W E L C O M E

An Addition to the Emerald Programming Language

User Guide

Created by
Olav Johan Myklestad Ekblom
and
Gaute Svanes Lunde

1 Introduction

This document is a guide for users of the Emerald programming language, explaining the use of the WELCOME language mechanism. The mechanism enables running programs to merge by exchanging object references, and allows for disjointed Emerald nodes to become acquainted.

The `welcome` operator takes an expression which evaluates to a type as its operand. The expression is blocking until a welcomable object conforming to the specified type is moved onto the local node, and returns a reference to that object. An object can be made welcomable by prefixing the `welcomable` keyword at its creation.

2 The Catcher and the Thrower

The basics of WELCOME language mechanism is best shown with the catcher and thrower programs, shown below. The catcher program welcomes an object conforming to the type `Catchable` and invokes the operation `sayHi` when such an object arrives. The thrower program moves a welcomable object conforming to the welcomed type to the node where the catcher resides. In the below examples we have modified the **Kilroy** program to act as the thrower.

The catcher program

```
1  const Catchable ← typeobject Catchable
2      op sayHi
3  end Catchable
4
5  const catcher ← object catcher
6      process
7          const catched ← welcome Catchable
8          catched.sayHi
9      end process
10 end catcher
```

The thrower program

```
1  const thrower ← welcomable object Kilroy
2      const me ← locate self
3      const all ← me$activeNodes
4
5      export operation sayHi
6          (locate self)$stdout.putstring["Kilroy was here\n"]
7      end sayHi
8
9      process
10         for n in all
11             move self to n$theNode
12         end for
13     end process
14 end Kilroy
```

3 Node Discovery

The WELCOME language mechanism allows for disjoint node graphs to be connected using the `welcome` expression. To discover a nearby previously unknown node, one can use the `setDiscoveredNodeEventHandler` operation of the `Node` object to be notified when such a node presents itself. The `setDiscoveredNodeEventHandler` takes a `Handler` as its argument, which needs to export the two operations `NodeUp` and `NodeDown` (similarly to the `setNodeEventHandler`). If the discovered node welcomes an object from the local node, the node graphs are merged and a regular `NodeUp` event is fired if a regular node event handler is present.

The server/client programs below is an example of node discovery in Emerald using the WELCOME language mechanism. The client program waits for another node to come within range and moves an emissary object to that node. The server acts like a catcher and welcomes the emissary object, merging the two node graphs.

The node discovery client program

```
const Serviceable ← welcomable class Serviceable
    export operation service
        (locate self)$stdout.putstring["I feel serviced\n"]
    end service
end Serviceable

const client ← object client
    const home ← locate self

    process
        home$discoveredNodeEventHandler ← object handlerObject
            export operation nodeUp[n : Node, t : Time]
                const serviceObject ← Serviceable.create
                move serviceObject to n
            end nodeUp

            export operation nodeDown[n : Node, t : Time]
            end nodeDown
        end handlerObject
    end process

end client
```

The node discovery server program

```
const Serviceable ← typeobject Serviceable
    op service
end Serviceable

const server ← object server
    process
        loop
            const toServe ← welcome Serviceable
            toServe.service
        end loop
    end process
end server
```

4 Edge Cases

Due to the distributed nature of the Emerald language, there are several edge cases to account for when using the WELCOME language mechanism. Below is a list of the most important edge cases:

- **No objects welcomed** - No node graphs are merged, and invocations on the discovered `Node` results in failure. Attempts on moving an object across unmerged node graphs will result in the object staying were it is, using the "best-effort" property of the `move` statement. Fixing an object at an unmerged node will result in a failure.
- **Discovered node disappears** - When a discovered node moves out of range or becomes unavailable, it will trigger a `nodeDown` event given by the handler from `setDiscoveredNodeEventHandler`, even if the node graphs have already merged. Thus, the `nodeDown` event for discovered nodes should only handle discovered nodes moving out of range, and not handle regular nodes becoming unavailable.
- **Moving a welcoming object** - If an object has a process that is blocking on a welcome expression is itself moved, it will continue to block on the destination node. It will stop blocking when an object conforming to the specified type is moved to its current location.