GrabCaster Manual

1 CONTENTS

Overview

2 BEFORE YOU START

2.1 UNINSTALL PREVIOUS BETA VERSIONS

Uninstall the old beta GrabCaster installations before installing a new one.
To uninstall the previous version enter in Windows Add Remove Program and uninstall the previous GrabCaster versions.

2.2 FIREWALL

GrabCaster can be extended to use different Messages Storage Providers (MSP). The current version uses Microsoft Azure Event Hubs (https://azure.microsoft.com/en-gb/services/event-hubs/) and Redis Database (http://redis.io), other MSPs will be implemented in the next versions.

Event Hubs is a highly scalable publish-subscribe event ingestor that can intake millions of events per second so that you can process and analyse the massive amounts of data produced by your connected devices and applications. Microsoft Events Hubss uses HTTP and AMPQ protocol, AMQP 1.0 is an efficient, reliable, wire-level messaging protocol that you can use to build robust, cross-platform, messaging applications.

Microsoft Azure EventHubs using AMQP requires the following firewall policies enabled:

Port number	Comments
80	HTTP port for certificate validation.
443	HTTPS
5671	Used to connect to Azure. If TCP port 5671 is unavailable, TCP port 443 is used.
9352	Used to push and pull data. If TCP port 9352 is unavailable, TCP port 443 is used.

These ports are normally opened by public internet providers.

For more information regarding Microsoft Service Bus settings, refer to this link

Redis is an open source (BSD licensed), in-memory **data structure store**, used as database, cache and message broker. It supports data structures such as strings, hashes, lists, sets,sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.

2.3 HIGH AVALABILITY AND CLUSTER

GrabCaster cans start as simple console application and Windows NT Service and it supports clustering and it can be also executed in a group of hosts that act like a single system and provide continuous uptime.

2.4 JSON

The GrabCaster configurations and settings are stored in JSON files. Be aware of the following reserved JSON characters:

- ‘ single quote
- ” quote
- \ backslash
- all control characters like \n \t
- [ ] to define an array
- { } to define an object

Important Note:
The use of any these characters in a JSON file can create an error. To use a special JSON character use the \ (backslash) escape character, for example to write a directory path use C:\\MyFolder instead of C:\MyFolder, to write “quote use \”. For more information refer to the official JSON web site http://json.org.

2.4.1 JSON Editors

Strong JSON skills are NOT required to edit GrabCaster files. Below are some recommended tools.

2.4.1.1 Notepad++ (Recommended - Free)

Notepad++ is a free source code editor and Notepad replacement that supports several languages. Running in the MS Windows environment. Its use governed by GPL License.

https://notepad-plus-plus.org/

2.4.1.2 Google Chrome JSON Editor (Free)

Perfect to view, edit and format JSON. It shows your data in an editable tree view and in a code editor. More information can be found here.

2.4.1.3 JSON Viewer (CodePlex - Free)

The JSON Viewer package is a set of 3 viewers available in the following flavours:

- A standalone viewer - JsonView.exe
- A plugin for Fiddler 2 (http://www.fiddler2.com/) - FiddlerJsonViewer.dll
- A visualizer for Visual Studio 2005 - JsonVisualizer.dll

More information can be found here

2.4.1.4 Altova XMLSpy (Commercial)

Altova XMLSpy provides a powerful JSON editor in addition to its XML editor with the following features:

- Advanced Text View with syntax colouring, source folding, book marking, and more.
- Intelligent JSON editing with element entry helpers.
- Grid/Tree View for viewing the structure/outline of the document.
- Graphical, drag-and-drop JSON editing in Grid/Tree View.
- One-click conversion between JSON <=> XML.

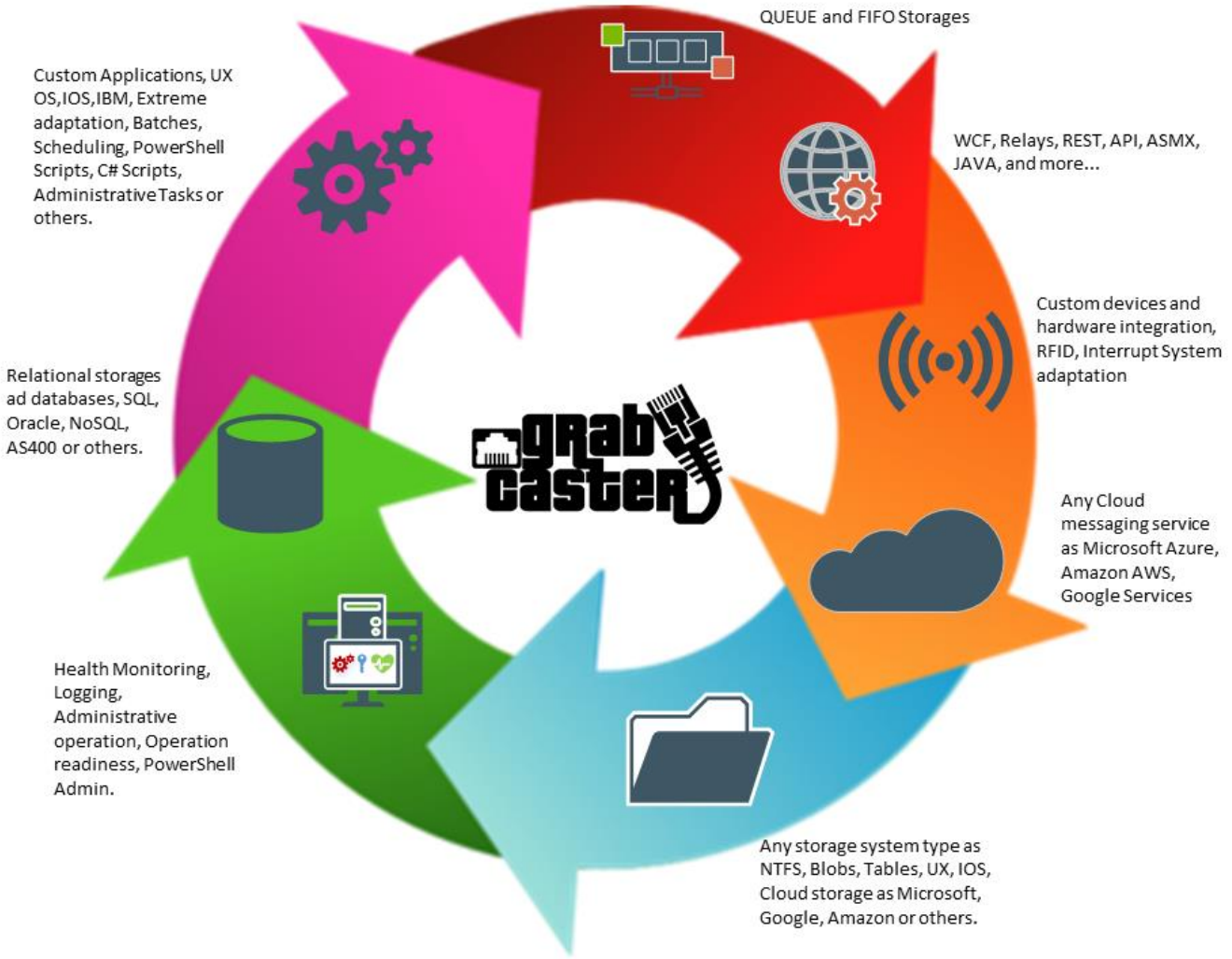
- JSON syntax checking and troubleshooting.
- And much more...

More information can be found here

3 GRABCASTER

GrabCaster is a smart framework focused into integrating technologies in fast and easy ways. The engine is completely extensible and customizable.

GrabCaster combines a complex event handling based on triggers and events with many different communication patterns to integrate and send data across the network. A trigger or event could be created using PowerShell or C#. GrabCaster doesn't necessary required a strong development skill to be used.



3.1 CORE FEATURES

Although GrabCaster contains different interesting features, below are the most important.

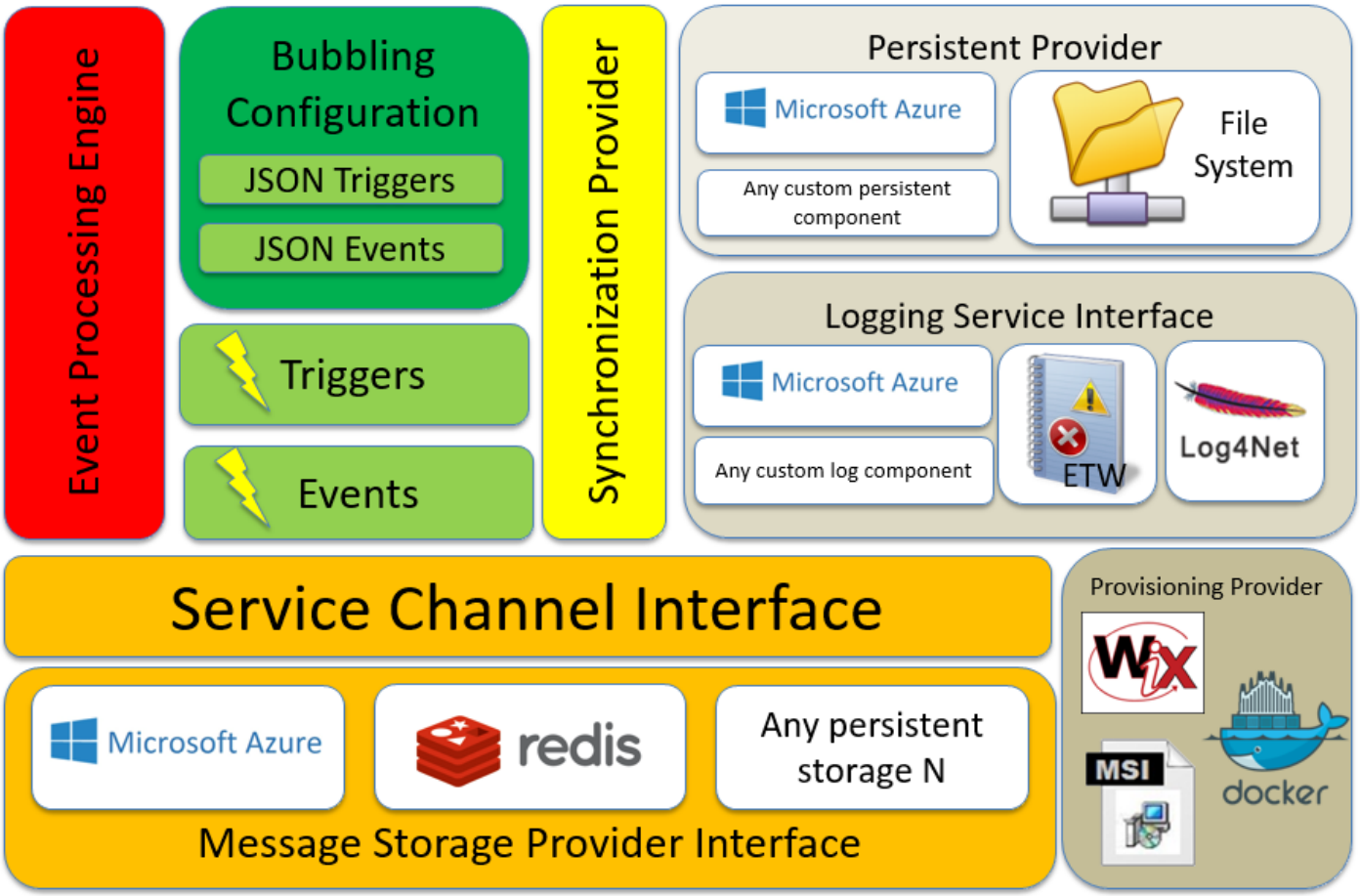
- Open Source

    - **GrabCaster framework** is open source that is available at GitHub. Available under **Reciprocal Public License 1.5**.
- For developer and end user
    - **GrabCaster SDK,** development templates and samples are available for developers to understand better how to extend it under **MIT License**.
- Fast installation
    - The installation is a very straightforward procedure.
- Easy configuration
    - The configuration is based on JSON and the engine exposes a HTTP REST interface to provide all the information about triggers and events.
- Extensible and customizable communication engine
    - GrabCaster engine is designed for extending to different communication frameworks. The current version only supports Microsoft Azure and Redis Database. Other engines will be implemented in next versions.
- Extensible and customizable logging engine
    - GrabCaster logging engine is fully extendable we can use our own different custom logging system. Current version uses Console, Windows Event Log and Microsoft Azure Event Hubs.
- Runnable in Console and as a Windows NT Service.
    - When starting GrabCaster you can choose to install as Windows NT Service or run as simple console program.
- Fast and easy scaling
    - Very fast and easy to create another GrabCaster clone in the same or remote machine.
- Fast and easy new integration implementation
    - Create a new trigger or event to integrate is very easy using a simple, straightforward development approach.
- Open Script Integration support
    - Possibility to create trigger and event using PowerShell and C# scripting. This would allow administrators creating a new integration implementation and not just developers.
- C# scripting rule engine
    - Rules can be executed remotely. The current rule engine is based on Roslyn and uses C# scripting.
- Multi-communication patterns
    - GrabCaster combines many different communication patterns together:
        - One and two way synchronous/asynchronous communication
        - Brokering messaging
        - One and two ways broadcasting
        - Peer2Peer
        - Publish/subscribe messaging
        - GrabCaster cans combine all of these together and more.
- Automatic synchronization and agile deployment
    - All GrabCaster points communicate together to synchronize triggers, events and remote configuration.

3.2 ARCHITECTURE

GrabCaster, with its flexible and scalable architecture, offers a powerful ready platform that facilitates development of a wide variety of integration solution patterns. The architecture is designed so that GrabCaster scales easily from low- to high-volume deployments.

The following figure shows a high-level overview of the GrabCaster architecture and capabilities:



The figure shows the key aspects of the architecture, including the following:

- Message Storage Provider Interface (MSPI)
- Service Channel Interface (SCI)
- Event Processing Engine (Core)
- Triggers and Events
- Bubbling Configuration
- Synchronization Provider
- Persistent Provider
- Logging Service Interface (LSI)
- Provisioning Provider

3.2.1 Message Storage Provider Interface (MSPI)

The MSPI manages the components used to communicate with the messaging persistent storage. The current version uses Microsoft Azure and Redis Database, about Microsoft Azure it uses different stacks as Event Hubs, Blobs and Table storage and next versions will provide other messaging persistent mechanisms.

3.2.2 Service Channel Provider Interface (SCPI)

The SCPI manages the communication between the GrabCaster's Core Engine and the MSPI.

3.2.3 Event Processing Engine (CORE)

The event-processing engine (EPE) is the core and it plays a central role in the stack as it manages the critical functionality of GrabCaster.

3.2.4 Triggers and events

Using Trigger and Events GrabCaster is able to integrate any kind of technology stack, application or device.
During the starting up GrabCaster looks in the [GrabCaster Installation Folder]\Bubbling\Triggers directory to check the triggers active.

For each configuration trigger file active GrabCaster starts a trigger.

> *Important note:*
> *GrabCaster provides an internal polling mode mechanism for the trigger which are not able to replicate this pattern but they need to do, for example external PowerShell scripts and external components.*

Any trigger contains a main "Execute method", during the execution the trigger uses all configuration file properties and demand to the GrabCaster EPE the trigger package context.

Important note:
GrabCaster maps any similar method and properties with the destination event, we can use this way to solve many interesting integration scenarios in easy way.

The GrabCaster EPE evaluates the numbers of events to execute and send the package to the SCI, the SCI will pass the package to the MSPI and the MSPI will resolve the Message Storage Component to use in order to send the package to the channels, if no channels or GrabCaster destination points as set the EPE executes the events locally without using the Service Channel Interface (SCI).

The GrabCaster destination point will receive the event package through the SCI which will demand it to the internal EPE to perform a deserialization, the EPE will perform an object mapping to map the object properties and the methods name results with the same name and it will execute the main event "Execute method".

The integration pattern used by GrabCaster is extremely transparent and extensible.

A trigger can execute multiple events and the events can also execute multiple events, a trigger and event map the same properties name each together locally or remotely.
The GrabCaster EPE is able to manage data context through the triggers and events with PowerShell scripting and C# scripting that means we can create triggers and event in PowerShell or using C# scripts, the possibility to create trigger and event using PowerShell or C# script is very useful for many reasons, no compilation, no real development environment need, an IT administrator can create a trigger using a known language script and PowerShell.

The trigger and event can drive an event execution using an internal rule engine based on C# script underpinned by Roslyn.
Using the code template and samples for triggers and events, the developer can construct an event-processing tree that scales from a simple to a complex business context in very simple way. The use of multiple logical events and rules in the event processing engine makes it possible to implement different communication patterns to solve a multitude of integration scenarios.

3.2.5 Bubbling Engine

The bubbling configuration provides the ability to configure and activate triggers and events. The Bubbling Engine exposes an HTTP REST service endpoint to execute trigger and retrieve the internal triggers and events configuration.

3.2.6 Synchronization Provider

One of the goals of the Synchronization Provider is to enable a communication between all the GrabCaster points, we can synchronize triggers and events between the GrabCaster points through any GrabCaster point REST API interface locally and remotely.

3.2.7 Persistent Provider

GrabCaster uses the internal persistent provider for many different operations, the current version implements Microsoft Azure Blob and local file system.

3.2.8 Logging Service Interface (LSI)

GrabCaster uses different logging patterns. The current version supports Console, Windows Event Log and Microsoft Event Hubs component.
We can implement our logging component and use it in GrabCaster, to implement a custom logging component refer to the Create new logging component chapter.

3.2.9 Service Channel

The service channel is an internal service which provides the communication between endpoints. The service uses a reflection mechanism to exchange the data between the engine and the Device Communication Component (DCM).

A MSPI is able to communicate and manage the communication between the GrabCaster points, the current GrabCaster version implements the Microsoft Azure MSPI and Redis Database next versions will implement other new MSPI.

4 HOW GRABCASTER WORKS

A GrabCaster Point (GCPoint) is essentially a generic service endpoint that is able to exchange data combining various communication patterns and across different channels.

When you install and configure a new GCPoint you create and configure a new Message Storage Provider (MSPI). As well as creating and identifying a new GrabCaster point ID and channel ID. The GCPoint will be able to subscribe itself to this channel or multiple channels.

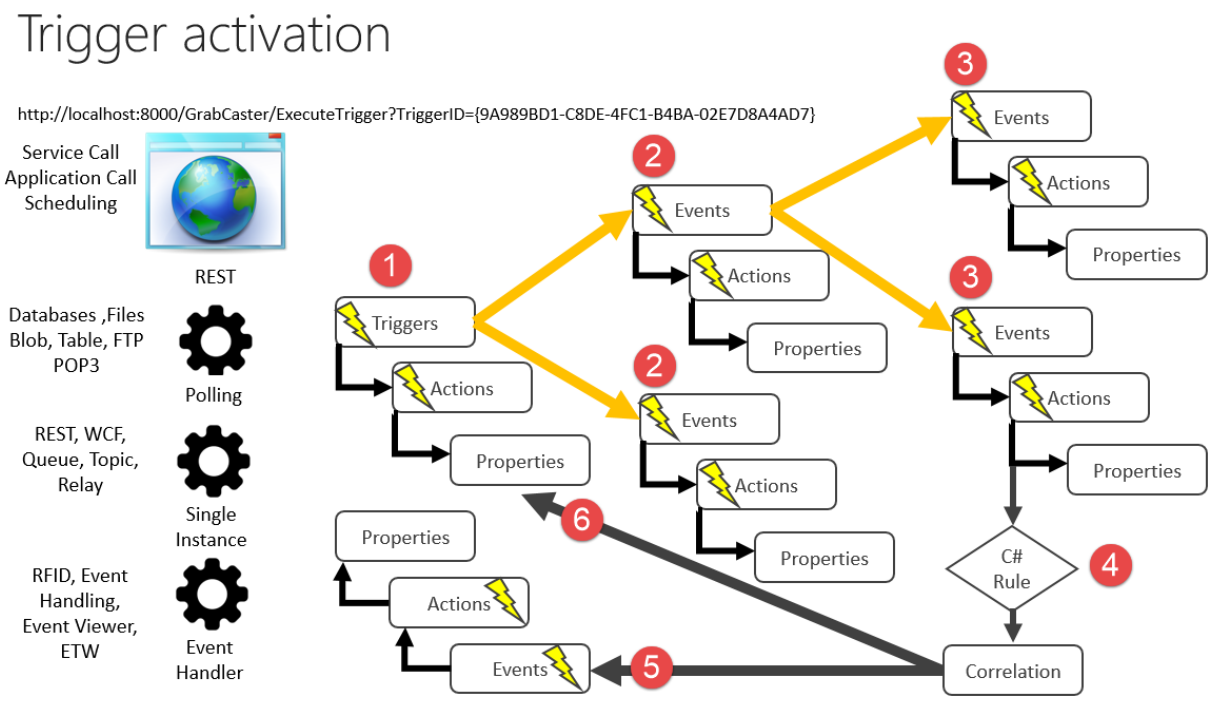
We have infinites options and combinations to subscribe and execute the GCPoints.

You can also use different MSPs types and organize different GCPoints per MSPI.

To activate a transmission, you need to configure a new trigger. A trigger is a .Net components that is specialized for particular integration stack, although some of them are also generic and are able to execute C# and PowerShell scripting.

A trigger can be mapped to one or multiple events and you can execute an event locally or remotely across the network. The data exchange is implicit between triggers and events and it is essentially based on a complex object mapping between properties and actions.

You don’t need to think how to send the data between trigger and events because Event Processing Engine luckily does this for us.



A trigger can be executed in four ways:

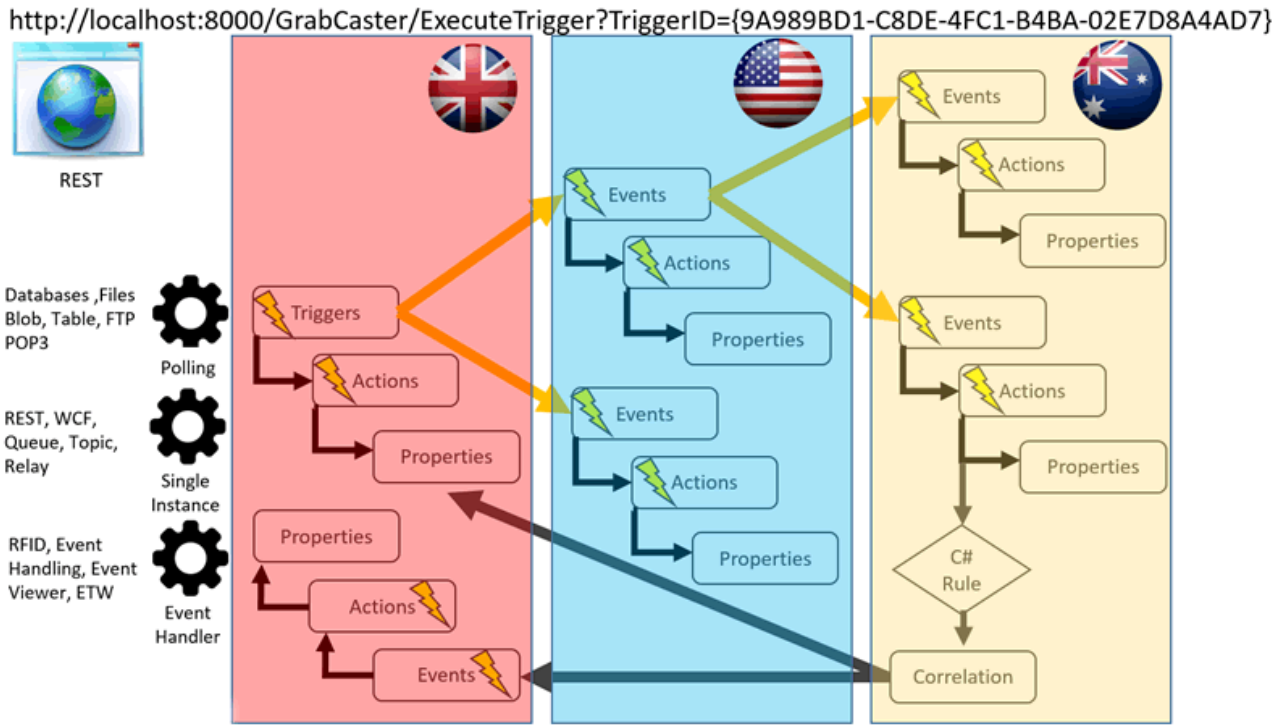
- REST Call
  - Any GrabCaster point exposes a REST API. You can use the REST API to execute a trigger or query the GCPoint.
    For more information refer to the GRABCASTER POINT SYNCHRONIZATION Chapter
- Polling
  - The trigger runs in polling mode. For example to execute a procedure in polling mode in a database, Microsoft BizTalk Sever uses similar approach.

    Important note:
    Current version supports a common polling time. Next version will support a single polling time per trigger.
- Single Instance
  - A single instance of the trigger will be executed
- Event Handler
  - The trigger is executed to manage an event handler interaction with the system. For example intercepting the operating system events and event viewer, ETW, Device Ports (RFID readers for example), File system Watcher patterns and so on.

A trigger (1) can execute an event or multiple events (2). And an event can also execute an event or multiple events (3) across the network.

A C# rule can be configured and executed locally or remotely (4) to drive the execution of other events (5) or manage a process correlation (6). GrabCaster uses Microsoft Roslyn to manage and execute the engine rule.

Any event can be executed locally or across the network. In the picture below the previous scenarios execution is split into three different countries.



5 CHANNELS AND GCPOINT INSTANCES

A GrabCaster point is a transmission and receiving point across a channel and using different providers the current version support Microsoft Azure and Redis Database. Any GCPoint create a preferential channel and we can send data across one channel, multiple channels or to all the channels and we can also specify to send event to multiple GC points.

Specifying a Channel ID = * means send the message to all the active channels in the same MSP.
Specifying a Point ID = * means send the message to all the active GC points in the same MSP.

We can combine different options as below:

Channel Id= * Point Id =*
Send the events to all the channels and all the GrabCaster points

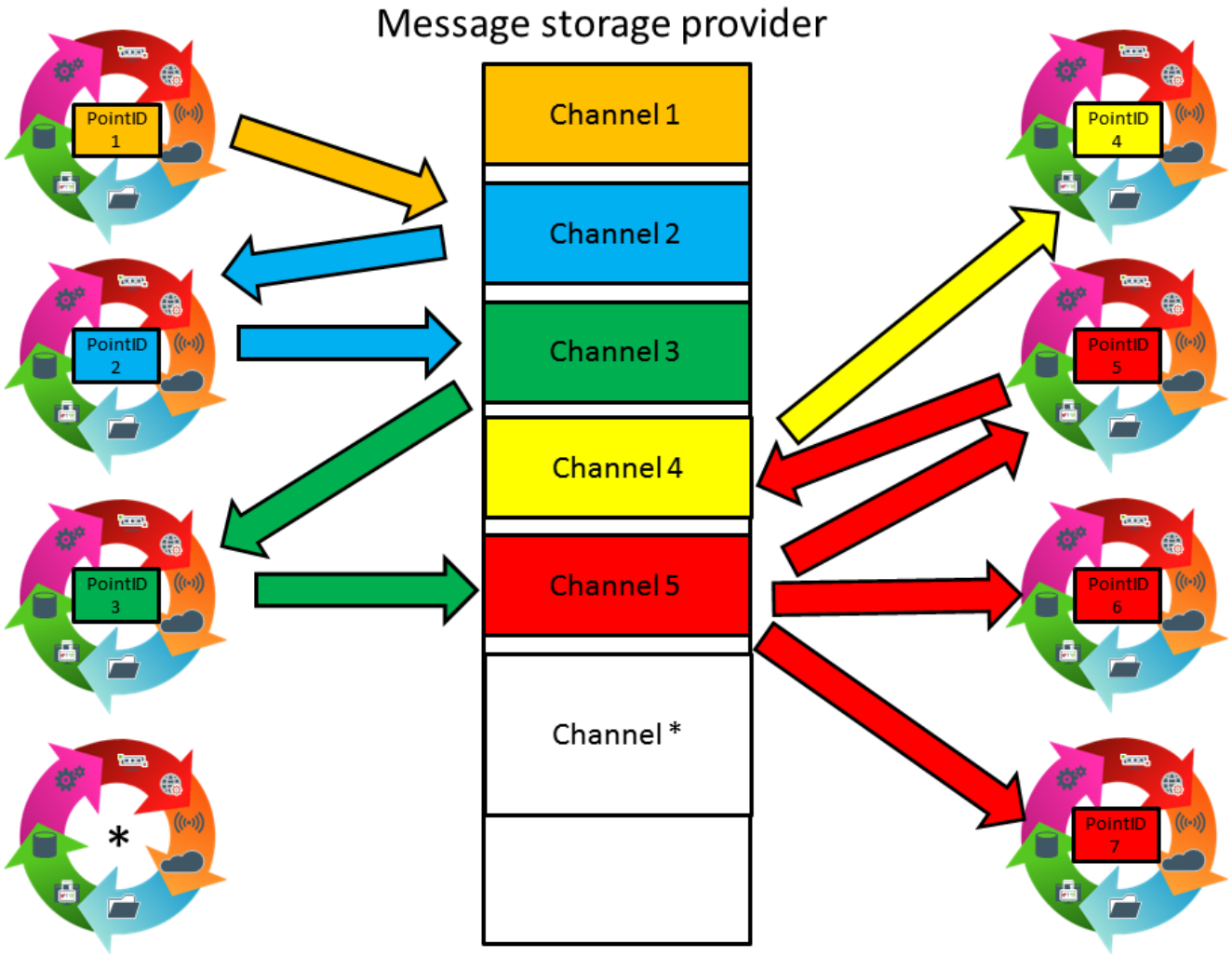
Channel Id = Channel 1 Point Id =*
Send to all the GrabCaster points in the Channel 1

Channel Id = * Point Id = Point 1
Send to the GrabCaster Point 1 looking in all the Channels

Channel Id = Channel 1 Point Id = Point 1
Send to the GrabCaster Point 1 in the Channel 1

The Channel Id or Point Id is not exclusive, any GrabCaster Point can change the channel Id or point Id at any time.

The picture below explains the relation between channels and GCPoint instances.



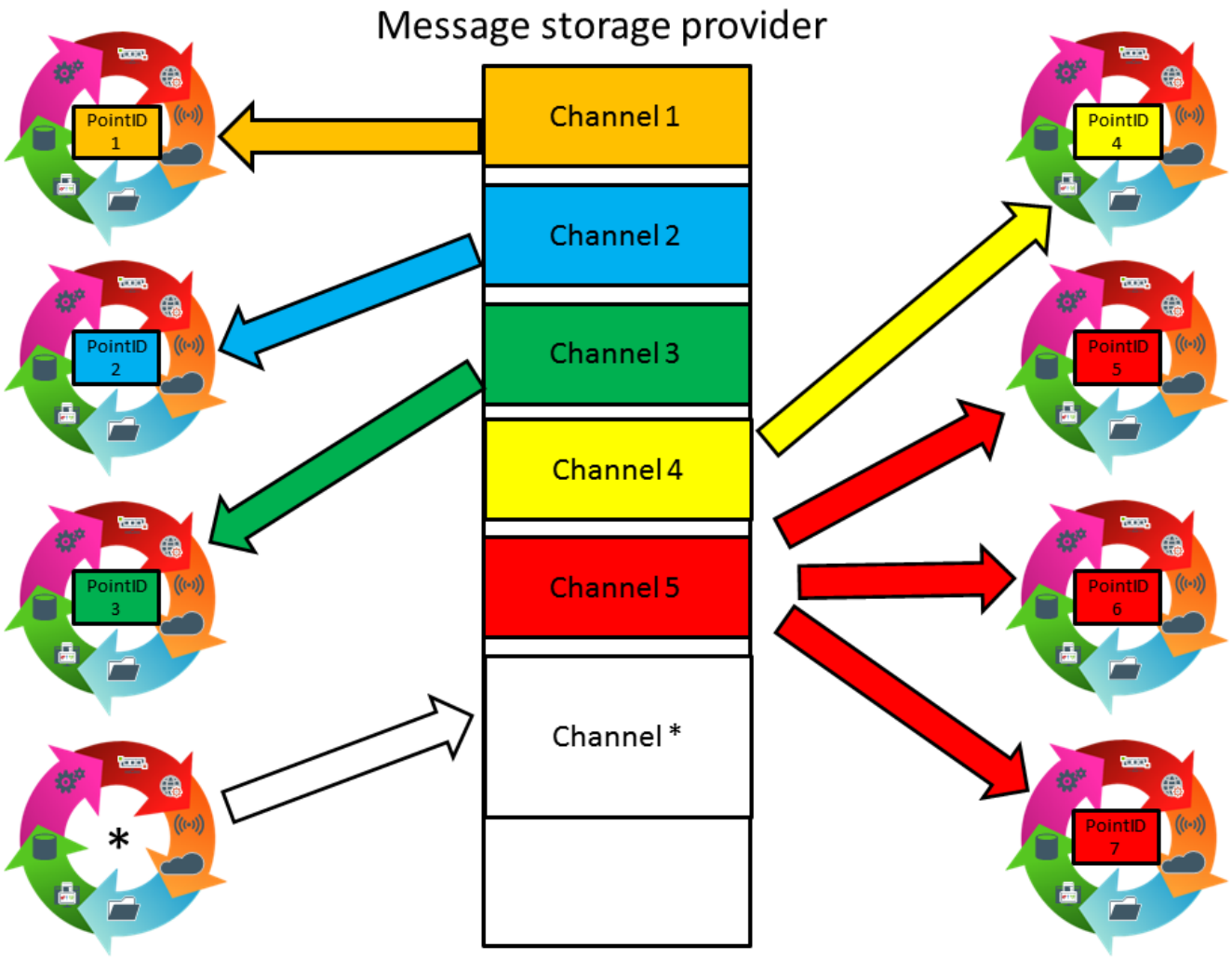
In order to send information across the channels we need to specify the ChannelID inside the trigger configuration file, each GCPoint subscribes to a channel, if no channels or point Id is specified the events will be mapped and executed directly in the GCPoint, this is very useful if you want to execute integration scenario in the same machine. For example collecting messages from the local event viewer machine and sending the data to a database.

The image above shows two different scenarios:

- Scenario 1
  - GCPoint subscribes to Channel 1 that executes a trigger to send the data to the Channel 2. GCPoint subscribed to Channel 2 will receive the data and execute the events. The event on GCPoint subscribed to the Channel 2 executes another event that sends the data to the Channel 3. GCPoint subscribed to Channel 3 receives the data and execute the events.
- Scenario 2
  - GCPoint subscribed to the Channel 3 execute a trigger and sends the data to Channel 5. All three GCPoint instances subscribed to Channel 5 will receive the data. One GCPoint subscribed to Channel 5 executes another event and it send the data to Channel 4. GCPoint subscribed to Channel 4 receives the data and execute the events.

As you can see a GCPoint is a generic broadcasting point. You can also change the subscription channel at any moment. If the channel doesn't exist, it will be created.

We can also have the option to send an event to all the GrabCaster points in the same MSPI using * (star) character for the Channel Id. This is very powerful for the case to execute something everywhere. For example a critical update across all servers or we need to send same data everywhere.

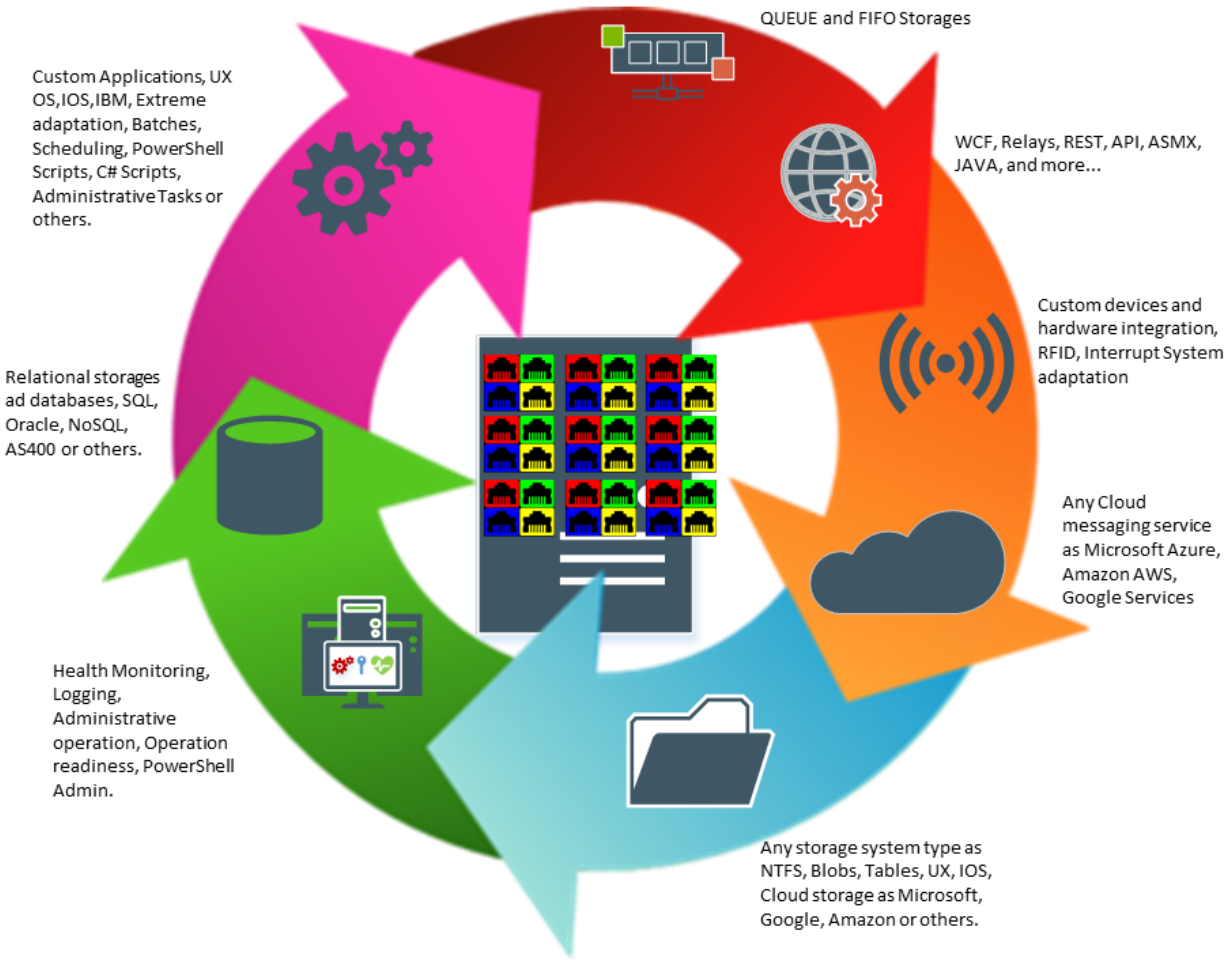


6 GRABCASTER INSTANCES

GrabCaster supports multiple instances running on the same machine.

> *Important note:*
> *Multiple GrabCaster instances in the same machine have to use different REST API port numbers.*

The current version supports two different running modes - as Console application or Windows NT Service. You can create GrabCaster clones in the same channel or different clones on different channels.

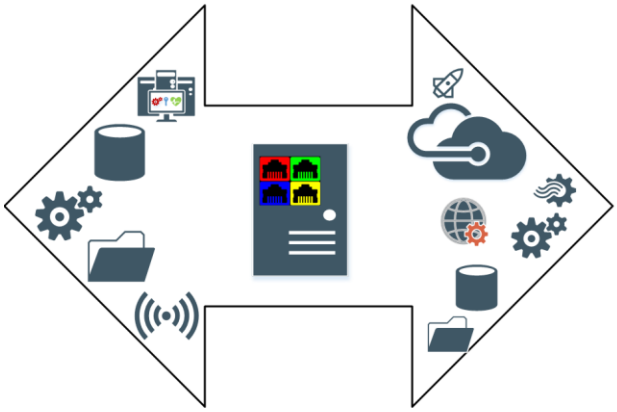


6.1 USE CASES

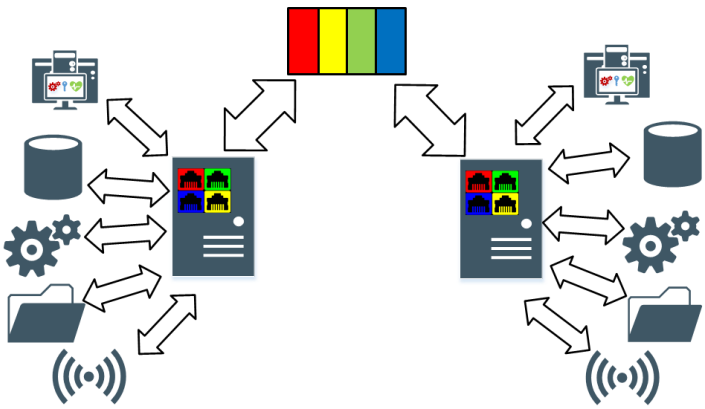
A typical use case is to use a GrabCaster point to integrate data between technology stacks in a plain and simple way, we can do that without specifying any channel or GrabCaster point Id in the trigger configuration file and the events will be executed internally by the GrabCaster point and directly.

For example you need to get a file and send the content to an Azure blob storage location. Or you need to collect data from the event viewer and execute a procedure for a particular error code or send this data to a database or to the cloud, or you just want to send data in an Azure Event Hub, essentially we use this approach when we don't need to execute events remotely.

The image below shows a classic scenario of integrating local technology stacks and executing events directly in the GrabCaster point.

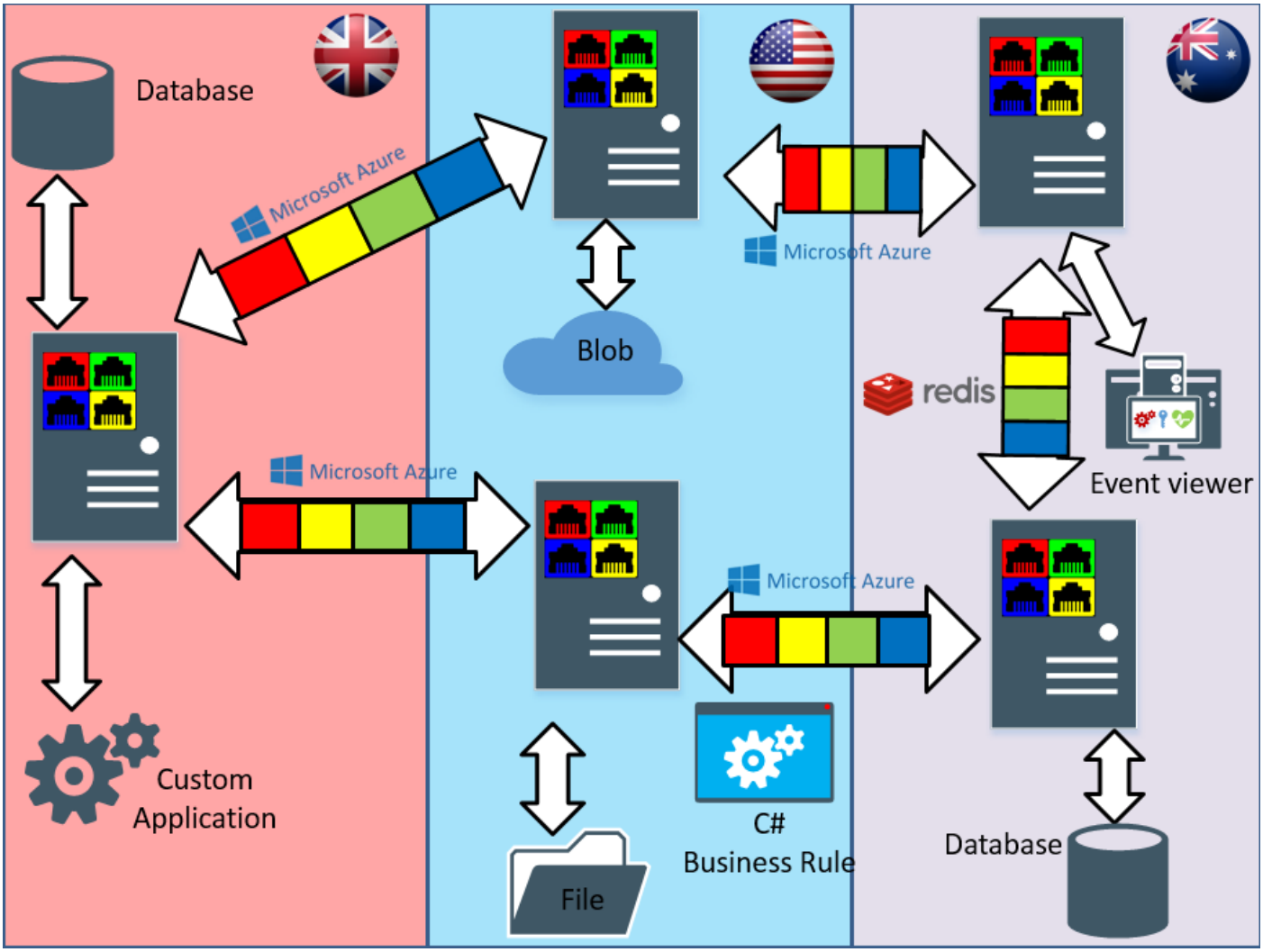


Using channels and points Id we are able to execute events remotely, the image below shows a classic scenario of integrating technology stacks through the local network and executing
events remotely.



We can also execute the events remotely and in different worldwide places and we can use different Message Storage Providers as Microsoft Azure and Redis Database together for different scopes and performances.

An example in the image below:



7 HOW TO GET STARTED

Important note:

Uninstall any old beta GrabCaster installations.
To uninstall the previous version enter in Windows Add Remove Program and uninstall the previous GrabCaster versions.

Install GrabCaster

The GrabCaster installation is a straightforward procedure, just three steps:

1. Download the last version of GrabCaster.msi from http://grabcaster.io
2. Execute the file GrabCaster.msi.
3. Select the directory installation and click install.

The current GrabCaster version is able to use different Message Storage Provider, the current version support Microsoft Azure Event Hubs and Redis Database, we can also use GrabCaster locally, in this case we don't need to configure anything but we won't be able to execute event remotely.

If you don't need to execute event remotely then GrabCaster is now ready to run, the Laboratory session contains many samples scenarios, you can refer to the GrabCaster documentation for more details.

7.1 Setting Up the Microsoft Azure environment

The current release version supports Microsoft Azure only. You need to create some simple Azure artefacts in order to run GrabCaster for the first time.

7.1.1 Create a Microsoft Azure account

First create a free trial Azure Account. You can refer to this link.

7.1.2 Create a Microsoft Azure Namespace

Next create a Microsoft Azure Namespace. Please refer to this link.

7.1.3 Create a Microsoft Azure Event Hubs

Now create a Microsoft Azure Event Hub. Please refer to this link.

Important Note:
In order to run GrabCaster you will need at a minimal 8 partitions for Event Hubs configuration.

7.1.4 Create a Microsoft Azure Storage Account

Now create a Microsoft Azure Storage Account. Please refer to this link.

7.2 Configure the broadcaster point

A broadcaster point would allow sending a transmission of messages from the broadcaster into the broadcast network that would deliver content to an audience. The broadcast network is represented by the message engine. For example Microsoft Azure and each broadcaster point receives messages through the channel.

Open File Explorer to the GrabCaster installation directory. Open the configuration file, *GrabCaster.cfg*, to configure the settings as instructed below.

7.2.1 AzureNameSpaceConnectionString

Set AzureNameSpaceConnectionString with the Microsoft Azure Namespace connection string.
GroupEventHubsName

Enter the name of Event Hubs you created in the previous section.

7.2.2 GroupEventHubsStorageAccountKey

Enter the shared key of the Microsoft Azure Storage Account. To get the Microsoft Azure Storage Account shared key refer to this link.

7.2.3 GroupEventHubsStorageAccountName

Enter the name of the Microsoft Azure Storage Account we created in the previous session. To get the Microsoft Azure Storage Account name, refer to this link.

7.2.4 GetLocalStorageConnectionString

Enter the local or shared file path to use as local persistent storage. The default value is [GrabCaster Installation Folder]\\PersistentStorage.

7.2.5 ChannelID

The ChannelID identifies the listening channel to use and to join. You can use any unique string value such as a 'Globally Unique Identifier' for example {D1AB5AD0-86F5-49A2-8A3C-ED2F60FE2E97} or any other value to identify unique channel. A good online resource is the Online GUID Generator.

> *Important note:*
> *The default value already contains a GUID {D1AB5AD0-86F5-49A2-8A3C-ED2F60FE2E97}, which you can keep and use.*

Generate a GUID and set the ChannelID.

When you install a new GrabCaster endpoint, you have the option to create new unique channel to use for sending and receiving messages. Or you can also configure different broadcaster points under the same channel.

7.2.6 ChannelName

Identify the unique name of the channel. You can use something useful to name the channel.

7.2.7 ChannelDescription

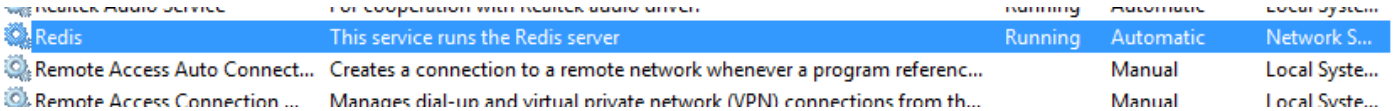
Set the channel description.

7.3 Setting Up the Redis Database Environment

The Redis installation is straightforward, download the Redis Database, you can use the Windows or Linux distribution and you can find the installation package here

http://redis.io/download

If you use the windows installation, after executing the msi file you will find the Redis NT service installed.



Run the Redis Windows NT Service and set the **RedisConnectionString** parameter in the GrabCaster configuration file, if you are using a local Redis service the connection string will be **localhost** otherwise will be **machinename:portnumber** or **IP:port number**.

You can change the Redis configuration in the file C:\Program Files\Redis\redis.windows-service.conf

For more information you can refer to the Redis Database documentation here

http://redis.io/documentation

8 File 2 File Integration Sample

Let's start with one of the most common and often used integration scenarios: receiving and sending a file to another partner. What we affectionately refer to as File2File integration.

The GrabCaster installation folder already contains different sample configuration files which you can find in the sub-folder *Bubbling*. This folder contains two subfolders, *Triggers* (to activate and configure new triggers) and Events (to activate and configure new events).

The GrabCaster Laboratory site session contains many samples to use.

8.1 CONFIGURING A NEW TRIGGER

Navigate to the Trigger configuration folder [*GrabCaster Installation Directory*]*Bubbling* directory. Inside you will find several files with [.off] extension, which means that these triggers are disabled. In order to enable a trigger or event, you need to change the file extension from [*.off*] to [*.trg*] .

Open for example the file “Demo File In_Local 2 Out_Local.trg” with your chosen JSON editor.

The trigger configuration file should appear something like below:

```
{
  "Trigger": {
1 "IdComponent": "{3C62B951-C353-4899-8670-C6687B6EAEFC}",
   "Name": "FileTrigger To Out",
   "Description": "Get file from disc",
   "TriggerProperties": [{
       2 "Name": "RegexFilePattern",
         "Value": ".(txt|a)"
       },
       {
       3 "Name": "DoneExtensionName",
         "Value": "done"
       },
       {
       4 "Name": "PollingTime",
         "Value": "5000"
       },
       {
         "Name": "InputDirectory",
       5 "Value": "C:\\Program Files (x86)\\GrabCaster\\Demo\\File2File\\In_Local"
       }]
   },
  "Events": [
6 {
7 "IdConfiguration": "{F4CF1B12-4EC4-43FA-8842-575A5E4CDE5A}",
  "IdComponent": "{D438C746-5E75-4D59-B595-8300138FB1EA}",
   "Name": "File2File Demo Event ",
   "Description": "Write file to the GrabCaster Demo OUT folder"
   }
  ]
}
```

1) Trigger IdComponent component to invocate
2) File extension to look (Internal functionality)
3) File extension to use to rename the file
4) Polling time
5) Input directory
6) IdConfiguration group to execute
7) IdComponent Event to execute

8.2 CONFIGURE A NEW EVENT

Navigate to the Events configuration folder under the [*GrabCaster Installation Directory*]\Bubbling directory. You will find several files with [.off] extension, which means that these events are disabled. In order to enable an event you need to change the file extension from [*.off*] to [*.evn*] as well as setting the properties in the configuration file.

Open the file “Demo File Out.evn” with your chosen JSON editor and change the settings you need as described below. All event configuration settings are explained in detail in the chapter Event Configuration in the GrabCaster Laboratory site session.

The event configuration file should appear something like below:

```
{
    "Event": {
      1 "IdConfiguration": "{F4CF1B12-4EC4-43FA-8842-575A5E4CDE5A}",
      2 "IdComponent": "{D438C746-5E75-4D59-B595-8300138FB1EA}",
        "Name": "Write file in OUT Local",
        "Description": "Write file in OUT Local",
        "EventProperties": [{
            "Name": "OutputDirectory",
          3 "Value": "C:\\Program Files (x86)\\GrabCaster\\Demo\\File2File\\Out\\FileTest.txt"
        }]
    }
}
```

1) IdConfiguration group to execute
2) IdComponent Event to execute
3) Output file to write.

8.3 START GRABCASTER

The current version supports two different running options:

- Console application - this is a good option if you want to execute GrabCaster in a fast way without particular system security requirements.
- Window NT Service - this option will install GrabCaster as Windows NT Service. The Windows NT mode will be explained in the next chapters.

Important note:
You can distribute and execute multiple GrabCaster instances in the same machine.

8.3.1 Console mode execution

Double-click on the GrabCaster desktop icon to execute GrabCaster console using the current user logon account.

Important note:
To execute GrabCaster in Administrator mode right-click on the GrabCaster icon and select "Run as administrator".

Press the "M" key to execute GrabCaster.

8.3.2 Windows NT mode execution

Double-click on the GrabCaster desktop icon to execute GrabCaster console using the current user logon account.

Important note:
To execute GrabCaster in Administrator mode right-click on the GrabCaster icon and select "Run as administrator".

8.3.2.1 Install a new GrabCaster NT Service point

Double-click on the GrabCaster desktop icon to execute GrabCaster console using the current user logon account.

Important note:
To execute GrabCaster in Administrator mode right-click on the GrabCaster icon and select "Run as administrator".

Press the "I" key to install a new GrabCaster Windows NT instance.

Specify the Windows NT service name and press enter.

8.3.2.2 Install a new GrabCaster NT Service point

Double-click on the GrabCaster desktop icon to execute GrabCaster console using the current user logon account.

> Important note:
> *To execute GrabCaster in Administrator mode right-click on the GrabCaster icon and select "Run as administrator".*

Press the "U" key to uninstall a GrabCaster Windows NT instance.
Specify the Windows NT service name and press enter.

8.4 CLONE A NEW GRABCASTER POINT

Double-click on the GrabCaster desktop icon to execute GrabCaster console using the current user logon account.

> Important note:
> *To execute GrabCaster in Administrator mode right-click on the GrabCaster icon and select "Run as administrator".*

We can decide to clone a new GrabCaster point to install and execute multiple instances with different configuration in the same machine.

Press the "C" key to create a new GrabCaster clone.
Specify the GC service name and press enter

8.5 SEND FILE LOCALLY

To execute this sample refer to the **Sample Demo File In_Local 2 Out_Local.trg** in the GrabCaster Laboratory site session.

8.6 SEND FILE REMOTELY

To execute this sample refer to the **Sample Demo File In_LocalRemote 2 Out.trg** in the GrabCaster Laboratory site session.

9 GRABCASTER REST POINT

GrabCaster implements a REST mechanism to provide the description of Triggers, Events and the configuration information. In order to use it, open Internet Explorer and navigate on http://localhost:8000 this is the default value we can change this value in the GrabCaster configuration value in the **WebApiEndPoint** key.

> *Important note:*
> *The port number needs to be different for each GC point in the same machine.*

The result provides all the information we need to interact with the broadcaster point and to configure triggers and events.

For more information refer to the GRABCASTER POINT SYNCHRONIZATION Chapter.

10 CONFIGURE AN EXISTING TRIGGERS AND EVENTS TEMPLATE

The current GrabCaster release provides sample triggers and events templates ready for use:

10.1 TRIGGERS TEMPLATES

- AzureBlobTrigger.dll

  Receives data from an Azure Blob storage.

- AzureQueueTrigger.dll

  Receives data from an Azure Queue.

- AzureTopicTrigger.dll

  Receives data from an Azure Topic using dynamic subscriptions.

- BulkSQLServerTrigger.dll

  Executes a SQL bulk SELECT statement. Useful when needing to extract and send large amount of records across the network.

- CSharpTrigger.dll

  Executes C# script.

- ETW.dll

  Captures events form Windows ETW.

- EventHubsTrigger.dll

  Receives data from Azure Event Hubs.

- EventViewerTrigger.dll

  Captures event from Windows event viewer.

- FileTrigger.dll

  Get file from a specific directory and extension.

- NOPTrigger.dll

  Simulates a NULL data trigger (Test and performance metric purpose only)

- PowershellTrigger.dll

  Executes PowerShell as a trigger. Very useful to implement those who don't have C# development skills.

- RfidTrigger.dll

  Receive data from a Phidget RFID device.

  IMPORTANT NOTE: The Phidget RFID device driver need to be installed before to activate the trigger.

- SQLServerTrigger.dll

  Execute a SQL statement in polling.

10.2 EVENTS TEMPLATES

- AzureBlobEvent.dll

Sends data to an Azure Blob storage.

- AzureQueueEvent.dll

  Sends data to an Azure Queue.

- AzureTopicEvent.dll

  Sends data to an Azure Topic from dynamic subscriptions.

- BulkSQLServerEvent.dll

  Executes a SQL bulk insert. Useful when receiving large number of records across the network, which has to be inserted into SQL Server.

- CSharpEvent.dll

  Executes C# script.

- DialogBoxEvent.dll

  Executes a Yes/No dialog box. Useful to use with a C# Rule.

- EventHubEvent.dll

  Sends data to Azure Event Hubs.

- FileEvent.dll

  Writes a file in a specific path with a specific name.

- MessageBoxEvent.dll

  Shows a MessageBox.

- NOPEvent.dll

  Simulates a null data event (Test and performance metric purpose only).

- PowershellEvent.dll

  Executes a PowerShell as event. Very useful for people who don't have C# development skills.

- TwilioEvent.dll
  - Sends a mobile text message using the Twilio provider.

10.3 STEPS TO CONFIGURE A NEW TRIGGER

To configure a new trigger you can query the GCPoint. Instructions on how to query a GCPoint, refer to the chapter *QUERY THE GRABCASTER POINT.*

- Select the trigger type to configure and get the JSON content from the JSON node

  For example for the file trigger look for file trigger and JSON node. For the event viewer trigger look for event viewer and JSON node.

- Open Explorer to *[GrabCaster Installation Folder]\Bubbling\Triggers*
- Create a new trigger configuration file

Good practice is to use a name that describes what the trigger does. For example for a file trigger that gets a file and then executes a write file event, the name could be GetFileFromSystemX2.off.

Important note
Use the extension .off to keep the new trigger off. You will activate the trigger later.

- Set the trigger properties and channels ID

  You can use a single or multiple channels. You can also specify the characters * (star) if you want to execute the event in all the GCPoint under our same Message Storage Provider.

10.4 STEPS TO CONFIGURE A NEW EVENT

To configure a new event you can query the GrabCaster point. Instructions on how to query a GCPoint refer to the chapter *QUERY THE GRABCASTER POINT.*

- Select the event type to configure and get the JSON content from the JSON node

  For example for the write file event look for **writefile** in the JSON nodes.

- Open Explorer to *[GrabCaster Installation Folder]\Bubbling\Events*
- Create a new event configuration file

  Good practice is to use a name that describes what the event does. For example for a write file event which writes a file could be WriteFileInSystemY.off.

  Important note
  Use a file name convention that you prefer to best support operational readiness. The configuration filename is completely irrelevant for the EPI.

- We can now rename the event file extension to .evn and enable the trigger.

11 DEVELOPING NEW TRIGGERS AND EVENTS

Creating a new trigger or event is very straightforward. GrabCaster already contains different sample templates that you can use and extend. For a full list, refer to the chapter *TRIGGERS TEMPLATES and EVENTS TEMPLATES*.

You can use several methods to create a new trigger or event, using Microsoft Visual Studio, PowerShell or C# script, depending on your requirement and situation.

11.1 USING MICROSOFT VISUAL STUDIO

Many sample of events and triggers are already included, we can start from an example to create a new one. Let's look at the file trigger on how simple this approach can be, the complete source code is under the FileTrigger repository in GitHub.

11.1.1 Develop a Trigger component

Create a public class named FileTrigger and inherit the interface `ITriggerType.`

```
public class FileTrigger : ITriggerType
```

Enrich the class with the TriggerContract attribute so that the EPI will be able to consider your component as GrabCaster Trigger.

11.1.1.1 Trigger Contract

```
/// <summary>
/// The file trigger.
/// </summary>
[TriggerContract("{3C62B951-C353-4899-8670-C6687B6EAEFC}", "FileTrigger", "Get the content from file in a specific directory or shared forlder.", false, true, false)]
0 references | 0 changes | 0 authors, 0 changes
public class FileTrigger : ITriggerType
```

Where:

- ID
  - Specify a unique ID to identify the trigger.
- Name
  - Specify the class name.
- Description
  - Specify a description
- PollingRequired
  - True if the trigger needs to be executed in polling.
- Shared
  - True if the trigger needs to be shared and synchronized with all the GCPoints in the same MPS.
- NOP
  - Test and performance metric purpose only.

Implement the interface methods and properties as well as adding the applicable attributes as shown below.

11.1.1.2 Property contract

```
/// <summary>
/// Gets or sets the regex file pattern.
/// </summary>
[TriggerPropertyContract("RegexFilePattern", "File pattern, could be a reular expression")]
1 reference | 0 changes | 0 authors, 0 changes
public string RegexFilePattern { get; set; }

/// <summary>
/// Gets or sets the polling time.
/// </summary>
[TriggerPropertyContract("PollingTime", "Polling time.")]
1 reference | 0 changes | 0 authors, 0 changes
public int PollingTime { get; set; }

/// <summary>
/// Gets or sets the done extension name.
/// </summary>
[TriggerPropertyContract("DoneExtensionName", "Rename extension file received.")]
2 references | 0 changes | 0 authors, 0 changes
public string DoneExtensionName { get; set; }

/// <summary>
/// Gets or sets the input directory.
/// </summary>
[TriggerPropertyContract("InputDirectory", "Input Directory location")]
2 references | 0 changes | 0 authors, 0 changes
public string InputDirectory { get; set; }
```

Where:

- Name
  - Specify the property name.
- Description
  - Specify a description.

11.1.1.3 Action Contract

```
/// <summary>
/// The execute.
/// </summary>
/// <param name="setEventActionTrigger">
/// The set event action trigger.
/// </param>
/// <param name="context">
/// The context.
/// </param>
[TriggerActionContract("{58EEAFEF-CF6A-44C3-9BB9-81EFD680CA36}", "Main action", "Main action description")]
13 references | 0 changes | 0 authors, 0 changes
public void Execute(SetEventActionTrigger setEventActionTrigger, EventActionContext context)
```

Where:

- ID
  - Any unique ID.
- Name
  - Specify the action name.
    The **Execute** event is an internal reserved event and has to be implemented because inherited by the interface.
    The Event Processing Engine performs an object mapping to map the object properties and the result of the methods with the same name and it will execute the main event main “Execute method”, in case of custom methods the attribute Name has to be same of the real method name.
- Description
  - Specify a description.

11.1.1.4 Trigger context

11.1.2 Implement the Execute

Since the EPI will perform the Execute method, it should contain the main logic of the component. For example for the file trigger the code could be:

```
/// </summary>
/// <param name="setEventActionTrigger">
/// The set event action trigger.
/// </param>
/// <param name="context">
/// The context.
/// </param>
[TriggerActionContract("{58EEAFEF-CF6A-44C3-9BB9-81EFD680CA36}", "Main action", "Main action description")]
13 references | 0 changes | 0 authors, 0 changes
public void Execute(SetEventActionTrigger setEventActionTrigger, EventActionContext context)
{
    try
    {
        while (true)
        {
            var reg = new Regex(this.RegexFilePattern);
            if (Directory.GetFiles(this.InputDirectory, "*.txt").Where(path => reg.IsMatch(path)).ToList().Any())
            {
                var file =
                    Directory.GetFiles(this.InputDirectory, "*.txt")
                        .Where(path => reg.IsMatch(path))
                        .ToList()
                        .First();
                var data = File.ReadAllBytes(file);
                PersistentProvider.PersistMessage(context);
                File.Delete(Path.ChangeExtension(file, this.DoneExtensionName));
                File.Move(file, Path.ChangeExtension(file, this.DoneExtensionName));
                this.DataContext = data;
                setEventActionTrigger(this, context);
            }

            Thread.Sleep(this.PollingTime);
        }
    }
    catch (Exception)
    {
        setEventActionTrigger(this, null);
    }
}
```

1) Where the code looks for a file in a directory and read all the byte.
2) Deletes the file.
3) Puts the content in the Trigger Context
4) Passes the context to the EPI

Important note:
This is a very simple implementation. You can improve a lot on this sample with transactional codes and reliability patterns. The current code is for sample purposes only! GrabCaster community contains many other sample templates and implementations.

11.1.2.1 Trigger context

In particular cases we need to manage the event context in different methods inside the class in this case we can create the EventActionContext to manage the context data and the SetEventActionTrigger to manage the delegation as below, refer to the **RfidTrigger** or **EventViewerTrigger** sample for more details.

```
/// <summary>
/// Gets or sets the context.
/// </summary>
24 references | 0 changes | 0 authors, 0 changes
public EventActionContext Context { get; set; }

/// <summary>
/// Gets or sets the set event action trigger.
/// </summary>
24 references | 0 changes | 0 authors, 0 changes
public SetEventActionTrigger SetEventActionTrigger { get; set; }
```

11.1.3 Deploy the trigger

Compile the trigger and copy the DLL component in the [GrabCaster Installation Folder]\Root_GrabCaster\Triggers.

11.1.4 Develop an Event component

Creating an event is similar to creating a trigger, just a bit simpler.

Create a public class named FileEvent and inherit the Event interface `IEventType.` Decorate the class with the EventContract attribute as below:

```
/// <summary>
/// The file event.
/// </summary>
[EventContract("{D438C746-5E75-4D59-B595-8300138FB1EA}", "Write File", "Write the content in a file in a specific folder.", true)]
0 references | 0 changes | 0 authors, 0 changes
public class FileEvent : IEventType
```

11.1.4.1 Property Contract

```
/// <summary>
/// Gets or sets the output directory.
/// </summary>
[EventPropertyContract("OutputDirectory", "When the file has to be created")]
1 reference | 0 changes | 0 authors, 0 changes
public string OutputDirectory { get; set; }
```

The context is the event context package the DataContext is the data property used by trigger and event, in case of file trigger the DataContext contains the file content in bytes.

11.1.4.2 Action Contract

```
/// <summary>
/// The execute.
/// </summary>
/// <param name="setEventActionEvent">
/// The set event action event.
/// </param>
/// <param name="context">
/// The context.
/// </param>
[EventActionContract("{1FBD0C6E-1A49-4BEF-8876-33A21B23C933}", "Main action", "Main action description")]
12 references | 0 changes | 0 authors, 0 changes
public void Execute(SetEventActionEvent setEventActionEvent, EventActionContext context)
```

11.1.5 Implement the Execute

1) An event receive the DataContext from the trigger, for example in case of file trigger it contains the file content in byte.

```
[EventActionContract("{1FBD0C6E-1A49-4BEF-8876-33A21B23C933}", "Main action", "Main action description")]
12 references | 0 changes | 0 authors, 0 changes
public void Execute(SetEventActionEvent setEventActionEvent, EventActionContext context)
{
    try
    {
        Debug.WriteLine("In FileEvent Event.");
1       File.WriteAllBytes(this.OutputDirectory + Guid.NewGuid() + ".txt", this.DataContext);
2       this.DataContext = Serialization.ObjectToByteArray(true);
        setEventActionEvent(this, context);
    }
    catch (Exception ex)
    {
3       Debug.WriteLine("FileEvent error > " + ex.Message);
        setEventActionEvent(this, null);
    }
}
```

2) Set the DataContext for the EPE and execute the propagation
   The EPE will expect a not null `DataContext to propagate the execution to do that we can simple set to true the DataContext value.`
3) `We can stop the event propagation setting the context to null value.`

Important note:
we can pass different content to the DataContext to provide continuous integration and call other events.

11.1.6 Deploy the event

Compile the event and copy the DLL component to the *[GrabCaster Installation Folder] \Root_GrabCaster\Events* directory.

11.1.7 Using PowerShell

You can also create trigger and event using PowerShell. In order to do that, you need to write a PowerShell script that uses the PowerShellTrigger and PowerShellEvent components.

You can use a PowerShell script directly inside the trigger configuration file for very simple scripting. Or you can create a PowerShell script file for more complex integration approach. Below is a sample PowerShell script file to integrate a file from a folder.

GetFile.ps1

```
1  Param($DataContext)
2
3          $Dir = get-childitem "C:\Users\Nino\Documents\GrabCaster\Demo\GrabCaster1\Demo\File2File\PowershellIn" -recurse
4          # $Dir |get-member
5          $files = $Dir | where {$_.extension -eq ".txt"}
6          for ($i=0; $i -lt $files.Count; $i++) {
7              $outfile = $files[$i].FullName + ".done"
8              $DataContext = Get-Content $files[$i].FullName
9              rename-item -path $files[$i].FullName -newname $outfile
10
11         }
12
```

1) Declare a general *$DataContext* variable in your PowerShell script that will hold data content and propagate the events.

Inside the PowerShellTrigger configuration, you need to insert your PowerShell script as below.

```
{
  "Trigger": {
    "IdComponent": "{18BB5E65-23A2-4743-8773-32F039AA3D16}",
    "Name": "Powershell Trigger File",
    "Description": "Powershell Trigger File",
    "TriggerProperties": [
    {
        "Name": "Script",
        "Value": ""
    },
    {
    1   "Name": "ScriptFile",
        "Value": "C:\\Program Files (x86)\\GrabCaster\\Demo\\GetFile.ps1"
    },
    {
        "Name": "MessageProperties",
        "Value": ""
    }
    ]
    },
  "Events": [
    {
    "IdConfiguration": "{F4CF1B12-4EC4-43FA-8842-575A5E4CDE5A}",
    "IdComponent": "{D438C746-5E75-4D59-B595-8300138FB1EA}",
    "Name": "File2File Demo Event ",
    "Description": "Write file to the GrabCaster Demo OUT local folder"
    }
  ]
}
```

Where:

- Script
  - If you want to use the script directly
- ScriptFile
  - If you want to use a PowerShell script file.

Another simple useful sample for using PowerShell scripting is executing a batch program from a PowerShelEvent

ExecAppBackup.ps1

```
Param($DataContext)
Invoke-Item "C:\Users\Nino\Documents\GrabCaster\Demo\GrabCaster1\Demo\BackupApp.exe"
```

Or pick up a particular log event form the Windows Event Viewer and send a mobile text message as in the Demo PowerShell EventViewer 2 TextMessage.trg sample, see below:

```
{
  "Trigger": {
        "IdComponent": "{18BB5E65-23A2-4743-8773-32F039AA3D16}",
        "Name": "Powershell Trigger",
        "Description": "Powershell Trigger",
    "TriggerProperties": [
    {
        "Name": "Script",
        "Value": "Param($DataContext)
                  $DataContext = Get-Eventlog -log application -after ((get-date).AddSeconds(-2)) -EntryType Error | Where-Object {($_.Source -eq 'DEMOTEXTMESSAGE')}"
    },
    {
        "Name": "ScriptFile",
        "Value": ""
    },
    {
        "Name": "MessageProperties",
        "Value": "MessageNumber"
    }
    ]
    },
  "Events": [
    {
        "IdConfiguration": "{F4CF1B12-4EC4-43FA-8842-575A5E4CDE5A}",
        "IdComponent": "{A5765B22-4003-4463-AB93-EEB5C0C477FE}",
        "Name": "Send Text",
        "Description": "Send Text because Powershell Demo"
    }
  ]
}
```

Where:

1) Execute the PowerShell script to look in the event viewer.
2) Execute the Twilio event to send the text message.

11.2 USING C# SCRIPT

Using C# script is the same approach used for PowerShell script.

The C# script below will get files from a directory with a specified extension, you can see that is similar to the FileTrigger trigger dll.

```
using Framework.Contracts;
using Framework.Contracts.Globals;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

try
{
    byte[] data = null;  1
    while (true)
    {
        var reg = new Regex(this.RegexFilePattern);
        if (Directory.GetFiles(this.InputDirectory, "*.txt").Where(path => reg.IsMatch(path)).ToList().Any())
        {
            var file =
                Directory.GetFiles(this.InputDirectory, "*.txt")
                    .Where(path => reg.IsMatch(path))
                    .ToList()
                    .First();
        2 data = File.ReadAllBytes(file);
            PersistentProvider.PersistMessage(context);
            File.Delete(Path.ChangeExtension(file, this.DoneExtensionName));
            File.Move(file, Path.ChangeExtension(file, this.DoneExtensionName));
            this.DataContext = data;
            setEventActionTrigger(this, context);
        }

        Thread.Sleep(this.PollingTime);
    }
}
catch (Exception)
{
    setEventActionTrigger(this, null);
}
```

In the trigger file configuration, you need to specify the C# script file location

```
        {
                "Name": "Script",
                "Value": ""
        },
        {
                "Name": "ScriptFile",
                "Value": "C:\\MyTriggersConfigurationDirectory\\CSharpDemoScript.cs"
        }
```

Where:

- Script
  - If you want to use the script directly.
- ScriptFile
  - If you want to use a C# script file.

12 CLUSTER GRABCASTER

GrabCaster can run as Windows NT Service and we can cluster the service, if we decide to use a cluster configuration we need to specify a common operational directory, to do that we need:

Set the Cluster key in the GrabCaster configuration file to true and set the common directory to use in the ClusterBaseFolder key, below a sample:

- “Cluster”: true
  - True if GrabCater is clustered
- "ClusterBaseFolder": "C:\\GrabCasterQuorum"
  - If GrabCaster is clustered this is the directory containing the Root_GrabCaster common directory.

Create the GrabCasterQuorum directory and copy the folder Root_GrabCaster from the GrabCaster installation directory into the GrabCasterQuorum folder.

13 GRABCASTER POINT SYNCHRONIZATION

The Synchronization Provider synchronizes all the GCPoints in the same Message Storage Provider. It synchronizes the shared components and manages the remote triggers and events configuration. The configuration of all the GCPoints is stored in the C:\Program Files (x86)\GrabCaster\Root_GrabCaster\Endpoints.

Inside the EndPoints directory you can find all the configurations of the other GCPoints. You can change a remote GCPoint trigger or event configuration.

To synchronize component and configuration files with the other points we can use the local REST endpoint exposed by the local GrabCaster point, below the REST calls available in the current version with sample calls.

http://localhost:8000/GrabCaster/SyncSendBubblingConfiguration?ChannelID=*&PointID=*

Send the local bubbling configuration to all the other GC point in all the channels.
We can drive channel and PointID using specific values, where “*” means all.

http://localhost:8000/GrabCaster/SyncSendFileBubblingConfiguration?ChannelID={047B6D1E-A991-4CB1-ACAB-E83C3BDC0097}&PointID={B0A46E60-443C-4E8A-A6ED-7F2CB34CF9E5}&FileName=FileName.trg or .evn

Send the specific bubbling configuration to the specific GC point.
We can drive channel and PointID using specific values, where “*” means all.

http://localhost:8000/GrabCaster/SyncSendRequestBubblingConfiguration?ChannelID=*&PointID=*

Send the local bubbling configuration to all the other GC points in all the channels.
We can drive channel and PointID using specific values, where “*” means all.

http://localhost:8000/GrabCaster/SyncSendComponent?ChannelID=*&PointID=*&IDComponent={3C62B951-C353-4899-8670-C6687B6EAEFC}

Send the specific component dll (trigger or event) to all the other GC points in all the channels.
We can drive channel and PointID using specific values, where “*” means all.

http://localhost:8000/GrabCaster/SyncSendRequestComponent? ChannelID={047B6D1E-A991-4CB1-ACAB-E83C3BDC0097}&PointID={B0A46E60-443C-4E8A-A6ED-7F2CB34CF9E5}&IDComponent={3C62B951-C353-4899-8670-C6687B6EAEFC}

Request the specific component dll (trigger or event) to the specific GC point in a specific channel.
We can drive channel and PointID using specific values, where “*” means all.

http://localhost:8000/GrabCaster/SyncSendRequestConfiguration?ChannelID=*&PointID=*

Request the local configuration file to all the other GC points in all the channels.
We can drive channel and PointID using specific values, where "*" means all.

http://localhost:8000/GrabCaster/RefreshBubblingSetting

Refresh the internal configuration, Triggers and events.

http://localhost:8000/GrabCaster/ExecuteTrigger?TriggerID={3C62B951-C353-4899-8670-C6687B6EAEFC}

Execute a specific trigger.

http://localhost:8000/GrabCaster/Configuration

Show the internal Triggers and Events configuration.

14 OPERATION READINESS

In this chapter the most relevant operational readiness areas.

14.1 GRABCASTER CONFIGURATION FILE

The GrabCaster configuration file is in the program file folder C:\Program Files (x86)\GrabCaster.

- "AzureNameSpaceConnectionString": "Endpoint=sb://[your namespace].servicebus.windows.net;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=[your secret]"
  - The Azure connection string used to connect to the Azure Cloud.
- GroupEventHubsName": "[azure event hubs name]"
  - The EventHubs name used by the MSP.
- "GroupEventHubsStorageAccountKey": "[your secret]"
  - The Azure storage account key used by the MSP.
- "GroupEventHubsStorageAccountName": "[azure storage account name]"
  - The Azure storage account name used by the MSP.
- "EventHubsStartingDateTimeReceiving": "0"
  - [**Advanced setting**] You can manage the persistence mechanism used by Azure EventHubs below the enumerations.
    - 0, start from the last checkpoint.
    - A datetime value, we can manage how long time back we want to start receiving from the Event Hubs refer to the EventHubsCheckPointPattern for more detail.
      For more information refer to the EventHubs site here.
- "EventHubsEpoch": 0
  - [**Advanced setting**] EventHubs receiver epoch feature provides users ability to ensure that there is only one receiver on a consumer group at any point in time, with the following rules.
    For more information refer to the EventHubs site here.
- ServiceBusConnectivityMode": "Http"
  - If Http the Microsoft Azure DSPI will use the Web socket pattern communication.
- "EventHubsCheckPointPattern": "DTUTCNOW"
  - If EventHubsStartingDateTimeReceiving = 1 we can use the value below:
    - `DTNOW receive the last checkpoint from datetime now`

  - DT receive the last checkpoint from EventHubsStartingDateTimeReceiving date time
  - DTEPOCH receive the last checkpoint from EventHubsEpoch specified
  - DTUTCNOW receive the last checkpoint from datetime UTC now
  - DTUTCNOWEPOCH receive the last checkpoint from EventHubsEpoch UTC specified
  - DTNOWEPOCH receive the last checkpoint from EventHubsEpoch NOW specified

  [**Advanced setting**] Define the pattern used by the EventHubs chekpoint, DTUTCNOW is the UTC date time we can also specify DTNOW which is the local date time.

- "EnablePersistingMessaging": false
  - If true, enables the local persistent messaging storage.
- “Cluster”: false
  - True if GrabCater is clustered
- "ClusterBaseFolder": "C:\\GrabCasterQuorum"
  - If GrabCaster is clustered this is the directory containing the Root_GrabCaster common directory.
- "WaitTimeBeforeRestarting": 5000
  - If the GrabCaster point needs to restart it will wait WaitTimeBeforeRestarting value. Setting is in milliseconds.
- "WebApiEndPoint": http://localhost:8000
  - Web API exposed by the GCPoint. Each GCPoint must start on a unique port number in the same machine.
- "PointId": "{B0A46E60-443C-4E8A-A6ED-7F2CB34CF9E5}",
  - Unique GC point identification
- "PointName": "My Point Name",
  - GC point name
- "PointDescription": "My Point Description",
  - GC point description
- "ChannelDescription": "[channel description]"
  - Channel Description
- "ChannelId": "{D1AB5AD0-86F5-49A2-8A3C-ED2F60FE2E97}"
  - Unique Channel identification
- "ChannelName": "[channel name]"
  - Channel Name
- "LocalStorageConnectionString": " [GrabCaster Installation Folder]\\PersistentStorage"
  - Folder used to persist the messages if EnablePersistingMessaging == true.
- "LocalStorageProvider": "FILE"
  - Type of storage provider, the current version support FILE only.
- "LoggingStateEnabled": true
  - Enable or disable the logging system.
- "EnginePollingTime": 2000
  - Global triggers polling time in milliseconds. Current version supports global polling trigger only.
- "EventHubsReceivingPattern": "Direct"
  - Receiving pattern used by the Azure EventHubs. Current version support Direct mode only.
- "LoggingComponent": "Framework.Log.EventHubs.dll"
  - Componet used by the SLI for the logging.
- "LoggingComponent": "Framework.Log.EventHubs.dll",
- "LoggingComponentStorage": "ehlog",
  - Event Hub name used by the log component

- "LoggingVerbose": false,
  - Enable the logging verbose mode
- "DisableDeviceProviderInterface": false,
  - Disable the Device Provider Interface, the GrabCaster point will work in local mode only, useful if we don't need to use a Message Storage Provider.

14.1.1 GrabCaster Throttling

GrabCaster implements an internal throttling mechanism we can change these setting with our requirement, we can decide to work more in memory than in persistent and working in memory will drastically increase the performances.

- "ThrottlingOnRampIncomingRateNumber": 1,
  - Number of message published before to start the flushing to the DSPI
- "ThrottlingOnRampIncomingRateSeconds": 1,
  - Time in seconds before to start the flushing of the messages received to the DSPI
- "ThrottlingOffRampIncomingRateNumber": 1,
  - Number of message delivered before to start the flushing to the DSPI
- "ThrottlingOffRampIncomingRateSeconds": 1,
  - Time in seconds before to start the flushing of the messages delivered to the DSPI
- "ThrottlingConsoleLogIncomingRateNumber": 10,
  - Number of log messages before to start the flushing to console
- "ThrottlingConsoleLogIncomingRateSeconds": 1,
  - Time in seconds before to start the flushing of the logging messages received to the cosole
- "ThrottlingLSILogIncomingRateNumber": 10,
  - Number of log messages before to start the flushing to the DSPI
- "ThrottlingLSILogIncomingRateSeconds": 1
  - Time in seconds before to start the flushing of the messages received to the DSPI

14.2 TROUBLESHOOTING UNEXPECTED BEHAVIOURS AND ERRORS

GrabCaster uses different log systems: Windows Event Viewer, Event Tracking Windows and abstracted logging mechanism. The current version implements Microsoft Azure EventHubs, but we can extend to other logging systems in very easy way. GrabCaster use the Console messaging during the console running mode only.

In case of an error or some unexpected behaviour during the first execution, refer to the Window Event Viewer or Windows and Service Log.

14.3 CONFIGURE THE CUSTOM LOGGING

The Service Logging Interface SLI provides a custom logging system based on Microsoft Azure Event Hubs. To set a custom logging component we need to specify the component name in the GrabCaster configuration file in the LoggingComponent node. The current component uses is Framework.Log.EventHubs.dll.

14.3.1 Configure Framework.Log.EventHubs.dll

Create a new Azure EventHubs dedicated to the log. Then you will be able to organize these event in a Blob, Table or Azure SQL simply using the Azure Stream Analytic service, for more information refer to SETTING UP GRABCASTER in the GrabCaster site.

The log component uses the same Microsoft Azure credential specified in the configuration file.
The event hub used is specified in the "LoggingComponentStorage" key in the Grab lothe currents setting node in the GrabCasteras below:

- AzureNameSpaceConnectionString
  - Specify the Azure Connection String, we can use the same used by the MSP.
- HubName
  - Specify the EventHubs name used for the logging.

14.4 CREATE YOUR CUSTOM LOGGIN COMPONENT

You can create your own custom logging component and configure the SLI to use it.

First start by creating a new public class and inherit the `IlogEngine` interface. Decorate the class with the LogContract attribute.

```
/// <summary>
/// The log engine.
/// </summary>
[LogContract("AEC1AF21-2131-475D-AEFE-DDCA2D835466", "LogEngine", "Event Hubs Log System")]
0 references | 0 changes | 0 authors, 0 changes
public class LogEngine : ILogEngine
```

Where

- ID
  - A unique identifier
- Name
  - Specify the class name
- Description
  - Specify a description

Implementing the interface methods, the SLI will call your component using the WriteLog method that will receive an object of LogMessage type.

```
/// <summary>
/// The write log.
/// </summary>
/// <param name="logMessage">
/// The log message.
/// </param>
/// <returns>
/// The <see cref="bool"/>.
/// </returns>
1 reference | 0 changes | 0 authors, 0 changes
public bool WriteLog(LogMessage logMessage)
{
    return LogEventUpStream.SendMessage(logMessage);
}
```

If WriteLog = false the SLI will consider it an internal critical error and it will ignore the call.

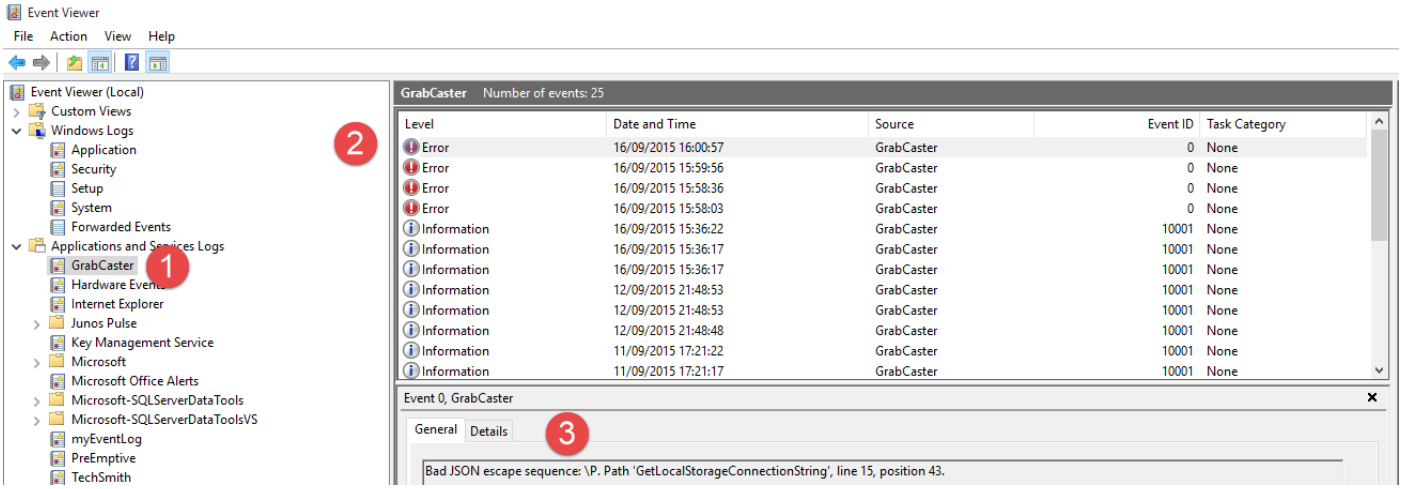
Compile the component and copy the DLL in the C:\Program Files (x86)\GrabCaster\Root_GrabCaster. Insert the complete file name with extension in the **LoggingComponent** node in the GrabCaster configuration file.

For the complete code refer to the GrabCaster.Framework.Log.EventHubs component in the GrabCaster.Framework repository in GitHub.

14.5 WINDOWS EVENT VIEWER

To open the windows viewer refer to the operating system version.

Windows 8 and 2012



GrabCaster sends errors and notification into the event viewer where:

1) GrabCaster application log
2) Windows Event viewer general logging
3) Windows Event viewer detail