
dinitroaniline	water	µg	1.42
dinoseb	water	µg	0.0032
dinoseb acetate	water	µg	0.1
dinoterb	water	µg	0.00034
dinoterb acetate	water	µg	0.0039
dioxane	water	µg	670
diquat	water	µg	2.1
endosulfan	water	µg	0.0003
endrin	water	µg	0.00007
epichlorohydrin	water	µg	1.8
ethanethiol	water	µg	1
ethanolamine	water	µg	17
ethyl acetate	water	µg	21.2
ethyl acrylate	water	µg	2
ethylamine	water	µg	4
ethylbenzene	water	µg	1.2
ethyl cyanide	water	µg	0.1
ethylene chloride	water	µg	13.5
ethylenediamine	water	µg	23
ethyl formate	water	µg	1
ethylhexanol	water	µg	3.2
ethyl oxalate	water	µg	0.1
ethyl propionate	water	µg	5.6
ethyltrichlorosilane	water	µg	1
fenthion	water	µg	0.07
ferric chloride	water	µg	3
ferric nitrate	water	µg	3
fluoroanilines	water	µg	0.25
formaldehyde	water	µg	0.9
furfural	water	µg	3.2
heptachlor	water	µg	0.0007

LCI constituent	Medium	Unit	Value
heptanes	water	µg	37.5
hexachlorobenzene	water	µg	0.032
hexachlorobutadiene	water	µg	0.009
hexachloroethane	water	µg	0.14
hexachlorophene	water	µg	0.01
hexanes	water	µg	0.4
hydrogen chloride (HCl)	water	µg	1
hydrocyanic acid	water	µg	0.0057
hydrogen peroxide	water	µg	4.2
hydroquinone	water	µg	0.0097
hydroxylamine sulphate	water	µg	0.72
iodomethane	water	µg	0.1
iron (Fe(II))	water	µg	9
isolan	water	µg	0.1
isopropanol	water	µg	1040
isopropylbenzene	water	µg	5
lead (Pb(II))	water	µg	0.1
lead acetate	water	µg	40
lead arsenates	water	µg	10
lead dioxide	water	µg	0.1
lead sulphate	water	µg	0.1
lindane	water	µg	0.0027
magnesium chlorate	water	µg	10
malathion	water	µg	0.1
maleic anhydride	water	µg	15
malononitrile	water	µg	0.1
maneb	water	µg	0.18
manganese (Mn)	water	µg	0.3
manganese chloride	water	µg	0.3
manganese dioxide	water	µg	0.3
mercaptoacetic acid	water	µg	0.1
mercuric acetate	water	µg	0.0009
mercuric chloride	water	µg	0.0009
mercuric oxide	water	µg	0.0009
mercurous chloride	water	µg	0.0009

LCI constituent	Medium	Unit	Value
mercury (Hg(II))	water	µg	0.022
mesityl oxide	water	µg	54
metaldehyde	water	µg	10
methacrylic acid	water	µg	0.1
methanol	water	µg	800
methomyl	water	µg	34
methoxychlor	water	µg	0.0007
methylacrelate	water	µg	0.1
methylamine	water	µg	2
methyl benzoate	water	µg	0.461
methyl chloroacetate	water	µg	0.1
methyl ethyl ketone	water	µg	322
methylhydrazine	water	µg	0.258
methyl methacrylate	water	µg	15.9
methyl salicylate	water	µg	1
methylstyrenes	water	µg	1
methyl vinyl ketone	water	µg	1
mevinphos	water	µg	0.0034
mirex	water	µg	0.02
monochlorobenzene	water	µg	2.4
naphthalene	water	µg	0.38
naphtha	water	µg	0.2
nickel (Ni)	water	µg	1.14
nickel chloride	water	µg	1.2
nicotene	water	µg	0.4
nitrobenzene	water	µg	4.3
nitrocresols	water	µg	4.61
nitroethane	water	µg	1
nitroglycerine	water	µg	0.1
nitromethane	water	µg	0.5
nitrophenols	water	µg	4.6
nitrotoluenes	water	µg	3
nonylphenol	water	µg	0.014
octanol	water	µg	2.3
octanone	water	µg	3.59

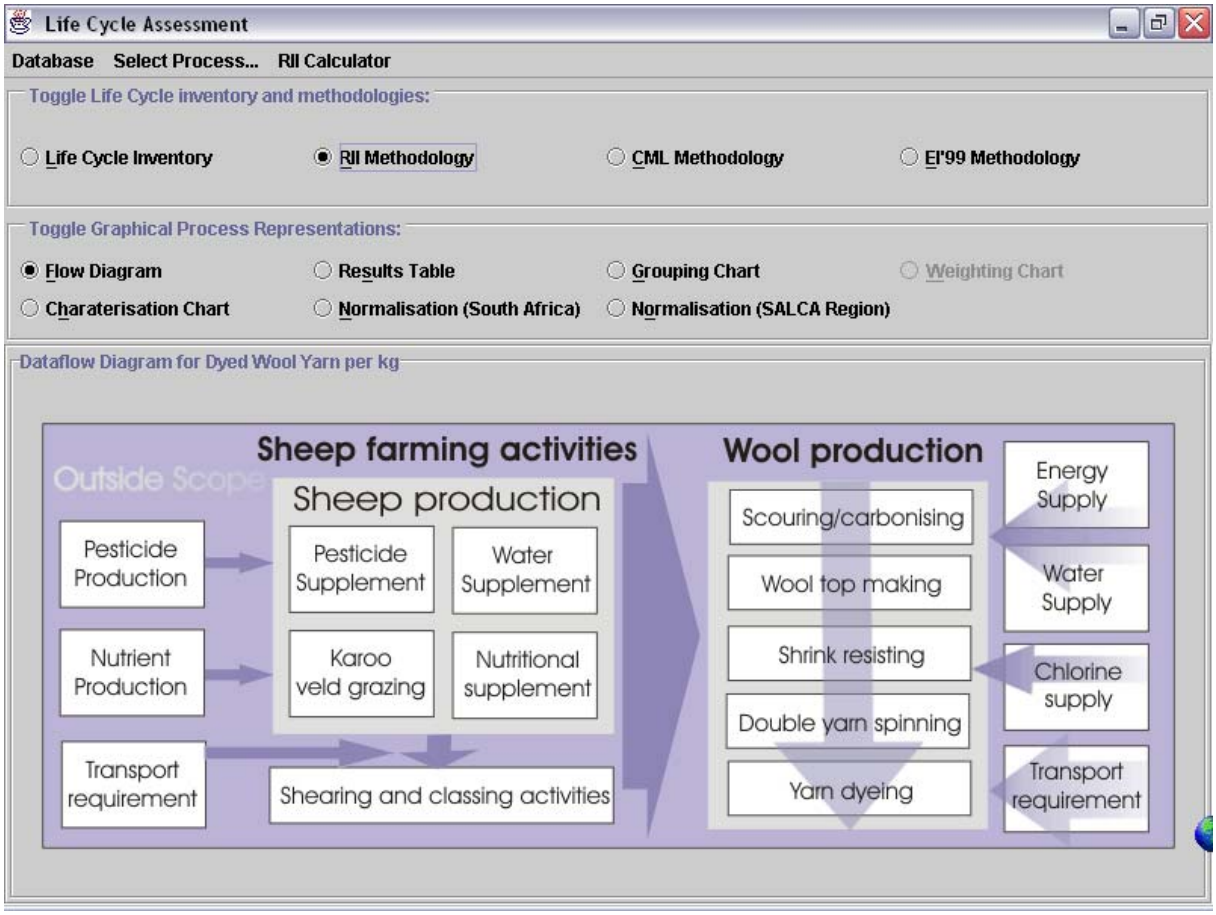
LCI constituent	Medium	Unit	Value
octanoic acid	water	µg	4.7
orthophosphoric acid	water	µg	1
osmium tetroxide	water	µg	0.1
oxirane	water	µg	9
paraformaldehyde	water	µg	3.1
paraquat	water	µg	3.2
parathion	water	µg	0.019
pentane	water	µg	10
perchloroethylene	water	µg	0.1
perchloric acid	water	µg	200
permethrin	water	µg	0.0006
phenol	water	µg	2.3
phospherous (P)	water	µg	0.1
phthalic anhydride	water	µg	1
picolines	water	µg	1
piperazine	water	µg	1
piperidine	water	µg	0.1
polychlorinated biphenyls	water	µg	0.002
potassium cyanide	water	µg	0.009
potassium dichromate	water	µg	2.5
potassium fluoride	water	µg	0.1
potassium hydrogen fluoride	water	µg	0.1
potassium hydrogen sulphate	water	µg	1
potassium hydroxide	water	µg	8
potassium nitrate	water	µg	1
potassium permanganate	water	µg	0.15
promecarb	water	µg	0.031
propanol	water	µg	320
propargyl alcohol	water	µg	0.153
propionic acid	water	µg	18.8
propoxur	water	µg	0.37
pyridine	water	µg	65
pyrrolidine	water	µg	0.5
quinoline	water	µg	1.15
resorcinol	water	µg	5.7

LCI constituent	Medium	Unit	Value
rotenone	water	µg	0.0021
saccharin	water	µg	1830
selenium (Se)	water	µg	0.26
selenium dioxide	water	µg	0.26
silver (Ag)	water	µg	2
silver nitrate	water	µg	0.0007
sodium azide	water	µg	0.5
sodium cacodylate	water	µg	1
sodium chlorate	water	µg	1
sodium cyanide	water	µg	0.0053
sodium dichromate	water	µg	1.5
sodium fluoride	water	µg	4.3
sodium fluorosilicate	water	µg	0.1
sodium hydroxide	water	µg	3.3
sodium hypochlorite	water	µg	0.007
sodium nitrate	water	µg	1
sodium nitrite	water	µg	6.4
sodium phosphate dibasic	water	µg	10
sodium phosphate tribasic	water	µg	10
strontium nitrate	water	µg	1
strychnine	water	µg	0.087
styrene	water	µg	2.5
sulphuric acid	water	µg	10
systox	water	µg	0.01
tributyltin oxide	water	µg	0.0003
terbufos	water	µg	0.0004
tetrachloroethane	water	µg	3.7
tetrachloroethylene	water	µg	0.1
tetraethyl lead	water	µg	0.002
tetrahydrofuran	water	µg	5
tetramethyl lead	water	µg	1.35
thallium nitrate	water	µg	0.1
thioglycolic acid	water	µg	0.5
thiosemicarbazide	water	µg	2.5
titanium (Ti)	water	µg	0.73

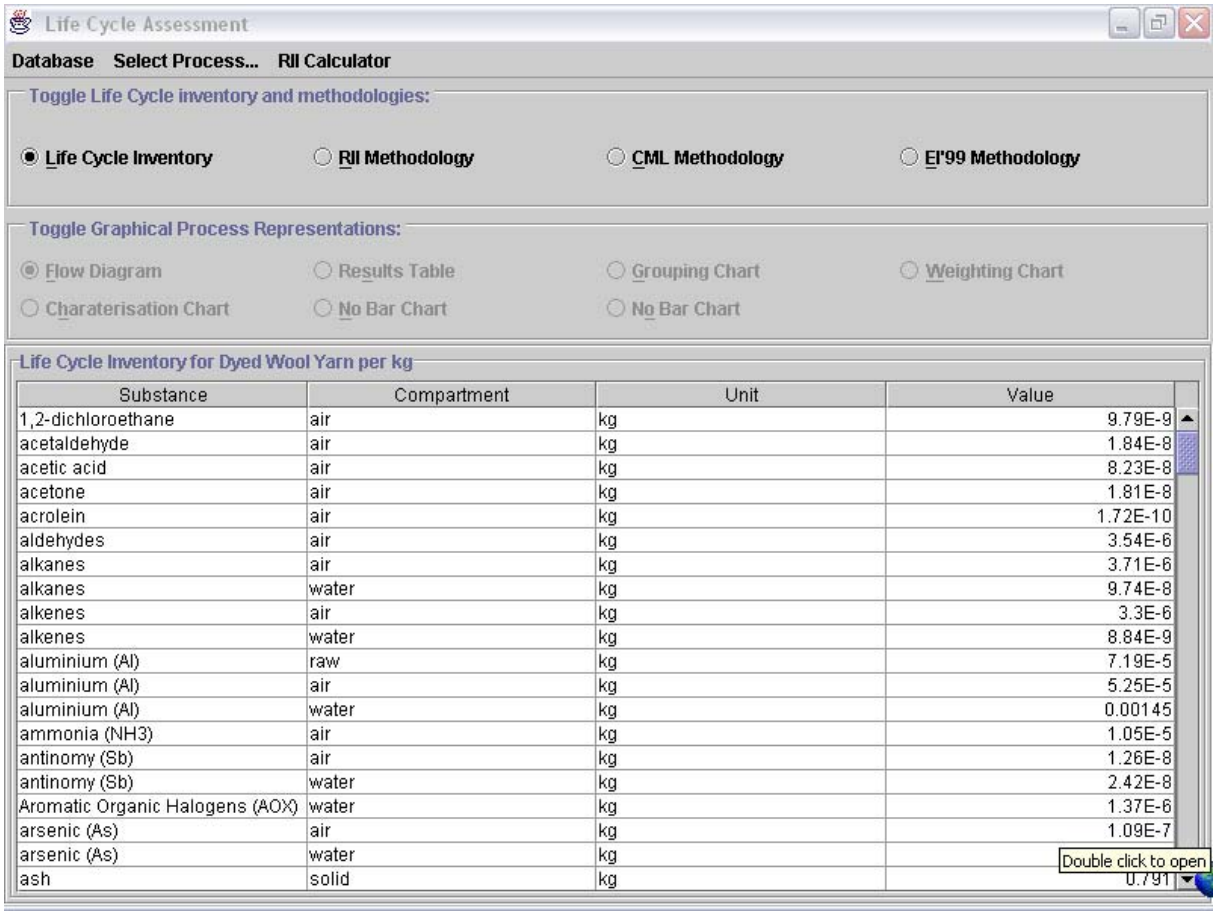
LCI constituent	Medium	Unit	Value
titanium tetrachloride	water	µg	0.73
toluene	water	µg	1.3
toluene diisocyanate	water	µg	0.1
triazophos	water	µg	0.56
trichloroacetic acid	water	µg	200
1,2,3-trichlorobenzene	water	µg	0.28
1,1,1-trichloroethane	water	µg	5.3
1,1,2-trichloroethane	water	µg	8.2
trichloroethylene	water	µg	0.1
2,4,5-trichlorophenol	water	µg	0.045
triethylamine	water	µg	8
triethylenetetramine	water	µg	1
trifluralin	water	µg	0.021
trimethylamine	water	µg	8
1,3,5-trimethylbenzene	water	µg	360
trinitrobenzene	water	µg	0.103
trinitrotoluene	water	µg	0.16
urea	water	µg	29
vanadium (V)	water	µg	1.3
vanadium pentoxide	water	µg	0.1
vinyl acetate	water	µg	1.8
vinyl bromide	water	µg	1
vinyl chloride	water	µg	0.1
warfarin	water	µg	1.2
xylene	water	µg	1.1
zinc (Zn(II))	water	µg	0.7
zinc chloride	water	µg	1
zirconium (Zr)	water	µg	2
Occupy as existing severely industrialised	non material	$mm^2.a$	345.11

Appendix G: Screen shots of the LCIA software application

Opening screen: wool life cycle assessment case study

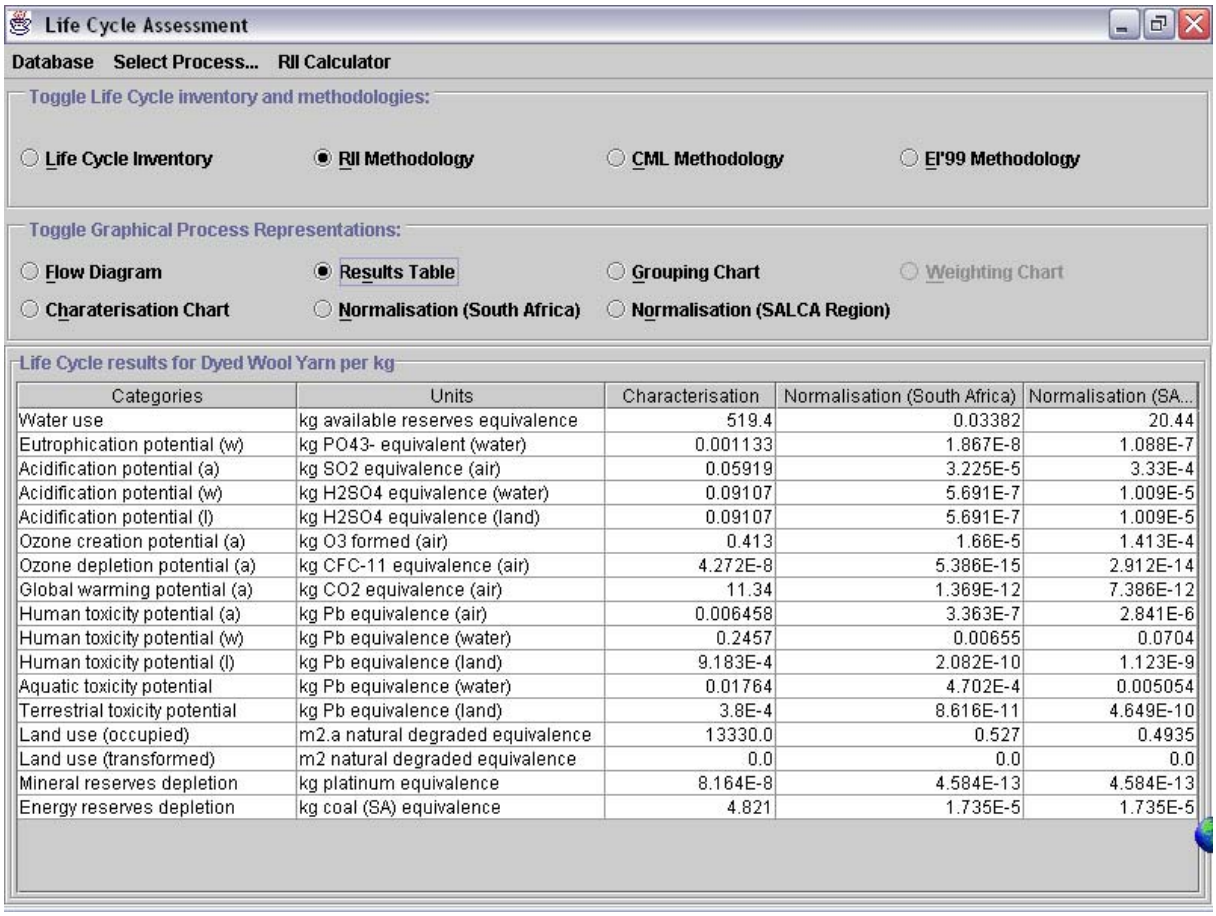


Life cycle inventory results: wool case study



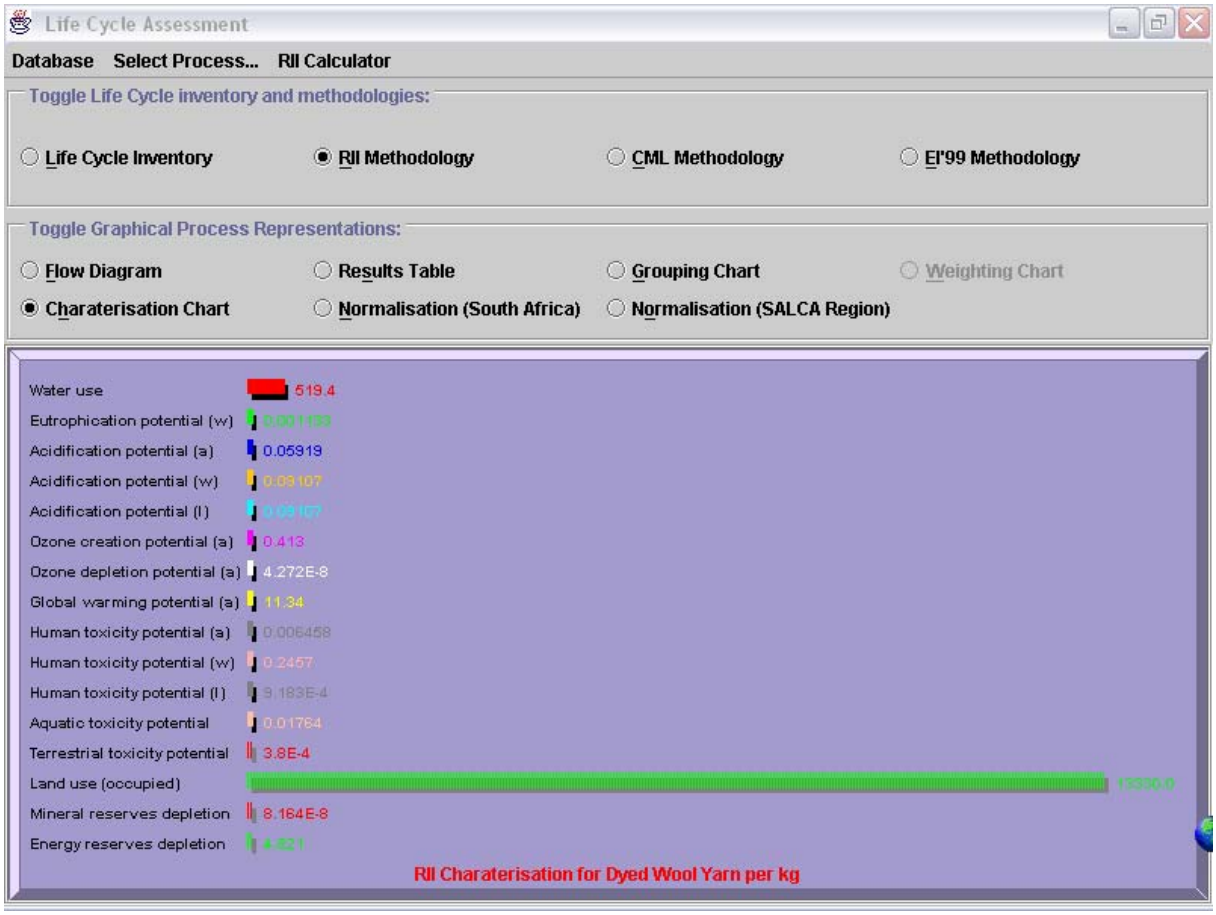
Substance	Compartment	Unit	Value
1,2-dichloroethane	air	kg	9.79E-9
acetaldehyde	air	kg	1.84E-8
acetic acid	air	kg	8.23E-8
acetone	air	kg	1.81E-8
acrolein	air	kg	1.72E-10
aldehydes	air	kg	3.54E-6
alkanes	air	kg	3.71E-6
alkanes	water	kg	9.74E-8
alkenes	air	kg	3.3E-6
alkenes	water	kg	8.84E-9
aluminium (Al)	raw	kg	7.19E-5
aluminium (Al)	air	kg	5.25E-5
aluminium (Al)	water	kg	0.00145
ammonia (NH3)	air	kg	1.05E-5
antinomy (Sb)	air	kg	1.26E-8
antinomy (Sb)	water	kg	2.42E-8
Aromatic Organic Halogens (AOX)	water	kg	1.37E-6
arsenic (As)	air	kg	1.09E-7
arsenic (As)	water	kg	
ash	solid	kg	0.791

Life cycle impact assessment results (RII): wool case study

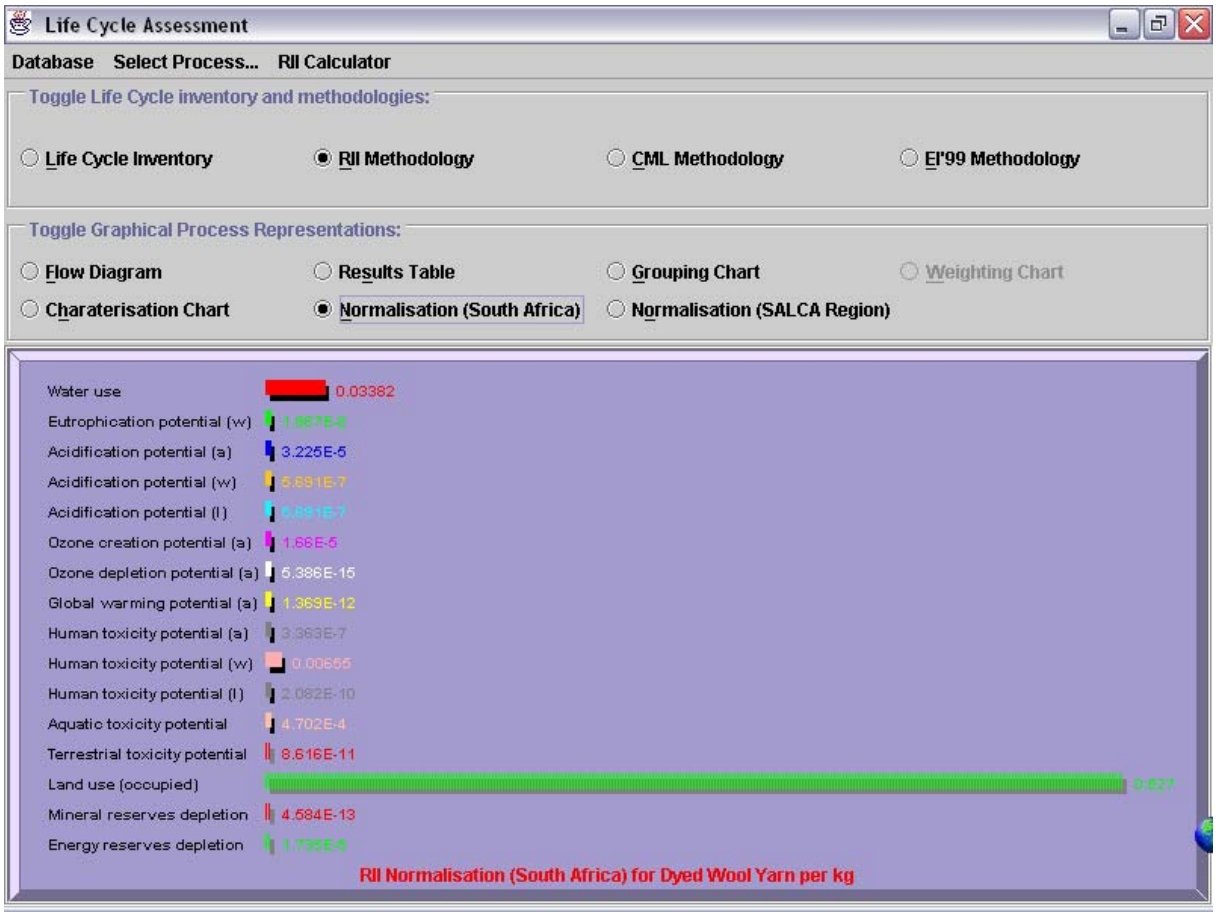


Categories	Units	Characterisation	Normalisation (South Africa)	Normalisation (SA...
Water use	kg available reserves equivalence	519.4	0.03382	20.44
Eutrophication potential (w)	kg PO43- equivalent (water)	0.001133	1.867E-8	1.088E-7
Acidification potential (a)	kg SO2 equivalence (air)	0.05919	3.225E-5	3.33E-4
Acidification potential (w)	kg H2SO4 equivalence (water)	0.09107	5.691E-7	1.009E-5
Acidification potential (l)	kg H2SO4 equivalence (land)	0.09107	5.691E-7	1.009E-5
Ozone creation potential (a)	kg O3 formed (air)	0.413	1.66E-5	1.413E-4
Ozone depletion potential (a)	kg CFC-11 equivalence (air)	4.272E-8	5.386E-15	2.912E-14
Global warming potential (a)	kg CO2 equivalence (air)	11.34	1.369E-12	7.386E-12
Human toxicity potential (a)	kg Pb equivalence (air)	0.006458	3.363E-7	2.841E-6
Human toxicity potential (w)	kg Pb equivalence (water)	0.2457	0.00655	0.0704
Human toxicity potential (l)	kg Pb equivalence (land)	9.183E-4	2.082E-10	1.123E-9
Aquatic toxicity potential	kg Pb equivalence (water)	0.01764	4.702E-4	0.005054
Terrestrial toxicity potential	kg Pb equivalence (land)	3.8E-4	8.616E-11	4.649E-10
Land use (occupied)	m2.a natural degraded equivalence	13330.0	0.527	0.4935
Land use (transformed)	m2 natural degraded equivalence	0.0	0.0	0.0
Mineral reserves depletion	kg platinum equivalence	8.164E-8	4.584E-13	4.584E-13
Energy reserves depletion	kg coal (SA) equivalence	4.821	1.735E-5	1.735E-5

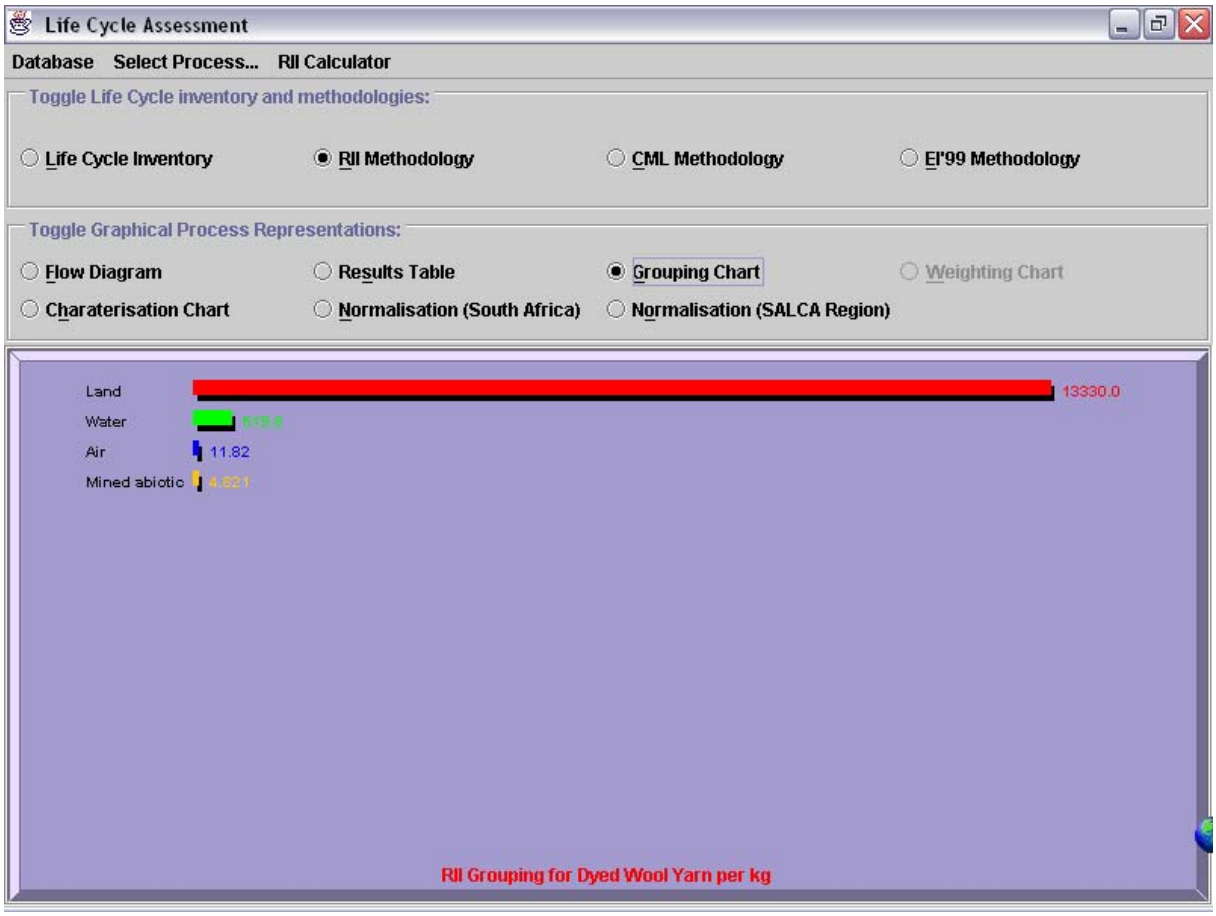
RII characterisation graphical results: wool case study



RII normalisation graphical results: wool case study



RII final results (natural resource groups): wool case study



Appendix H: LCIA software application and Java code

The application software (downloadable from the internet [211]) is used as a tool to demonstrate the different Life Cycle Impact Assessment (LCIA) methodologies that are used for Life Cycle Assessments (LCAs). Three LCIA methodologies are incorporated in the software:

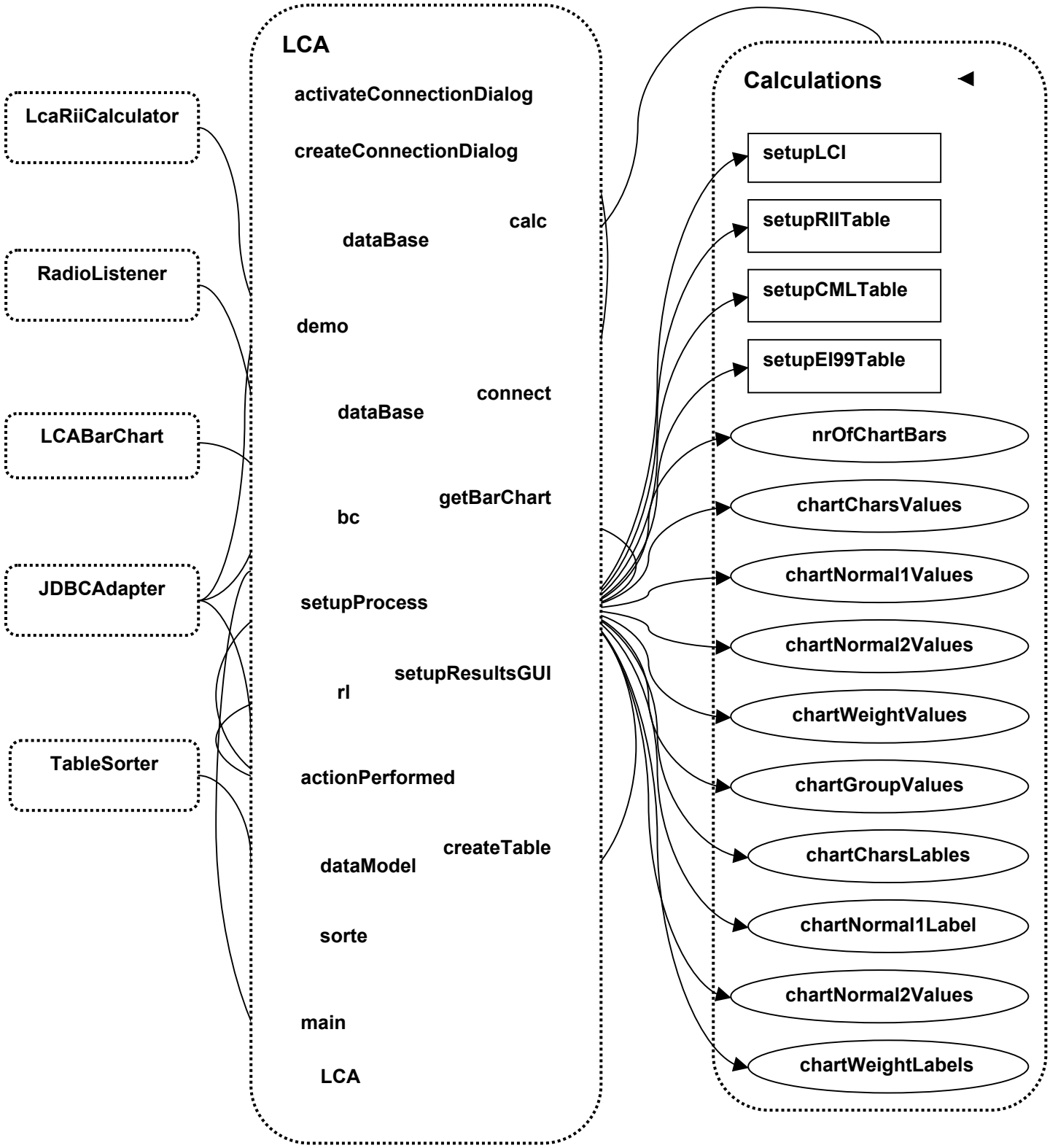
- Resource Impact Indicator (RII) methodology, from the University of Pretoria, South Africa,
- Centre for Environmental Studies (CML) methodology, from the University of Leiden, the Netherlands, and
- Eco-Indicators '99 methodology, from Pré Consultants, the Netherlands.

The results of the LCIA models are displayed for two complete South African Life Cycle Inventories (LCIs):

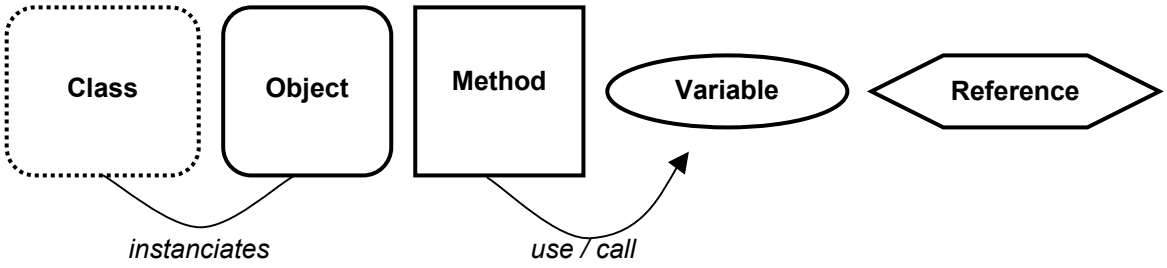
- The manufacturing of dyed two-fold yarn wool, i.e. a cradle-to-gate life cycle system.
- The manufacturing and assembly of magnesium alloy brake callipers in automobiles for export to and use in Europe.

As an additional feature, the application also provides users with the option to use a RII Calculator, which is based on the RII methodology. Thereby, indicator values of the impacts on the four natural resource groups (Water, Air, Land and Mined Abiotic Resources) can be calculated for systems where only limited process information is available, i.e. water and energy usage, and waste production (see Chapters 4 and 5). The RII Calculator uses South African specific information and is based on Figure 4.7 of Chapter 4.

This appendix explains the Java code for the application software, and is therefore intended for possible further development and improvement. Extracts from the code are given for explanatory purposes.



The figure on the previous page represents the interaction of the main `LCA.java` class with the other Java classes. It must be noted that only the relevant methods and variables from the classes are defined. The figure represents relationships as follows:



LCA.java

The first piece of code is a void method `activateConnectionDialog()` that activates the connection dialog for the user. The second public void method creates the connection dialog box.

```
etchedBorder      = BorderFactory.createEtchedBorder();
beveledBorder     = BorderFactory.createRaisedBevelBorder();
lowBevelBorder    = BorderFactory.createLoweredBevelBorder();
emptyBorder       = BorderFactory.createEmptyBorder(20,20,20,20);
emptyEtched       = BorderFactory.createCompoundBorder(etchedBorder,
                     BorderFactory.createEmptyBorder(5,5,5,5));
```

Source code H.1

The `LCA.java` class constructor follows. The constructor of the LCA class is responsible for most of the application set-up when the application is started.

First, it initialises the borders used in the application Graphical User Interface (GUI). Then it creates the connection dialog box with the `createConnectionDialog()` method call. Thereafter, the menu bar and menu items are set-up as well as the radio buttons in the menu bar. An `ActionCommand`, `ActionListener` and `Mnemonic`

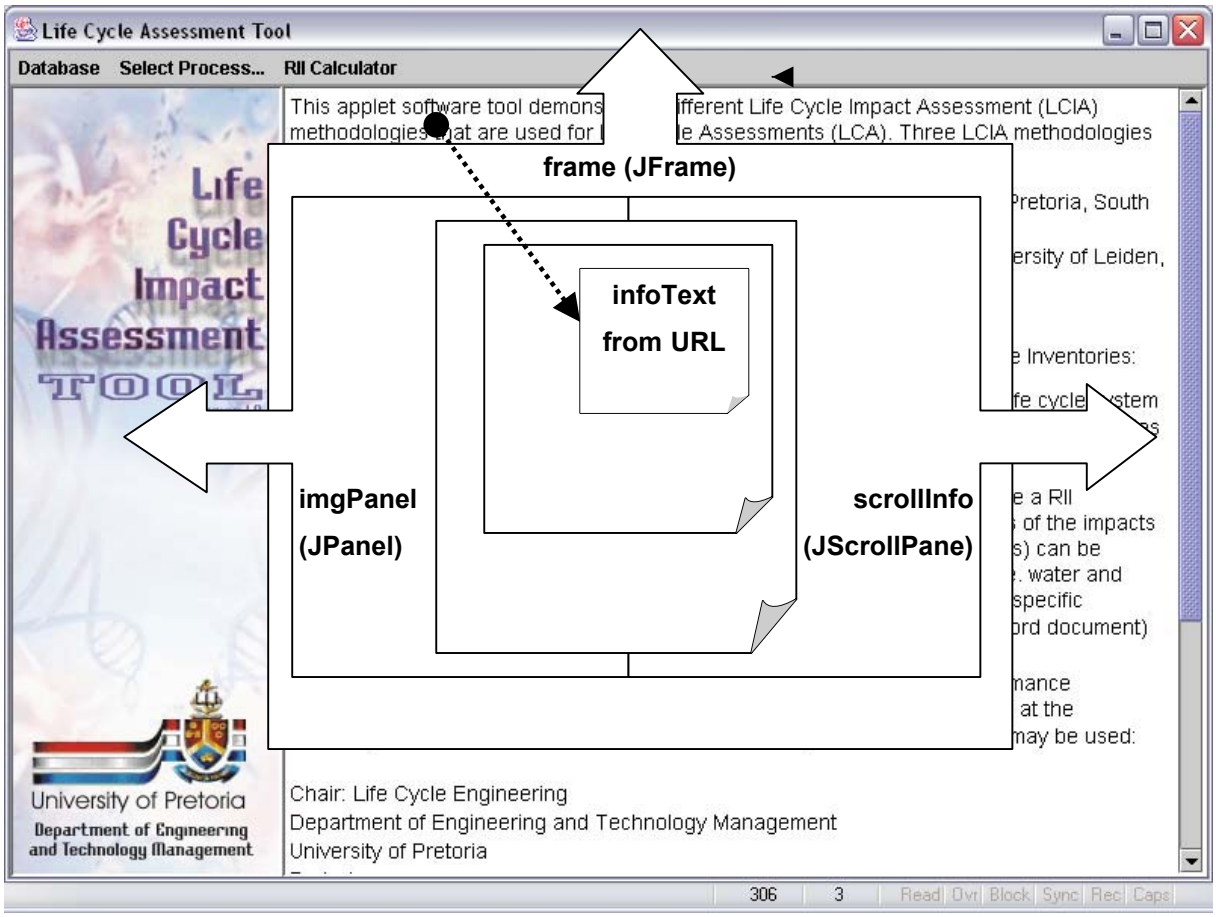
key events are added to each button on the menu bar. Then the main panel set-up is done with the following code:

```
//Main Panel Setup

      mainPanel = new JPanel(new BorderLayout());

      //Set image left
      imgPanel = new JPanel(new GridLayout(1,1));
      dnaimg = new ImageIcon("images/lcadna5.jpg");
      img = new JLabel(dnaimg);
      imgPanel.setBorder(BorderFactory.createLoweredBevelBorder());
      imgPanel.setPreferredSize(new Dimension(180,580));
      imgPanel.add(img);
      mainPanel.add(imgPanel, BorderLayout.WEST);
      try {
         String s = "file:"
         + System.getProperty("user.dir")
         + System.getProperty("file.separator")
         + "images/info.html";
         URL infoText = new URL(s);
         text = new JEditorPane(infoText);
         text.setEditable(false);
         JPanel textPane = new JPanel(new GridLayout(1,1));
         //textPane.setBorder(BorderFactory.createLoweredBevelBorder());
         textPane.add(text);
         JScrollPane scrollInfo = new JScrollPane(textPane,
            JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
            JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
         textPane.setPreferredSize(new Dimension(450,750));
         scrollInfo.setBorder(BorderFactory.createLoweredBevelBorder());
         mainPanel.add(scrollInfo, BorderLayout.CENTER);
            } catch (IOException ioe)
            {
            System.out.println("Info.html not found!");
            }
```

Source code H.2

```
// Create a Frame and put the main panel in it.
frame = new JFrame("Life Cycle Assessment Tool");
frame.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {System.exit(0);}});
frame.setBackground(Color.lightGray);
frame.getContentPane().add(mainMenu, "North");
frame.getContentPane().add(mainPanel, "Center");
frame.pack();
frame.setVisible(false);
frame.setBounds(0, 0, 800, 580);

//Start application...
activateConnectionDialog();
calc = new Calculations(dataBase);
```

Source code H.3



In order to start the application the `activateConnectionDialog()` method is called, which in turn calls the `connect()` method that instantiates and establishes a connection with the database using the parameters obtained by the connection dialog box (Source code H.4). Now it is possible to create a new instance of the `Calculations.java` class as a reference to the database connection called 'database' is available. This is necessary for the constructor of the `Calculations.java` class (Source code H.3).

```
public void connect() {
        dataBase = new JDBCAdapter(
        serverField.getText(),
        driverField.getText(),
        userNameField.getText(),
```

Source code H.4

The `actionPerformed(ActionEvent e)` method reacts on any input received from the radio buttons on the 'Select Process' menu bar. It converts the input to a string representing one of the processes from the list then it calls the `setupResultsGUI()` method which sets up the resulting GUI and then it calls the `setupProcess(source)` method where the source indicates which process will be assessed.

The `setupResultsGUI()` method is responsible for the set-up of four radio buttons, which allows the user to select one of the three methodologies and also the inventory of the selected process. This method will also set-up up to seven other radio buttons that makes it possible for the user to select the graphical presentation of the selected process. Each methodology defines its own bar chart representation for example: the RII methodology defines Characterisation, Normalisation and Grouping while the Eco Indicator uses Characterisation, Normalisation and Weighting and the CML methodology defines only Characterisation and Normalisation values. Therefore, each group of radio buttons is unique for each newly selected methodology. An external class named `RadioListener`, instantiated in this method, responds on the actions by the user and sets up the GUI accordingly as each

selection of a methodology has different types of bar chart representations as explained before. Therefore, the constructor of the `RadioListener` class needs references to the entire GUI; panels, buttons, etc. that needs to be changed by the listener.

The `setupProcess(source)` method (with the source parameters) is the selected process for which the assessment results need to be set-up. The method can be divided into four main units. The first unit sets up the imaging part of the resulting GUI, such as the data flow diagram. The following units set up the tables and bar charts for each of the different methodologies in this order: RII, CML and Eco Indicator '99. There are two important methods used in `setupProcess(…)`:

- The `createTable(data, names, whichTableHeader)` method where data is an Object matrix containing the table data, names a String array containing the table header names, and
- *whichTableHeader* is an integer indicator of which table headers is being used – this is necessary since each table must be rendered differently as their headers differs.

Thus, if a new table is added to the GUI, new table headers are defined in an array and it is then specified which columns should be rendered by adding a new table rendering strategy to the if-block, which will be indicated by the *whichTableHeader* parameter. The defined `LCATable` class sets up a data model of the data and the names received for the table. This data model is sent to the `TableSorter` class. This class implements the `TableModel` class, but does not store or copy the data in the `TableModel`, instead it maintains an array of integers that is kept the same size as the number of rows in its model. When the model changes it notifies the sorter that something has changed. At a later stage, changes made by the user to the data in the tables could be saved to a database, but this functionality is not currently used within this application. The sorted data is sent to the `Table` class and a table is created and returned on a scroll bar (Source code H.5).

```
public JScrollPane createTable(final Object[][] data,
                                                final String[] names,
                                                int whichTableHeader)
{
      LCATable dataModel = new LCATable(data, names);
      // Create the table
      TableSorter  sorter = new TableSorter(dataModel);
      JTable    table = new JTable(sorter);
      sorter.addMouseListenerToHeaderInTable(table);
      table.setAutoResizeMode(1);
      TableColumn numbersColumn;
      DefaultTableCellRenderer numberColumnRenderer =
                              new DefaultTableCellRenderer() {
      public void setValue(Object value) {
            int cellValue = (value instanceof Number) ?
                  ((Number)value).intValue() : 0;
            setForeground((cellValue >= 0) ? Color.black : Color.red);
            setText((value == null) ? "" : value.toString()); }};
      numberColumnRenderer.setHorizontalAlignment(JLabel.RIGHT);
      if (whichTableHeader == 1) {
            numbersColumn = table.getColumn("Value");
            numbersColumn.setCellRenderer(numberColumnRenderer);
      } else if (whichTableHeader == 2) {
            numbersColumn = table.getColumn("Characterisation");
            numbersColumn.setCellRenderer(numberColumnRenderer);
            numbersColumn = table.getColumn("Normalisation (World)");
            numbersColumn.setCellRenderer(numberColumnRenderer);
            numbersColumn =
                  table.getColumn("Normalisation (West European)");
            numbersColumn.setCellRenderer(numberColumnRenderer);
      } else if (whichTableHeader == 3) {
            //for the rest of the columns…

      JScrollPane scrollpane = new JScrollPane(table);

      return scrollpane;

}//create Table
```

Source code H.5

In the `getBarChart(String chartTitle, int nrOfBars, float[] data, String[] labels)` method, where *chartTitle* is a String containing the name of the bar chart, *nrOfBars* represents the number of bars on the chart, *data* is a float array that contains the bar values in the correct sequence, and *labels* is a String array that contains the bar names in the same sequence. This method returns a *JPanel* with the bar chart on it, but in reality it only obtains the parameters needed for the constructor of the `LCABarChart` class. It then creates a new class instance with the given parameters and then calls a public method named `drawChart()` in this class that creates the bar chart and returns a *JPanel* which, on its return, is also returned by this method. The rest of the method body is needed to create the different values for the rest of the `LCABarChart` class instantiation.

For each methodology the number of bar charts differs as well as the table setup. Refer to Source Code H.6 for the methodology set-up done in `setupProcess(…)`. `calc.setupRIITable(…)`, `calc.nrOfChartBars(…)`, `calc.chartNormal1Values(…)` and `calc.chartNormal1Labels(…)` are calls to public methods in the `Calculations.java` class. Their return types are as discussed earlier. Their functionality is discussed under the `Calculations.java` class.

Calculations.java

The constructor of the `Calculations.java` class receives the database object from the `JDBCAdapter` class and makes it available to the rest of the class members. The `doPost(…)`, `doGet(…)` and `gotoPage(…)` methods are only for the purposes of the server/client interaction and communication. The calculations are done on the server side and the GUI resides on the client side.

The `getLCIOutputs(String process)` method is run from the LCA class. This method runs a query on the database with the received process name and retrieves the inventory of for this process and writes the data to four vectors: *substances*, *compartments*, *units* and *totals*.

The `convertUnits()` method converts the units of the values retrieved from the database. This is done by multiplying or dividing the totals with the appropriate factors to obtain the desired units for the tables. The following conversions are done to standard units:

- Pounds(lb) to kilogram(kg)
- Squared centimeters to squared meters
- Squared millimeters to squared meters
- Cubic centimeters to cubic meters
- Joule(J) to kilo joule(kJ)
- Mega joule(MJ) to kilo joule(kJ)
- Grams(g) to kilograms(kg)
- Milligram(mg) to kilogram(kg)
- Microgram(µg) to kilogram(kg)
- Nano gram(ng) to kilogram(kg)
- Pico gram(pg) to kilogram(kg)

Then the converted units and totals are written to two new vectors named *cTotals* and *cUnits*. The `makeLCItable()` method is then used to write the new set of vectors into a matrix that will be used in the inventory table in the GUI.

`setupLCI(…)` is a public method that could be called outside this class that would perform the operations as described above and return a matrix containing the inventory data.

The `getImpacts(String impactDBtable)` method receives the name of the impacts table. A query is then run from the database and the impacts substances that match the categories and substances from the inventory are then retrieved and written into a matrix containing the *totals*, *units* and *compartments*.

The `calculateCharacterization (String normalizeTable)` method actually also calculates the first normalisation values as the characterisation values are only calculated by summing the impact totals for each unique category. A query

retrieves the normalisation factors from the database for each category and these values are then multiplied with the previously calculated characterisation values. The calculated values are written in two vectors named *chars* and *normal*.

The `calculateNormalisation(String normalizeDBtable1)` method makes use of any other normalisation table to calculate the rest of the normalisation values, as it is written generically. These values are written to a vector named *normal2T*.
The `trimNumbers()` method is written to convert the numbers in the above mentioned vectors to an easy readable format.

If a methodology requires that weighting values must be calculated, the `calculateWeighting(String weightingTable)` method is called. This method also runs a query that retrieves the weighting factors for each category from the database and then multiplies these values with the characterization values in the chars vector. These values are written to a vector called *ecoWeighting*.

Grouping is only required by the RII methodology. The grouping values are only presented in a chart on the GUI. The method `calculateRIIGrouping(String groupingTable)` retrieves the grouping for each category from the database. That is:

- Land
- Water
- Air
- Mined abiotic

It then sums the substance values by their categories that fall into these grouping areas. It writes these values to a float array called *riiGrouping* in the above order.

The `fillRIIMatrix()`, `fillCMLMatix()` and `fillEI99Matrix()` methods set up the matrixes for each of the different methodologies to be called for the tables in the GUI.

The `returnCharsChartValues()`, `returnNormal1ChartValues()`, `returnNormal2ChartValues()` and `returnWeightingValues()` method set up vectors containing the data for the bar charts in the GUI.

The `flushLciVectors()` and `flushImpactVectors()` methods are only written for the housekeeping of the class variables. All the vectors need to be cleared before they are populated with new data.

The `setupRIITable(…)`, `setupCMLTable(…)`, `setupEI99Table(…)` are public methods; available outside the class and called from LCA.java and make the data accessible for the GUI. Note that all the methods discussed earlier are private to the class.

The methods called hereafter are for the set up and use of the RII calculator. They don't differ from those discussed earlier, although their names have been changed. It is noted that the code is the same. The important method for the RII calculator is the `riiCalculator(…)` method that receives a number of parameters. This method sets up the vectors with data for the used inventory for water, steam, fuel, etc. It then retrieves the impacts of the parameters used, calculates the normalisation and then performs the grouping as discussed above. The resulting float array *riiGrouping* is then used for the GUI.

The source for the Calculations.java class is given below:

```
import java.sql.*;
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
//import com.*;
import lca.javagently.Text;

public class Calculations { //FOR APPLET USE: extends HttpServlet {
  Vector totals = new Vector();
  Vector substances = new Vector();
```

```
Vector compartments = new Vector();
Vector units  = new Vector();
//The c prefix indicate a quantity conversion
Vector cUnits = new Vector();
Vector cTotals = new Vector();
Vector impcom = new Vector();
Vector imptot = new Vector();
Vector impsub = new Vector();
Vector impcat = new Vector();
Vector impuni = new Vector();
Vector impact = new Vector();
Vector invtot = new Vector();
Vector chars = new Vector();
Vector cats = new Vector();
Vector normal = new Vector();
//The T indicates that the numbers are trimmed
Vector normal2T = new Vector();
Vector charsT = new Vector();
Vector normalT = new Vector();
Vector trimChars = new Vector();
Vector normfact = new Vector();
Vector categories = new Vector();
Vector catunits = new Vector();
Vector ecoWeighting = new Vector();
//RII Calculation vectors:
Vector riiSubstances = new Vector();
Vector riiCompartments = new Vector();
Vector riiTotals = new Vector();
Vector riiUnits = new Vector();
//***************************
Object[][] impacts;
Object[][] lci;

//Applet setup parameters:
String process;
Boolean cmlNorm1, cmlNorm2, cmlNorm3, cmlNorm4;
Boolean riiNorm1, riiNorm2, riiNorm3, riiNorm4, riiNorm5;
Boolean riiGroup, riiCalc;
//Applet request strings
String applet = "APPLET";
```

```
  String lcimat = "LCIMAT";
  String cmlmat = "CMLMAT";
  String riimat = "RIIMAT";
  String ecomat = "ECOMAT";

  String query, lcp;
  String [] cmlTables = {"Adep", "AP", "Ecotox", "EP", "GWP",
                                    "Htox", "Landuse", "ODP", "POCP",
"Solids"};
  //Chart setup variables:
  int nrOfChartBars;
  float [] chartCharsValues, chartNormal1Values, chartNormal2Values;
  float [] chartWeightValues, chartGroupValues;
  float [] riiGrouping;
  String [] chartCharsLabels, chartNormal1Labels, chartNormal2Labels;
  String [] chartWeightLabels;

  double [] cmlValues = new double[10];

  int ss, sr;

  private ServletContext context;   // objects used to transfer control to
the jsp pages
    private RequestDispatcher dispatcher;

  JDBCAdapter dataBase;
    Connection connection = null;   // the connection to the database
  Statement stmt = null;   // a statement object for the queries
  String dbName;           // the database name

  public Calculations(JDBCAdapter db) {
      dataBase=db;
  }

  public Calculations() {}

  /*****************************************************
  THE FOLLOWING WILL ONLY BE USED BY AN APPLET/SERVLET
  *****************************************************
  public void doPost(HttpServletRequest req, HttpServletResponse res)
```

```
          throws ServletException, IOException
    {
      doGet(req, res);
  }//do Post

  public void doGet(HttpServletRequest req, HttpServletResponse res)
          throws ServletException, IOException
    {
      String url = "jdbc:odbc:wool";
      String driverName = "sun.jdbc.odbc.JdbcOdbcDriver";
      String user = "";
      String passwd = "";

      ObjectInputStream in = new ObjectInputStream(req.getInputStream());
      try {
            String request = (String)in.readObject();
            System.out.println("Applet requested: " + request);
            if (request.compareTo(applet)==0) {
                  //No need for database connection!
                  process = (String)in.readObject();
            } else if (request.compareTo(lcimat)==0) {
                  //Need to connect to the database...
                  Class.forName(driverName);
                  connection =
DriverManager.getConnection(url,user,passwd);

                  dataBase = new JDBCAdapter(url, driverName, user,
                                                      passwd, connection);
                  System.out.println("Database connection established.");
                  if (process != null) setupLCI (process);
                  else System.out.println("Reload applet. Process not
found!");
                  //Matrix must be ready to send.
                  res.setContentType("application/x-java-serialized-
object");
                  ObjectOutputStream out = new
ObjectOutputStream(res.getOutputStream());
                  out.writeObject(lci);
                  out.flush();
            } else if (request.compareTo(cmlmat)==0) {
```

```
                    //Verify if database have been closed.
                    setupCMLTable((String)in.readObject(),
                                                (String)in.readObject(),
                                                (String)in.readObject());
                    //Required data must be ready to send
                    res.setContentType("application/x-java-serialized-
object");
                    ObjectOutputStream out = new
ObjectOutputStream(res.getOutputStream());
                    out.writeObject(impacts);
                    returnChartValues(normalT, categories);
                    out.writeObject(new Integer(nrOfChartBars));
                    out.writeObject(chartValues);
                    out.writeObject(chartLabels);
                    returnChartValues(normal2T, categories);
                    out.writeObject(new Integer(nrOfChartBars));
                    out.writeObject(chartValues);
                    out.writeObject(chartLabels);
                    returnChartValues(charsT, categories);
                    out.writeObject(new Integer(nrOfChartBars));
                    out.writeObject(chartValues);
                    out.writeObject(chartLabels);
                    out.flush();
                    flushImpactVectors ();
                    System.out.println(":::Request for tables successfully
completed!");
               } else if (request.compareTo(riimat)==0) {
                    //Verify if database have been closed.
                    setupRIITable((String)in.readObject(),
                                                (String)in.readObject(),
                                                (String)in.readObject(),
                                                (String)in.readObject());
                    //Required data must be ready to send
                    res.setContentType("application/x-java-serialized-
object");
                    ObjectOutputStream out = new
ObjectOutputStream(res.getOutputStream());
                    out.writeObject(impacts);
                    returnChartValues(normalT, categories);
                    out.writeObject(new Integer(nrOfChartBars));
```

```
                    out.writeObject(chartValues);
                    out.writeObject(chartLabels);
                    returnChartValues(normal2T, categories);
                    out.writeObject(new Integer(nrOfChartBars));
                    out.writeObject(chartValues);
                    out.writeObject(chartLabels);
                    returnChartValues(charsT, categories);
                    out.writeObject(new Integer(nrOfChartBars));
                    out.writeObject(chartValues);
                    out.writeObject(chartLabels);
                    out.writeObject(riiGrouping);
                    out.flush ();
                    flushImpactVectors ();
                    System.out.println(":::Request for tables successfully
completed!");
               } else if (request.compareTo(ecomat)==0) {
                    //Verify if database have been closed.
                    setupEI99Table((String)in.readObject(),
                                        (String)in.readObject(),
                                        (String)in.readObject());
                    //Required data must be ready to send
                    res.setContentType("application/x-java-serialized-
object");
                    ObjectOutputStream out = new
ObjectOutputStream(res.getOutputStream());
                    out.writeObject(impacts);
                    returnChartValues(normalT, categories);
                    out.writeObject(new Integer(nrOfChartBars));
                    out.writeObject(chartValues);
                    out.writeObject(chartLabels);
                    returnChartValues(ecoWeighting, categories);
                    out.writeObject(new Integer(nrOfChartBars));
                    out.writeObject(chartValues);
                    out.writeObject(chartLabels);
                    returnChartValues(charsT, categories);
                    out.writeObject(new Integer(nrOfChartBars));
                    out.writeObject(chartValues);
                    out.writeObject(chartLabels);
                    out.flush ();
                    flushImpactVectors ();
```

```
                  System.out.println(":::Request for tables successfully
completed!");
            } else System.out.println("Request unknown. Exit normally...");
      } catch (IOException e) {
            // if there is an error with the database source transfer
control to an error page
            //connection = null;
            //gotoPage("/lca/errorDbSource.jsp",req,res);
            System.out.println("An error occured. Objects received not in
sinc.");
      } catch (ClassNotFoundException cnfe) {
            System.out.println("Class not found! "+ cnfe);
      } catch     (SQLException sqle) {
            System.out.println("SQL Error @ " + sqle);
            }//catch
    }//doGet

      private void gotoPage(String address,
                                    HttpServletRequest request,
                                    HttpServletResponse response)
                                    throws ServletException, IOException {
            RequestDispatcher dispatcher =
            getServletContext().getRequestDispatcher(address);
            dispatcher.forward(request, response);
      }

//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//
//*    RETRIEVE INVENTORY FROM DATABASE,                     *//
//*    COVERT UNITS/VALUES,                                  *//
//*    SETUP LCI TABLE WITH COLLECTED DATA                   *//
//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *//
   public void getLCIOutputs (String process) {

      query = "SELECT * FROM [" + process + "]";

      dataBase.executeQuery(query, true);

      sr = dataBase.getRowCount();

      for (int p = 0; p < sr; p++) {
```

```
            substances.add(dataBase.getValueAt(p,0));
            compartments.add(dataBase.getValueAt(p,1));
            units.add(dataBase.getValueAt(p,2));
            totals.add(dataBase.getValueAt(p,3));
      }
      System.out.println("Inventory read from database.");
      System.out.println("Inventory vectors completed.");

   }//getProcessOutputs

   private void convertUnits () {
      double convert;
      double divBy;
      String iStr;
      Object cUnit, cTotal;

      //System.out.println(sr);
      //Convert units in inventory vectors
      for (int i = 0; i < sr; i++){
            //System.out.println(units.get(i) + " " + i);
            iStr = (String)units.get(i);//.valueOf(units.get(i));
            convert = Double.parseDouble(totals.get(i).toString());
            //System.out.print(iStr + " ");
            String gram = "g";
            String miligram = "mg";
            String microgram ="µg";
            String nanogram = "ng";
            String picogram = "pg";
            String joule = "J";
            String megaj = "MJ";
            String pound = "lb";
            String sqaredcm = "cm2.a";
            String sqaredmm = "mm2.a";
            String cubiccm = "cm3";
            String kilobarq = "kBq";

            if (kilobarq.compareTo(iStr)==0) {
                  convert *= 1000;
                  cUnits.add("Bq");
```

```
                cTotals.add(new
Double(Double.parseDouble(Text.writeDouble(convert, 0, 3))));
            }
            else if (pound.compareTo(iStr)==0) {
                convert *= 0.4535924;
                cUnits.add("kg");
                cTotals.add(new
Double(Double.parseDouble(Text.writeDouble(convert, 0, 3))));
            }
            else if (sqaredcm.compareTo(iStr)==0) {
                convert /= 100*100;
                cUnits.add("m2.a");
                cTotals.add(new
Double(Double.parseDouble(Text.writeDouble(convert, 0, 3))));
            }
            else if (sqaredmm.compareTo(iStr)==0) {
                convert /= 1000*1000;
                cUnits.add("m2.a");
                cTotals.add(new
Double(Double.parseDouble(Text.writeDouble(convert, 0, 3))));
            }
            else if (cubiccm.compareTo(iStr)==0) {
                convert /= 100*100*100;
                cUnits.add("m3");
                cTotals.add(new
Double(Double.parseDouble(Text.writeDouble(convert, 0, 3))));
            }
            else if (joule.compareTo(iStr)==0) {
                convert /= 1000.0;
                cUnits.add("kJ");
                cTotals.add(new
Double(Double.parseDouble(Text.writeDouble(convert, 0, 3))));
            }
            else if (megaj.compareTo(iStr)==0) {
                convert *= 1000.0;
                cUnits.add("kJ");
                cTotals.add(new
Double(Double.parseDouble(Text.writeDouble(convert, 0, 3))));
            }
            else if (pound.compareTo(iStr)==0) {
```

```
                convert /= 1000.0;
                cUnits.add("kg");
                cTotals.add(new
Double(Double.parseDouble(Text.writeDouble(convert, 0, 3))));
            }
            else if (gram.compareTo(iStr)==0) {
                divBy = 1000.0;
                convert /= divBy;
                cUnits.add("kg");
                cTotals.add(new
Double(Double.parseDouble(Text.writeDouble(convert, 0, 3))));
            }
            else if (miligram.compareTo(iStr)==0) {
                divBy = 1000000.0;
                convert /= divBy;
                cUnits.add("kg");
                cTotals.add(new
Double(Double.parseDouble(Text.writeDouble(convert, 0, 3))));
            }
            else if (microgram.compareTo(iStr)==0) {
                divBy = 1000000000.0;
                convert /= divBy;
                cUnits.add("kg");
                cTotals.add(new
Double(Double.parseDouble(Text.writeDouble(convert, 0, 3))));
            }
            else if (nanogram.compareTo(iStr)==0) {
                divBy = 1000000000000.0;
                convert /= divBy;
                cUnits.add("kg");
                cTotals.add(new
Double(Double.parseDouble(Text.writeDouble(convert, 0, 3))));
            }
            else if (picogram.compareTo(iStr)==0) {
                divBy = 1000000000000000.0;
                convert /= divBy;
                cUnits.add("kg");
                cTotals.add(new
Double(Double.parseDouble(Text.writeDouble(convert, 0, 3))));
            }
```

```
            else {
                  cUnits.add(iStr);
                  cTotals.add(new
Double(Double.parseDouble(Text.writeDouble(convert, 0, 3))));
            }
      }//for i
      System.out.println("Inventory units and values converted.");

   }//convertUnits

   private void makeLCItable () {

      //There are only 4 colmns in the LCI table
      //and it would not change; thus:

      lci = new Object[sr][4];

      //Insert values in LCI matrix.
      for (int i = 0; i < sr; i++) {
            lci[i][0] = substances.get(i);
            lci[i][1] = compartments.get(i);
            lci[i][2] = cUnits.get(i);
            lci[i][3] = (Double)cTotals.get(i);
            //System.out.println(cUnits.get(i));
      }//for i

      System.out.println("LCI matrix completed and ready for table.");
   }//makeLCItable

   public void setupLCI (String _process) {
      lcp = _process;
      getLCIOutputs (_process);
      convertUnits ();
      makeLCItable ();
   }

//* * * * * * * * * * * * * * * * * * * * * * * * * * * *//
//*    RETRIEVE AND CALCULATE:                          *//
//*      - CHARACTERIZATION VALUES                      *//
//*    - NORMALIZATION VALUES                           *//
```

```
//*    - WEIGHTING VALUES (if any)                    */
//*    BASED ON INVENTORY DATA.                       */
//* * * * * * * * * * * * * * * * * * * * * * * * * * */

   private void getImpacts (String impactDBtable) {

      for (int p = 0; p < sr; p++) {
            query = "SELECT Compartment, Category, Substance, Total, Unit
FROM ["+ impactDBtable + "]";
            query += " WHERE Substance = '" + (String)substances.get(p);
            query += "' AND Compartment = '" + (String)compartments.get(p);
            query += "'";

            dataBase.executeQuery(query, false);
            ss = dataBase.getRowCount();
            for (int q = 0; q < ss; q++) {
                  impcom.add(dataBase.getValueAt(q,0));
                  impcat.add(dataBase.getValueAt(q,1));
                  impsub.add(dataBase.getValueAt(q,2));
                  imptot.add(dataBase.getValueAt(q,3));
                  impuni.add(dataBase.getValueAt(q,4));
                  invtot.add(cTotals.get(p));
            }//for q

      }//for p

      System.out.println("Impacts retrieved from " + impactDBtable +"
table.");
      //System.out.println("Counted: " + imptot.size());

   }// getImpacts

   private void calculateCharacterization (String normalizeDBtable) {

      double charVal;
      double normVal;
      query = "SELECT Category, Unit, Factor FROM [" + normalizeDBtable +
"] ";
      dataBase.executeQuery(query, false);
      int nrOfCats = dataBase.getRowCount();
```

```
      //Write categories to a vector & their units.
      for (int i = 0; i < nrOfCats; i++) {
            categories.add(dataBase.getValueAt(i,0));
            catunits.add(dataBase.getValueAt(i,1));
            normfact.add(dataBase.getValueAt(i,2));
      }

      //Now, sum all values of the same category.
      for (int j = 0; j < categories.size(); j++)
      {
            int counter = 0;
            charVal = 0;
            normVal = 0;
            for (int i = 0; i < imptot.size(); i++)
            {
                  if
(String.valueOf(categories.get(j)).compareTo(String.valueOf(impcat.get(i)))
== 0)
                  {     //System.out.println(impsub.get(i));
                        charVal +=
(Double.parseDouble(String.valueOf(imptot.get(i))) *

   Double.parseDouble(String.valueOf(invtot.get(i))));
                        counter++;
                  }//if
            }//for i
            normVal= Double.parseDouble(String.valueOf(normfact.get(j)));
            normVal*=charVal;
            //System.out.println(categories.get(j) + "\t: " + counter);
            chars.add(new Double(charVal));
            normal.add(new Double(normVal));

      }
      System.out.println("Characterisation values calculated.");
   }//calculateCharacterization

   //Normalisation for the rest of the regions are calculated here:
   private void calculateNormalisation (String normalizeDBtable1)
   {
      double normVal;
```

```
      String normQuery = "SELECT Factor FROM [" + normalizeDBtable1 + "] ";
      dataBase.executeQuery(normQuery, false);
      for (int i = 0; i < categories.size(); i++) {
            normVal =
(Double.parseDouble(String.valueOf(dataBase.getValueAt(i,0)))) *

   (Double.parseDouble(String.valueOf(chars.get(i))));
            normal2T.add(new
Double(Double.parseDouble(Text.writeDouble(normVal, 0, 3))));
      }//for i
      System.out.println("Normalisation values calculated.");
   }//calculateNormalization

   //Trim the numbers in the vectors: chars, normal & normal2T;
   private void trimNumbers () {
      double val;

      for (int i = 0; i < categories.size(); i++) {
            val = Double.parseDouble(String.valueOf(chars.get(i)));
            charsT.add(new Double(Double.parseDouble(Text.writeDouble(val,
0, 3))));
            val = Double.parseDouble(String.valueOf(normal.get(i)));
            normalT.add(new Double(Double.parseDouble(Text.writeDouble(val,
0, 3))));
      }
      System.out.println("Numbers trimmed to perfection.");
   }

   //Calculate weighting (if required)
   private void calculateWeighting (String weightingTable)
   {
      double weightVal;
      String query = "SELECT Factor FROM [" + weightingTable + "] ";
      dataBase.executeQuery(query, false);
      for (int i = 0; i < categories.size(); i++) {
            weightVal =
(Double.parseDouble(String.valueOf(dataBase.getValueAt(i,0)))) *
            (Double.parseDouble(String.valueOf(chars.get(i))));
            ecoWeighting.add(new
Double(Double.parseDouble(Text.writeDouble(weightVal, 0, 3))));
```

```
      }//for i
      System.out.println("Weighting values calculated.");
   }//calculateWeighting

   //A non-generic method for calculating the grouping values for RII
   //Note: Not for matrix setup / table purposes - only for charts
   private void calculateRIIGrouping(String groupingTable)
   {
      riiGrouping = new float[4];
      double landVal=0, waterVal=0, airVal=0, mineVal=0;
      String[] groups = {"Land", "Water", "Air", "Mined abiotic"};
      String query = "SELECT Category, Group FROM [" + groupingTable + "]";
      dataBase.executeQuery(query, false);
      for (int i = 0; i < categories.size(); i++) { //For the 4 distinct
groups
            if
(String.valueOf(dataBase.getValueAt(i,1)).compareTo("Land")==0)
                  landVal +=
(Double.parseDouble(String.valueOf(chars.get(i))));
            else if
(String.valueOf(dataBase.getValueAt(i,1)).compareTo("Water")==0)
                  waterVal +=
(Double.parseDouble(String.valueOf(chars.get(i))));
            else if
(String.valueOf(dataBase.getValueAt(i,1)).compareTo("Air")==0)
                  airVal +=
(Double.parseDouble(String.valueOf(chars.get(i))));
            else if
(String.valueOf(dataBase.getValueAt(i,1)).compareTo("Mined abiotic")==0)
                  mineVal +=
(Double.parseDouble(String.valueOf(chars.get(i))));
      }
      //Add grouping values to vectors:
      riiGrouping[0] = (Float.parseFloat(Text.writeDouble(landVal, 0, 3)));
      riiGrouping[1] = (Float.parseFloat(Text.writeDouble(waterVal, 0,
3)));
      riiGrouping[2] = (Float.parseFloat(Text.writeDouble(airVal, 0, 3)));
      riiGrouping[3] = (Float.parseFloat(Text.writeDouble(mineVal, 0, 3)));

   }//calculateRIIGrouping
```

```
private void fillRIIMatrix () {

   impacts = new Object[categories.size()][5];

   //System.out.println(sr);
   for (int i = 0; i < categories.size(); i++) {
         impacts[i][0] = categories.get(i);
         impacts[i][1] = catunits.get(i);
         impacts[i][2] = charsT.get(i);
         impacts[i][3] = normalT.get(i);
         impacts[i][4] = normal2T.get(i);
   }
   System.out.println("RII Impact matrix completed.");
}

private void fillCMLMatrix () {

   impacts = new Object[categories.size()][5];

   //System.out.println(sr);
   for (int i = 0; i < categories.size(); i++) {
         impacts[i][0] = categories.get(i);
         impacts[i][1] = catunits.get(i);
         impacts[i][2] = charsT.get(i);
         impacts[i][3] = normalT.get(i);
         impacts[i][4] = normal2T.get(i);
   }

   System.out.println("CML Impact matrix completed.");
}

private void fillEI99Matrix () {

   impacts = new Object[categories.size()][5];

   //System.out.println(sr);
   for (int i = 0; i < categories.size(); i++) {
         impacts[i][0] = categories.get(i);
         impacts[i][1] = catunits.get(i);
         impacts[i][2] = charsT.get(i);
```

```
            impacts[i][3] = normalT.get(i);
            impacts[i][4] = ecoWeighting.get(i);
      }
      System.out.println("EI99 Impact matrix completed.");
   }

   private void returnCharsChartValues() {

      nrOfChartBars = charsT.size();
      chartCharsValues = new float[charsT.size()];
      chartCharsLabels = new String[charsT.size()];
      float p;
      int j=0;
      for (int i=0; i < charsT.size(); i++) {
            p = Float.parseFloat(String.valueOf(charsT.get(i)));
            if (p != 0.0) {
                  chartCharsValues[j] = p;
                  chartCharsLabels[j++] =
String.valueOf(categories.get(i));
            } else nrOfChartBars--;
      }
   }

   private void returnNormal1ChartValues() {
      nrOfChartBars = normalT.size();
      chartNormal1Values = new float[normalT.size()];
      chartNormal1Labels = new String[normalT.size()];
      float p;
      int j=0;
      for (int i=0; i < normalT.size(); i++) {
            p = Float.parseFloat(String.valueOf(normalT.get(i)));
            if (p != 0.0) {
                  chartNormal1Values[j] = p;
                  chartNormal1Labels[j++] =
String.valueOf(categories.get(i));
            } else nrOfChartBars--;
      }
   }

   private void returnNormal2ChartValues() {
```

```
      nrOfChartBars = normal2T.size();
      chartNormal2Values = new float[normal2T.size()];
      chartNormal2Labels = new String[normal2T.size()];
      float p;
      int j=0;
      for (int i=0; i < normal2T.size(); i++) {
            p = Float.parseFloat(String.valueOf(normal2T.get(i)));
            if (p != 0.0) {
                  chartNormal2Values[j] = p;
                  chartNormal2Labels[j++] =
String.valueOf(categories.get(i));
            } else nrOfChartBars--;
      }
   }

   private void returnWeightChartValues() {
      nrOfChartBars = ecoWeighting.size();
      chartWeightValues = new float[ecoWeighting.size()];
      chartWeightLabels = new String[ecoWeighting.size()];
      float p;
      int j=0;
      for (int i=0; i < ecoWeighting.size(); i++) {
            p = Float.parseFloat(String.valueOf(ecoWeighting.get(i)));
            if (p != 0.0) {
                  chartWeightValues[j] = p;
                  chartWeightLabels[j++] =
String.valueOf(categories.get(i));
            } else nrOfChartBars--;
      }
   }

   public void flushLciVectors () {
      totals.removeAllElements();
      substances.removeAllElements();
      compartments.removeAllElements();
      units.removeAllElements();
      cUnits.removeAllElements();
      cTotals.removeAllElements();
   }
```

```
private void flushImpactVectors () {
   impcom.removeAllElements();
   imptot.removeAllElements();
   impsub.removeAllElements();
   impcat.removeAllElements();
   impuni.removeAllElements();
   impact.removeAllElements();
   invtot.removeAllElements();
   chars.removeAllElements();
   charsT.removeAllElements();
   cats.removeAllElements();
   normal.removeAllElements();
   normalT.removeAllElements();
   normal2T.removeAllElements();
   normfact.removeAllElements();
   categories.removeAllElements();
   catunits.removeAllElements();
   ecoWeighting.removeAllElements();
}

public void setupRIITable (String riiImpacts,
                                        String riiNormalisation,
                                        String riiNormalisationII,
                                        String riiGrouping)
{
   if (substances.size() > 0) {
         getImpacts (riiImpacts);
         calculateCharacterization (riiNormalisation);
         calculateNormalisation (riiNormalisationII);
         calculateRIIGrouping (riiGrouping);
         trimNumbers ();
         fillRIIMatrix ();
         returnCharsChartValues();
         returnNormal1ChartValues();
         returnNormal2ChartValues();
   } else System.out.println("You need to set up the LCI before " +
               "the impacts could be calculated.");
   flushImpactVectors ();
}//setupRIITable
```

```
public void setupCMLTable (String cmlImpacts,
                                        String cmlNormalisation,
                                        String cmlNormalisationII)
{
   if (substances.size() > 0) {
         getImpacts (cmlImpacts);
         calculateCharacterization (cmlNormalisation);
         calculateNormalisation (cmlNormalisationII);
         trimNumbers ();
         fillCMLMatrix ();
         returnCharsChartValues();
         returnNormal1ChartValues();
         returnNormal2ChartValues();
   } else System.out.println("You need to set up the LCI before " +
               "the impacts could be calculated.");
   flushImpactVectors ();

}//setupCMLTable

public void setupEI99Table (String ecoImpacts,
                                        String ecoNormalisation,
                                        String ecoWeighting)
{
   if (substances.size() > 0) {
         getImpacts (ecoImpacts);
         calculateCharacterization (ecoNormalisation);
         calculateWeighting (ecoWeighting);
         trimNumbers ();
         fillEI99Matrix ();
         returnCharsChartValues();
         returnNormal1ChartValues();
         returnWeightChartValues();
   } else System.out.println("You need to set up the LCI before " +
               "the impacts could be calculated.");
   flushImpactVectors ();

}//setupEI99Table
```

```
   /*************** RII CALCULATOR ****************/

   private void setupRIIvectors (double z) {
      double n=0;
      for (int i = 0; i < totals.size(); i++) {
            n = (Double.parseDouble(String.valueOf(cTotals.get(i))));
            riiTotals.add(new Double(n*z));
            riiCompartments.add(compartments.get(i));
            riiSubstances.add(substances.get(i));
            riiUnits.add(cUnits.get(i));
      }
      System.out.println(riiTotals);
   }

   private void riiCalcGetImpacts () {

      for (int p = 0; p < sr; p++) {
            query = "SELECT Compartment, Category, Substance, Total, Unit
FROM [RII Impacts]";
            query += " WHERE Substance = '" + (String)riiSubstances.get(p);
            query += "' AND Compartment = '" +
(String)riiCompartments.get(p);
            query += "'";

            dataBase.executeQuery(query, false);
            ss = dataBase.getRowCount();
            for (int q = 0; q < ss; q++) {
                  impcom.add(dataBase.getValueAt(q,0));
                  impcat.add(dataBase.getValueAt(q,1));
                  impsub.add(dataBase.getValueAt(q,2));
                  imptot.add(dataBase.getValueAt(q,3));
                  impuni.add(dataBase.getValueAt(q,4));
                  invtot.add(riiTotals.get(p));
            }//for q

      }//for p

   }
```

```
   private void riiCalcCharNorm (String table) {

      double charVal;
      double normVal;
      query = "SELECT Category, Unit, Factor FROM [" + table + "] ";
      dataBase.executeQuery(query, false);
      int nrOfCats = dataBase.getRowCount();

      //Write categories to a vector & their units.
      for (int i = 0; i < nrOfCats; i++) {
            categories.add(dataBase.getValueAt(i,0));
            catunits.add(dataBase.getValueAt(i,1));
            normfact.add(dataBase.getValueAt(i,2));
      }

      //Now, sum all values of the same category.
      for (int j = 0; j < categories.size(); j++)
      {
            int counter = 0;
            charVal = 0;
            normVal = 0;
            for (int i = 0; i < imptot.size(); i++)
            {
                  if
(String.valueOf(categories.get(j)).compareTo(String.valueOf(impcat.get(i)))
== 0)
                  {     //System.out.println(impsub.get(i));
                        charVal +=
(Double.parseDouble(String.valueOf(imptot.get(i))) *

   Double.parseDouble(String.valueOf(invtot.get(i))));
                        counter++;
                  }//if
            }//for i
            normVal= Double.parseDouble(String.valueOf(normfact.get(j)));
            normVal*=charVal;
            chars.add(new Double(charVal));
            normal.add(new Double(normVal));
      }
```

```
      System.out.println("Characterisation and Normalisation values
calculated for RII calculator.");
   }//calculateCharacterization

   private void riiCalcChart ()
   {
      riiGrouping = new float[4];
      double landVal=0, waterVal=0, airVal=0, mineVal=0;
      String query = "SELECT Category, Group FROM [RII Grouping]";
      dataBase.executeQuery(query, false);
      for (int i = 0; i < categories.size(); i++) { //For the 4 distinct
groups
            if
(String.valueOf(dataBase.getValueAt(i,1)).compareTo("Land")==0)
                  landVal +=
(Double.parseDouble(String.valueOf(normal.get(i))));
            else if
(String.valueOf(dataBase.getValueAt(i,1)).compareTo("Water")==0)
                  waterVal +=
(Double.parseDouble(String.valueOf(normal.get(i))));
            else if
(String.valueOf(dataBase.getValueAt(i,1)).compareTo("Air")==0)
                  airVal +=
(Double.parseDouble(String.valueOf(normal.get(i))));
            else if
(String.valueOf(dataBase.getValueAt(i,1)).compareTo("Mined abiotic")==0)
                  mineVal +=
(Double.parseDouble(String.valueOf(normal.get(i))));
      }
      //Add grouping values to vectors:
      riiGrouping[0] = (Float.parseFloat(Text.writeDouble(landVal, 0, 3)));
      riiGrouping[1] = (Float.parseFloat(Text.writeDouble(waterVal, 0,
3)));
      riiGrouping[2] = (Float.parseFloat(Text.writeDouble(airVal, 0, 3)));
      riiGrouping[3] = (Float.parseFloat(Text.writeDouble(mineVal, 0, 3)));
      System.out.println("Chart values completed for RII Calculator");

   }//riiCalcChart

   public void riiCalculator (String normalType,
```

```
                                        double waterX,
                                        double electX,
                                        double steamX,
                                        double fuelX,
                                        double wasteX)
{
        flushLciVectors();
        flushImpactVectors();
        riiTotals.removeAllElements();
        riiSubstances.removeAllElements();
        riiUnits.removeAllElements();
        riiCompartments.removeAllElements();

        if (waterX!=0.0) {
            getLCIOutputs("Water LCI");
            convertUnits();
            setupRIIvectors(waterX);
        }
        if (steamX!=0.0) {
            getLCIOutputs("Steam LCI");
            convertUnits();
            setupRIIvectors(steamX);
        }
        if (fuelX!=0.0) {
            getLCIOutputs("Diesel Fuel LCI");
            convertUnits();
            setupRIIvectors(fuelX);
        }
        if (wasteX!=0.0) {
            getLCIOutputs("Waste Treatment LCI");
            convertUnits();
            setupRIIvectors(wasteX);
        }
        if (riiTotals.size()!=0) {
            riiCalcGetImpacts ();
            riiCalcCharNorm (normalType);
            riiCalcChart ();
        }

        //Because Normalisation for Electricity is
```

```
            //calculated by SALCA Region 3 for all inputs
            //this must be done seperate.
            if (electX!=0.0) {
                  double land=0, water=0, air=0, mine=0;
                  if (riiTotals.size()!=0) {
                        land = riiGrouping[0];
                        water = riiGrouping[1];
                        air = riiGrouping[2];
                        mine = riiGrouping[3];
                  }
                  flushLciVectors ();
                  flushImpactVectors ();
                  riiTotals.removeAllElements ();
                  riiSubstances.removeAllElements ();
                  riiUnits.removeAllElements ();
                  riiCompartments.removeAllElements ();
                  getLCIOutputs("Electricity LCI");
                  convertUnits ();
                  setupRIIvectors(electX);
                  riiCalcGetImpacts ();
                  riiCalcCharNorm ("RII Normalisation (SALCA Region 3)");
                  riiCalcChart ();
                  riiGrouping[0] += land;
                  riiGrouping[1] += water;
                  riiGrouping[2] += air;
                  riiGrouping[3] += mine;
            }
            System.out.println(riiGrouping[0]);
            System.out.println(riiGrouping[1]);
            System.out.println(riiGrouping[2]);
            System.out.println(riiGrouping[3]);
   }

}//class
```