21

BASIC PROGRAMMING EXAMPLES

I am a partner and co-founder of Integrasoft. We have dedicated the company to highly distributed computing environments in the financial sector since its inception in 1997. In 2001, we focused our collective experience on bridging the chasm of data management in relation to the grid compute environments. Many of the principles discussed in this book are a direct result of our work and are addressed in our product known as the Integrasoft Grid Fabric (IGF), a distributed data management system for the compute grid.

IGF is a purely data management system that is designed to sit on top of any vendor's "data distribution engine." Currently, IGF's data distribution engine is a distributed cache that spans the entire grid space.

IGF supports data regions and the various data management policies discussed below. Three working code examples developed using the IGF product are described below. The first two are simple "HelloWorld" examples that show fine and coarse granularity of data atoms in a data grid. The latter is more involved, covering a random-number surface used in a Monte Carlo simulation. I hope that these examples will be found useful in further expansion of the theories or application that we have covered.

Distributed Data Management for Grid Computing, by Michael Di Stefano
Copyright © 2005 John Wiley & Sons, Inc.

HELLOWORLD EXAMPLE

Coarse Granularity

This example shows a coarse-grained object whose entire data attributes will be stored in the IGF data grid as a single IGF data atom. Note that the business logic of "IGFCustomObject" does not need any modifications to be able to support the data grid. Through its inheritance from IGFCacheable, the IGFCustomObject becomes an entity that can be stored in any IGF data grid collection. When placed in an IGF collection, it will be assigned a "logical" name that can be used by the collection's "get( )" command for later retrieval. Since IGFCustomObject is an IGFCacheable entity, it through any IGF Collection's "put( )" command can be placed into the collection. The collection and all its entities are live data atoms in the data grid. By default, the collection and its entities are now under the data grid's management policies, which include the regionalization, synchronization, replication, and distribution policies. Any application connected to the data grid can open the collection and via the collection's "get( )" command can "query the data grid" for any individual entity in the collection by name, for example. Because of the coarse granularity of the data grid atom, IGFCustomObject is "queryable" as a single data point (data atom); applications cannot see into the IGFCustomObject and thus query the data grid for it by any of its data attributes of "m_yield," "m_maturity," and "m_pointName." The object is opaque to the outside world; this is what is meant by "coarse granularity." However, once retrieved from the data grid through the get( ) command, the application can operate on the object and its attributes as any other normal object. Any resulting changes of state of the object are reflected in the data grid once it is "put" back (updated) into the collection.

The following is an example of the code required to define ("coarse data atom"), store ("writer program"), and access ("reader program") a coarse data atom:

Coarse Data Atom

```
1   import com.integrasoftware.GridFabric.Cartridges.
    Framework.model.*;
2   import com.integrasoftware.GridFabric.Cartridges.Basic.
    DataCartridge.model.*;
3   import com.integrasoftware.GridFabric.Cartridges.Basic.
    DataCartridge.*;
4   import com.integrasoftware.GridFabric.Integration.Data.
    Framework.model.*;
5   /**
6   *Title: IGFCustomObject<
7   *Description: This custom object demonstrates the ability
    to store leaf nodes with both IGF
8   *and generic Java values. Leaf nodes can be inserted in a
    variety of IGF collections
9   *Copyright: Copyright (c) 2003
10  *Company: Integrasoft LLC
```

```
11  *@version 1.0
12  */
13
14  public class IGFCustomObject implements IGFCacheable
15        //Leaf nodes MUST implement IGFCacheable
16  {
17  public IGFBasicFloat m_yield;
18  public java.util.Date m_maturity;
19  public String m_pointName;
20
21  public IGFCustomObject()  //These are empty for now
22  {
23  }
24
25  public IGFCachePolicy cachePolicy()  //These are empty for now
26  {
27  return null;
28  }
29
30  public void cachePolicy(IGFCachePolicy policy)//These are
    empty for now
31  {
32  }
33 }
```

Writer Program

```
1   import com.integrasoftware.GridFabric.Cartridges.Basic.
    DataCartridge.model.*;
2   import com.integrasoftware.GridFabric.Cartridges.Basic.*;
3   import com.integrasoftware.GridFabric.Cartridges.Basic.
    DataCartridge.*;
4   import com.integrasoftware.GridFabric.Cartridges.Frame-
    work.control.*;
5   import java.util.*;
6
7   import java.io.*;
8
9   /**
10  *Title: IGFObjectGraph3Writer
11  *Description: This example demonstrates the creation of a
    custom leaf object, a leaf can
12  *contain any number and type of attributes as long as it
    implements IGFCacheable
13  *Copyright: Copyright (c) 2003
14  *Company: Integrasoft LLC
15  *@version 1.0
16  */
```

```
17
18 public class IGFObjectGraph3Writer {
19 public IGFObjectGraph3Writer()
20   {
21   }
22
23 public static void main (String argv[])
24 {
25 java.util.Properties props = new java.util.Properties();
26
27
28 //Create a cartridge on region called "Default", pass in
   properties in event more configuration parameters are needed
29 //
30 IGFCartridge cart = IGFBasicCartridgeFactory.instance().
   create("Default",
31   props);
32
33 //Obtain a handle to the Data Cartridge associated w/Region
   "Default"
34 IGFBasicDataCartridge dCart = (IGFBasicDataCartridge)
   cart.data();
35
36 //Construct a new List associated w/Region "Default"
37 IGFBasicList testList=(IGFBasicList) dCart.
   obtainCacheableEntityNamed(
38   "IGFBasicList");
39
40 //Iterate through list and populate a custom leaf object
41   for (int i = 1;i < 10; i++) {
42
43 //Create a leaf object
44 IGFCustomObject o = new IGFCustomObject();
45
46 //Create an IGF Float associated w/Region "Default"
47 IGFBasicFloat mYield=(IGFBasicFloat)dCart.
   obtainCacheableEntityNamed("IGFBasicFloat");
48
49 //Populate IGFFloat
50 mYield.setValue((float)Math.random()*100);
51
52 o.m_yield = mYield;
53 o.m_maturity = new Date();
54 o.m_pointName = "3YR";
55
56 //Add custom leaf object to list
57 testList.add(o);
58
59 System.out.println("Index:"+i+"yield:"+o.m_yield+"mat:"+
```

```
60        o.m_maturity);
61    }
62
63    //Insert list into region "default"
64    dCart.putRoot("YieldCurve1", testList);
65  }
66
67 }
```

Reader Program

```
1   import com.integrasoftware.GridFabric.Cartridges.Basic.
    DataCartridge.model.*;
2   import com.integrasoftware.GridFabric.Cartridges.Basic.*;
3   import com.integrasoftware.GridFabric.Cartridges.Basic.
    DataCartridge.*;
4   import com.integrasoftware.GridFabric.Cartridges.Frame-
    work.control.*;
5   import java.util.*;
6
7   import java.io.*;
8
9   /**
10  *Title: IGFObjectGraph3Writer
11  *Description: This example reads IGFCacheable objects from
    the Data Grid
12  *Copyright: Copyright (c) 2003
13  *Company: Integrasoft LLC
14  *@version 1.0
15  */
16
17   public class IGFObjectGraph3Reader {
18
19    public IGFObjectGraph3Reader() {
20    }
21
22    public static void main (String argv[])
23      {
24      java.util.Properties props = new java.util.Properties();
25      IGFCartridge cart = IGFBasicCartridgeFactory.instance().
    create("Default",
26        props);
27      IGFBasicDataCartridge dCart=(IGFBasicDataCartridge)
    cart.data();
28
29      IGFBasicList testList = (IGFBasicList) dCart.getRoot
    ("YieldCurve1");
30      for (int i = 1; i < 10; i++)
31      {
```

```
32      IGFCustomObject o = (IGFCustomObject)testList.get(i);
33      System.out.println("Index:"+i+"yield:"+o.m_yield.
    getValue()+"mat:"+
34                  o.m_maturity);
35      }//for ()
36
37    }
38   }
```

Fine Granularity

Now let us investigate an example of a fine-grained data atom. The first thing to notice is that there is no IGFCustomObject that inherits from the IGFCacheable. In this situation the object to be data-grid-enabled is IGFObjectGraph1Writer, where some of its data attributes are natively data-grid-enabled. It is important to note that not all of an object's data attributes need to be data-grid-enabled. Those attributes that are data-grid-enabled "live" in the IGF data grid and are managed by its distributed data management policies. Those that are not data-grid-enabled will reside in the local heap of the process space of the IGFObjectGraph1Writer instance.

The IGFObjectGraph1Writer instance is "put" into the IGF data grid with the logical name of ROOTOBJECT, the name that will be used for later queries and retrieval. Some of the data attributes of the IGFObjectGraph1Writer are collection classes, maps, lists, and arrays. The other data attributes are basic, such as IGFInteger and IGFFloat. Any program that accesses the IGF data grid can get the IGFObjectGraph1Writer from the data grid and directly access any of the "data-grid-enabled attributes." Those programs can change or update the values of these attributes via their respective "put( )" operations, which will immediately take effect in the IGF data grid and can be accessed by any other program viewing or accessing the IGFObjectGraph1Writer "ROOTOBJECT" instance in the IGF data grid. The fact that the internal data attributes of the IGFObjectGraph1Writer are directly IGF data-grid-enabled means that access to these data attributes is direct and transparent to all on the IGF data grid. Therefore, the IGFObjectGraph1Writer is said to be a fine-grained data atom. A sample code for loading of the data atom into the data grid ("writer program") and retrieval of data atom ("reader program") is

Writer Program

```
1   import com.integrasoftware.GridFabric.Cartridges.Basic.
    DataCartridge.model.*;
2   import com.integrasoftware.GridFabric.Cartridges.Basic.*;
3   import com.integrasoftware.GridFabric.Cartridges.Basic.
    DataCartridge.*;
4   import com.integrasoftware.GridFabric.Cartridges.
    Framework.control.*;
```

```
5  import com.integrasoftware.GridFabric.Integration.Data.
   Framework.model.*;
6  import java.util.*;
7
8  import java.io.*;
9
10  /**
11 *Title: IGFObjectGraph1Writer
12 *Description: This object demonstrates the creation and
   population of a custom Root Object.
13 *Root Objects MUST inherit from IGFBasicObject, and can
   contain both native as well as IGF types
14 *Copyright: Copyright (c) 2003
15 *Company: Integrasoft LLC
16 *@version 1.0
17 */
18
19 //Must inherit from IGFBasicObject
20 public class IGFObjectGraph1Writer extends IGFBasicObject
   {
21 public IGFBasicMap m_map1, m_map2;
22 public IGFBasicList m_list1, m_list2;
23 public IGFBasicInt m_int1;
24 public IGFBasicFloat m_float1;
25 public IGFBasicNativeDoubleArray m_dArray;
26 public int foo;
27
28 public IGFObjectGraph1Writer() {
29 super();
30 }
31
32 public static void main(String[] args) {
33  try {
34   IGFObjectGraph1Writer graph1 = new
   IGFObjectGraph1Writer();
35   java.util.Properties props = new java.util.Properties();
36
37
38   //Create a cartridge on region called "Default", pass in
   properties in event more configuration parameмeters are
   needed
39   IGFCartridge cart = IGFBasicCartridgeFactory.instance().
   create("Default",
40     props);
41
42   //Obtain a handle to the Data Cartridge associated
   w/Region "Default"
43   IGFBasicDataCartridge dCart = (IGFBasicDataCartridge)
   cart.data();
```

```
44
45   //Start populating graph1 attributes
46   graph1.foo=-90;
47
48   //Construct new objects for attributes of graph1
49   graph1.m_map1 =
50         (IGFBasicMap) dCart.obtainCacheableEntityNamed
     ("IGFBasicMap");
51
52   graph1.m_map2 =
53         (IGFBasicMap) dCart.obtainCacheableEntityNamed
     ("IGFBasicMap");
54
55   graph1.m_list1 =
56         (IGFBasicList) dCart.obtainCacheableEntityNamed
     ("IGFBasicList");
57
58   graph1.m_list2 =
59         (IGFBasicList) dCart.obtainCacheableEntityNamed
     ("IGFBasicList");
60
61   graph1.m_int1 =
62         (IGFBasicInt) dCart.obtainCacheableEntityNamed
     ("IGFBasicInt");
63
64   graph1.m_float1 =
65         (IGFBasicFloat) dCart.obtainCacheableEntityNamed
     ("IGFBasicFloat");
66
67   IGFBasicDouble dbl2 =
68         (IGFBasicDouble) dCart.obtainCacheableEntityNamed
     ("IGFBasicDouble");
69
70   graph1.m_dArray = (IGFBasicNativeDoubleArray)
71               dCart.obtainCacheableEntityNamed
     ("IGFBasicNativeDoubleArray");
72
73   //populate ints/floats/doubles
74   graph1.m_int1.setValue(94);
75   graph1.m_float1.setValue((float) 95.443);
76   dbl2.setValue(97.998);
77
78   //add int/float into list1
79   graph1.m_list1.add(graph1.m_int1);
80   graph1.m_list1.add(graph1.m_float1);
81
82   //add int/double into list2
83   graph1.m_list2.add(graph1.m_int1);
84   graph1.m_list2.add(dbl2);
```

```
85
86     //add double/int into map1
87     graph1.m_map1.put("TESTDOUBLE", graph1.m_float1);
88     graph1.m_map1.put("TESTINT", graph1.m_int1);
89
90     //add double into map2
91     graph1.m_map2.put("TESTDOUBLE", dbl2);
92
93     //populate native double array
94     for (int i = 1; i < 100; i++)
95      graph1.m_dArray.putAt(i, Math.log(10.332*i));
96
97     //insert object as root node into region "default"
98      dCart.putRoot("ROOTOBJECT", graph1);
99   } catch (Exception ex)
100  {
101  ex.printStackTrace();
102  }
103 }
104
105 public IGFCachePolicy cachePolicy()
106 {
107  return null;
108 }
109
110 public void cachePolicy(IGFCachePolicy policy)
111 {
112 }
113
114 }
```

Reader Program

```
1   import
    com.integrasoftware.GridFabric.Cartridges.Basic.DataCar-
    tridge.model.IGFBasicObject;
2
3   /**
4   *Title: IGFObjectGraph1Reader
5   *Description: This object demonstrates how to read a Custom
    Root Node and traverse it
6   *Copyright: Copyright (c) 2003
7   *Company: Integrasoft, LLC
8   *@version 1.0
9   */
10
11  import com.integrasoftware.GridFabric.Cartridges.Basic.
    DataCartridge.model.*;
12  import com.integrasoftware.GridFabric.Cartridges.Basic.*;
```

```
13 import com.integrasoftware.GridFabric.Cartridges.Basic.
   DataCartridge.*;
14 import com.integrasoftware.GridFabric.Cartridges.Frame-
   work.control.*;
15 import java.util.*;
16
17
18 import
   com.integrasoftware.GridFabric.Cartridges.Basic.DataCar-
   tridge.model.IGFBasicObject;
19
20
21 public class IGFObjectGraph1Reader extends IGFBasicObject {
22    public IGFObjectGraph1Reader() {
23    }
24     public static void main(String[] args) {
25       IGFObjectGraph1Writer graph1;
26       java.util.Properties props=new java.util.Properties();
27
28     //Create a cartridge on region called "Default", pass
   in properties in event more configuration parameters are
   needed
29     IGFCartridge cart=IGFBasicCartridgeFactory.instance().
   create("Default", props);
30
31     //Obtain a handle to the Data Cartridge associated
   w/Region "Default"
32     IGFBasicDataCartridge dCart=(IGFBasicDataCartridge)-
     cart.data();
33
34     //get root custom object from region "Default"
35     graph1=(IGFObjectGraph1Writer)dCart.getRoot
   ("ROOTOBJECT");
36
37     //get a previously inserted int from m_map1
38     IGFBasicInt mapInt=(IGFBasicInt)graph1.m_map1.get
     ("TESTINT");
39
40     //get a previously inserted double from m_map1
41     IGFBasicFloat mapDouble=(IGFBasicFloat)graph1.m_map1.
   get("TESTDOUBLE");
42
43     //get a previously inserted double from m_map2
44   IGFBasicDouble mapDouble2=(IGFBasicDouble)graph1.m_map2.
   get("TESTDOUBLE");
45
46     //get a previously inserted int from m_list1
47   IGFBasicInt listInt=(IGFBasicInt)graph1.m_list1.get(1);
48
```

```
49      //test for quality to ensure that the objects are actu-
    ally identical
50      boolean cmpAre=(mapInt.equals(graph1.m_int1) && mapDou-
      ble.equals(graph1.m_float1));
51
52      double [] dblArr = new double[100];
53      for (int i = 1; i < dblArr.length; i++)
54      {
55        dblArr[i] = graph1.m_dArray.getAt(i);
56        System.out.println("dblArr[+"+i+"]: "+dblArr[i]);
57      }
58
59
60      for (int i = 1; i < graph1.m_list1.size()+1;i++)
61      {
62        System.out.println("List1:"+graph1.m_list1.get(i).
    toString());
63      }
64
65      for (int k = 1; k < graph1.m_list2.size()+1; k++)
66      {
67        System.out.println("List2:"+graph1.m_list2.get(k).
    toString());
68      }
69
70    }
71
72 }
```

RANDOM-NUMBER SURFACE EXAMPLE

This example is a random-number surface used in a Monte Carlo simulation. Traditionally, Monte Carlo simulations are run from start to finish before any results are visible. In grid-enabled environment, the Monte Carlo simulation can be optimized by slicing the simulation across the compute grid as worklets with small input data sets generating large amounts of interim data to ultimately return a simple result(s). One of the interim data sets is a random-number surface. Providing a "schema" for such interim surfaces and data-grid-enabling them yields two optimizations: further parallelization through finer-grained compute worklets to build the interim surfaces and the reuse of previously built surfaces. The first optimization allows what was one worklet, "build random-number surface," to become many worklets, each contributing to a single entry of data point of the random-number surface.

The second optimization enables data reuse from one Monte Carlo simulation to the next. For example, a random-number surface "FooBar" built in one simulation can be reused in subsequent simulations, thus eliminating the need for additional computation cycles, which would be required to rebuild the data surfaces. In

order for other applications to reuse the data surface, the IGF data grid must allow the application to "query" the data surface that it needs.

If a data surface can be queried, then not only can subsequent Monte Carlo simulations benefit from the preexisting data surfaces, which are stored in the IGF data grid, but also any "observer" program can read and monitor data surfaces as they are being built in real time. Thus, this creates a new class of applications instead of batch applications, which used to run overnight. For example, a running Monte Carlo simulation can be monitored, if diverging it can be terminated in the middle of its processing. Conversely, if convergence is satisfied prior to completion, it can be terminated early. This is yet another form of optimization offered through the smart utilization of compute resource.

All the optimizations highlighted above and the possibility of new business observer programs are possible only through data-grid-enabling a Monte Carlo simulation. The example random-number surface illustrated below is a common part of any Monte Carlo simulation:

```
1   /**
2    *Title: IGFObjectGraph1Reader
3    *Description: This object demonstrates how to read a
    Custom Root Node and traverse it
4     *Copyright: Copyright (c) 2003
5     *Company: Integrasoft, LLC
6     *@version 1.0
7     */
8   \\RandomField\\Harness\\SurfaceDemo\\IGFEuropeanCallOp-
    tionPopulator.java
9   package com.integrasoftware.GridFabric.Cartridges.Random-
    Field.Harness.SurfaceDemo;
10
11  import com.integrasoftware.GridFabric.Cartridges.Frame-
    work.control.IGFPopulator;
12  import com.integrasoftware.GridFabric.Cartridges.Frame-
    work.control.IGFDataCartridge;
13  import com.integrasoftware.GridFabric.Cartridges.Random-
    Field.DataCartridge.IGFRandomFieldDataCartridge;
14  import com.integrasoftware.GridFabric.Integration.Data.
    Framework.model.IGFCacheable;
15  import com.integrasoftware.GridFabric.Cartridges.Frame-
    work.model.IGFScenario;
16  import com.integrasoftware.GridFabric.Cartridges.Frame-
    work.model.IGFPathList;
17  import com.integrasoftware.GridFabric.Cartridges.Frame-
    work.model.IGFDataPoint;
18  import com.integrasoftware.GridFabric.Cartridges.Frame-
    work.model.IGFPath;
19  import com.integrasoftware.GridFabric.Cartridges.Random-
    Field.IGFRandomFieldFactory;
```

```
20 import com.integrasoftware.GridFabric.Cartridges.Frame-
   work.model.IGFDataPointList;
21 import com.integrasoftware.GridFabric.Cartridges.Frame-
   work.control.IGFCartridge;
22 import javax.swing.JFrame;
23 import java.awt.GridLayout;
24 import java.awt.Dimension;
25 import java.awt.Color;
26 import javax.swing.JPanel;
27 import com.klg.jclass.chart3d.*;
28 import com.klg.jclass.chart3d.j2d.*;
29 import com.klg.jclass.chart3d.data.*;
30
31 public class IGFEuropeanCallOptionPopulator implements
   IGFPopulator, Service
32 {
33   private IGFCartridge m_cartridge;
34   private String m_scenarioName;
35   private int m_dimensionality;
36   private double m_strikePrice;
37   private double m_timeInterval;
38   private double m_assetPrice;
39   private double m_sigma;
40   private double m_contIR;
41   private double m_divYield;
42   private double m_timeStep;
43   private double m_dT;
44   private double m_nudt;
45   private double m_sigsdt;
46   private double m_lnAssetPrice;
47   private JobContext m_context;
48
49   public IGFEuropeanCallOptionPopulator()
50   {
51     m_strikePrice = 100;
52     m_timeInterval = 1;
53     m_assetPrice = 100;
54     m_sigma = 0.2;
55     m_contIR = 0.06;
56     m_divYield = 0.03;
57     m_timeStep = 100;
58     m_dT=m_timeInterval/m_timeStep;
59     m_nudt = (m_contIR-m_divYield-1/2*m_sigma*m_sigma)*m_dT;
60     m_sigsdt = m_sigma*Math.sqrt(m_dT);
61     m_lnAssetPrice = Math.log(m_assetPrice);
62   }
63
64 public static final void main(String argv[])
65 {
```

```
66  IGFEuropeanCallOptionPopulator pop = new IGFEuropeanCall
    OptionPopulator();
67    pop.connectToRandomFieldWith(10,"MonteCarlo","Random-
    FieldCartridge");
68    pop.cartridge().compute().populator(pop);
69    for (int i = 0; i < 10; i++)
70      pop.populate();
71  }
72
73  public void populate()
74  {
75
76    IGFRandomFieldDataCartridge dCart = (IGFRandomFieldData-
    Cartridge)cartridge().data();
77    IGFScenario scene = dCart.scenarios().findScenario
    (scenarioName());
78    IGFPathList paths=scene.paths();
79    IGFPath path=dCart.makePath();
80    paths.add((IGFCacheable)path);
81    IGFDataPointList points = path.points();
82    for (int i=0; i < m_timeStep-1; i++)
83    {
84      IGFDataPoint aPointInPath = dCart.makeDataPoint();
85      double stdNormal = Math.random();
86      aPointInPath.point().put(0, stdNormal);
87      points.add(aPointInPath);
88      }
89
90    int lastPath = paths.size();
91    java.util.Vector vector=new java.util.Vector();
92    vector.addElement(new Integer(lastPath));
93    dCart.generateEventForNameSpace("STR", vector);
94
95  }
96
97  public IGFCartridge cartridge()
98  {
99    return m_cartridge;
100   }
101
102   public void cartridge(IGFCartridge cartridge)
103   {
104     m_cartridge = cartridge;
105   }
106
107   public void connectToRandomFieldWith(int dimensionality,
      String scenarioName, String cartridgeName)
108   {
109     java.util.Properties props = new java.util.Properties();
```

```
110   props.setProperty(IGFDataPoint.DIMENSION, String.
    valueOf(dimensionality));
111   scenarioName(scenarioName);
112   dimensionality(dimensionality);
113   cartridge(IGFRandomFieldFactory.instance().create
    (cartridgeName, props));
114 }
115
116 public String scenarioName()
117 {
118 return m_scenarioName;
119 }
120
121
122 public void scenarioName(String sceneName)
123 {
124   m_scenarioName=sceneName;
125 }
126
127 public int dimensionality()
128 {
129   return m_dimensionality;
130 }
131
132 public void dimensionality(int dim)
133 {
134   m_dimensionality = dim;
135 }
136
137
138 public void invoke(JobContext arg0, InputMessage arg1,
    OutputMessage arg2) throws Exception, SystemException
139 {
140   connectToRandomFieldWith(10,"MonteCarlo","RandomField-
    Cartridge");
141   m_context = arg0;
142   cartridge().compute().populator(this);
143   for (int i=0; i < 10; i++)
144   populate();
145   StringBuffer sb_test = new StringBuffer();
146   String a = ((TextInputMessage)arg1).get();
147   //perform a simple transformation (to upper case) and
    append its task ID
148   sb_test = sb_test.append(a.toUpperCase()+"... Task
    ID:"+arg1.getTaskID().getValue());
149   ((TextOutputMessage)arg2).set(sb_test.toString());
150 }
151
152 }
```