Portrait
Foundation

Request Token Tracing Overview

Edition 2.2

11 January 2013

Pitney Bowes
Software

Portrait Fundation
Request Token Tracing Overview

Acknowledgement of trademarks

About Portrait Software

Portrait Software is now part of Pitney Bowes Software Inc.

Portrait Software enables organizations to engage with each of their customers as individuals, resulting in improved customer profitability, increased retention, reduced risk, and outstanding customer experiences. This is achieved through a suite of innovative, insight-driven applications which empower organizations to create enduring one-to-one relationships with their customers.

Portrait Software was acquired in July 2010 by Pitney Bowes to build on the broad range of capabilities at Pitney Bowes Software for helping organizations acquire, serve and grow their customer relationships more effectively. The Portrait Customer Interaction Suite combines world leading customer analytics, powerful inbound and outbound campaign management, and best-in-class business process integration to deliver real-time customer interactions that communicate precisely the right message through the right channel, at the right time.

Our 300 + customers include industry-leading organizations in customer-intensive sectors. They include 3, AAA, Bank of Tokyo Mitsubishi, Dell, Fiserv Bank Solutions, Lloyds Banking Group, Merrill Lynch, Nationwide Building Society, RACQ, RAC WA, Telenor, Tesco Bank, T-Mobile, Tryg and US Bank.

Pitney Bowes Software Inc. is a division of Pitney Bowes Inc. (NYSE: PBI).

For more information please visit: http://www.pitneybowes.co.uk/software/

UK

Portrait Software
The Smith Centre
The Fairmile
Henley-on-Thames
Oxfordshire, RG9 6AB, UK

Email: support@portraitsoftware.com
Tel: +44 (0)1491 416778
Fax: +44 (0)1491 416601

America

Portrait Software
125 Summer Street
16th Floor
Boston, MA 02110
USA

Email: support@portraitsoftware.com
Tel: +1 617 457 5200
Fax: +1 617 457 5299

Norway

Portrait Software
Portrait Million Handshakes AS
Maridalsveien. 87
0461 Oslo
Norway

Email: support@portraitsoftware.com
Tel: +47 22 38 91 00
Fax: +47 23 40 94 99

About this document

Purpose of document

This document gives a detailed explanation of how requests may be traced through Portrait Foundation using Request Tokens. It describes how performance information may be logged by request, and how Request Tokens may be passed in and out of Portrait Foundation in calls to web services.

Intended audience

This document is intended for use by anyone developing extensions for or integrations with Portrait Foundation.

It will also be of use to anyone interested in monitoring Portrait Foundation performance in a live system.

Related documents

Generic Access Implementation Guide

WCF Service Support Developers Guide

Software release

Portrait Foundation 4.4 Update 2 or later.

Contents

1 Overview

The request token tracing mechanism allows you to log the progress of requests in Portrait Foundation through the various tiers. This helps identify the proportion of time that is being spent in each part of the system, including the web server, the Process Engine, DATs and external systems called.

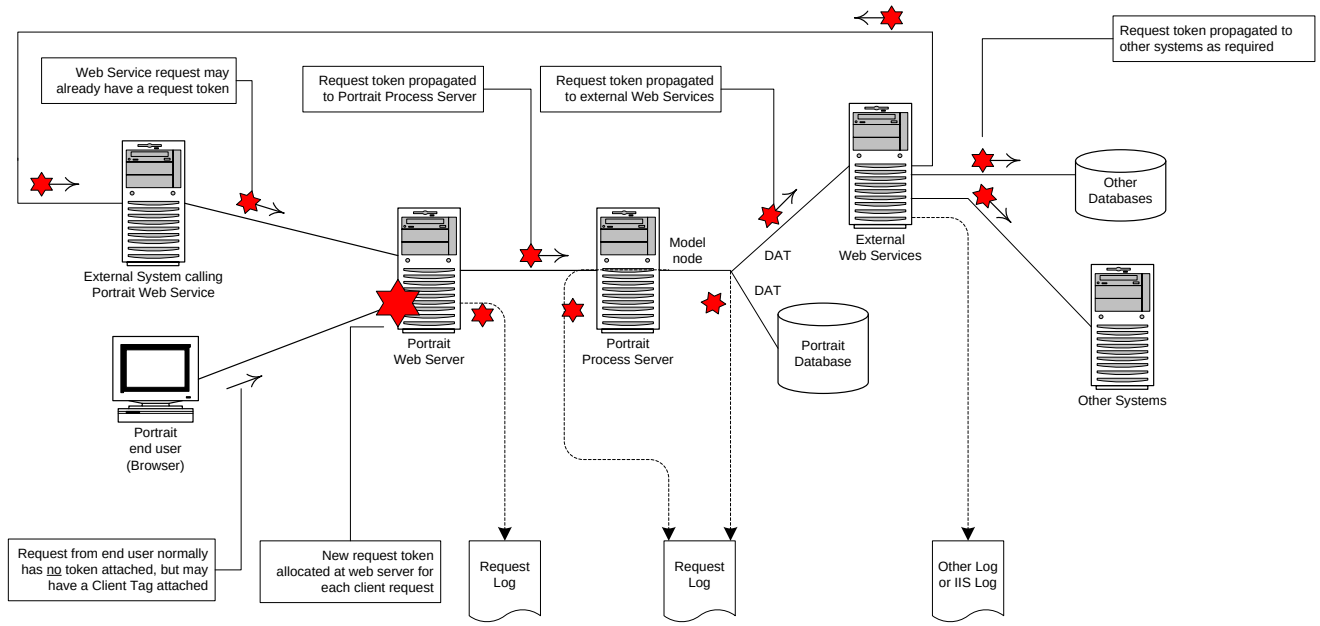
1.1 Main features

- A Request Token is allocated for every request that arrives at the Portrait web tier.
- This is passed down out of band[1] as the request is processed, and timing information may be logged at various layers of the system.
- The Request Token may be passed out to any external web services called.
- External systems calling Portrait's web services may pass in a Request Token to be used instead of one being allocated.
- The Portrait web application can pass a Client Tag with every request, which gets logged along with the Request Token and represents "what the user has just done".
- Request Token logging has been implemented to have the minimum performance impact and is therefore feasible to have enabled (at least at some layers) in a live system.

1.2 Architectural overview

Figure 1 shows the architecture in a simplified Portrait Foundation system. For the purposes of this diagram it is assumed that the Portrait Web Server and Process Server reside on different boxes.

Figure 1 – Overview of the solution components



- A request from the Portrait Foundation end user arrives at the Portrait Web Server and a new Request Token is allocated:

[1] The term *out of band* refers to passing data between components without explicitly adding it to interfaces. A number of mechanisms exist, including the use of custom SOAP or HTTP headers or COM "channel hooks".

  - Optionally timing information by request may be logged at this point.
- The Request Token is passed with the request to the Process Server:
  - Timing information may be logged for requests arriving at the Process Server.
  - Timing information may also be logged for calls to DATs, configurable by individual DAT.
- The Request Token may be passed on to external web services called.
- If they result in a call back into a Portrait web service, the same Request Token may be passed back in, so that the whole sequence is associated with the original request.
- Data logged by this mechanism is to dedicated CSV files located on the servers. It is independent of the normal Portrait Foundation logging mechanism.
- Aggregation or other processing of the data logged by this mechanism is performed outside of Portrait Foundation.

1.3 Anticipated usage

Care has been taken to ensure that Request Token logging has the minimum impact on performance. It is therefore feasible to enable a certain amount of logging even in a live system.

It is anticipated that it may be enabled at the following points in a live system:

- All requests arriving at the Portrait Web Server.
- For specific DATs that are known or suspected to have performance issues – for example those that call external web services.

Logging of requests arriving at the Portrait Process Server and for other DATs can be enabled as and when required.

Note that large amounts of data will be generated, so mechanisms will have to be put in place for "housekeeping" – to move the log files off the live servers as necessary.

1.4 Data volumes

As a rule of thumb, each record logged by this mechanism will average 300 bytes.

A user action at the Portrait web site will result in one or more requests arriving at the Portrait Web Server. In this context a "request" at the Portrait Web Server equates to running a model, operation or group of custom controls.

So, in a system that processes an average of five Portrait Web Server requests per second over a 12 hour day, Request Token logging at this level would result in around 60 Megabytes per day on that server:

300 bytes x 5 per second x 60 seconds x 60 minutes x 12 hours

To calculate the size the Request Token log files on the Process Server, you can calculate on the following basis:

- Each request at the Portrait Web Server generally corresponds to one request at the Portrait Process Server.
- Each request at the Process Server might result in any number of DATs being called, but typically between 1 and 10. However, it is unlikely that you would want to enable logging for every DAT in the system.

Configuration of this mechanism allows you to specify a maximum log file size, after which the log file is archived with a unique name and a new one started.

1.5 Use cases

1.5.1 Performance post mortem

A user of the live system calls the helpdesk as follows:

User: "Portrait Foundation was slow yesterday afternoon, about 3pm"

Helpdesk: "What were you doing, exactly?"

User: "I'm not sure – something to do with Accounts I think"

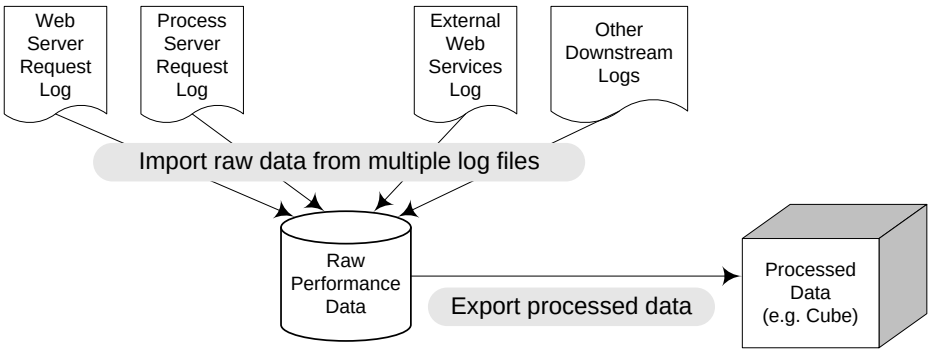
Analysis of the Portrait Web Server request log file will enable the helpdesk to:

- confirm that performance for this user was indeed poor, and at what time;
- identify exactly what the user was doing at the time;
- identify whether the poor performance was due to Portrait Foundation, or a call to an external web service. Is another system in fact to blame?

1.5.2 System health monitoring

Request Token log files are generated from the live system and "sliced and diced" to monitor performance over time.

Figure 2 – Processing of multiple logfiles



Performance data could be analysed in many ways including:

- by Activity Token (for a given user's session)
- by Client Tag (for a given user action)
- by Request Token (for a given request)
- by request type
- by Operation
- by top-level Model
- by DAT

This allows monitoring of system performance over time, or definition of meaningful Service Level Agreements (SLAs).

2 Detailed design

2.1 Allocation of request tokens

A request token is allocated in the web server for any request that does not already have such a token (see sections 2.3 and 3.1 for details).

The request token is a GUID in order to ensure uniqueness across all web servers, for example:

```
3631EBB9-6CAF-4dd8-AEA0-F6D2C9F66F52
```

It is not necessary to have Request Token tracing enabled at the Web Server in order for a request token to be allocated.

2.2 Client tags

Any request arriving at the Portrait Foundation web tier may contain a client tag. This is a free-format piece of text that describes in some way what the user is currently doing. For example, to indicate that they have pressed a button on a particular dialog, or that they have selected a tab page.

This client tag may be passed to the web server along with each request generated by the user action. Thus if selecting a tab page results in two web server requests, the same client tag would be passed to both, to indicate that these requests were caused by the user selecting this tab page.

The Portrait Foundation web site has been modified to generate such client tags. While they may not represent with 100% accuracy what the user has just done – for example, it is easy for a project team to customise the web site in order to defeat this – in most cases they give a good indication of what the user was doing and which requests originated from the same user action.

Also, the Portrait Generic Service has been modified to generate a suitable client tag when Portrait web services are called. You can however override this as described in section 3.1 with your own value.

2.3 Portrait Web Channel Enabler, Client Tag, Request Token

Any request to the Portrait Web Channel Enabler may include a request token and optionally a client tag. These are passed in the request string as **PortraitRequestToken** and **PortraitClientTag** respectively.

If a request token is passed in, it must be a valid GUID. It will then be used for this request, instead of a new one being allocated.

2.4 Logging data in the Web tier

This can be enabled and disabled in the Portrait Management Console, as described in section 2.8.1. Logging takes place for each request that arrives, with data as specified in section 2.9. The elapsed times recorded at this level correspond to the actual wait times experienced by the user, excluding any network latency.

2.5 Passing request tokens to the Process Server

The request token is passed from the Portrait Web Server to the Process Server using the COM channel hook mechanism. This holds the data in Thread Local Storage (TLS). When new threads are spawned within the Process Engine the TLS data is copied to each such thread.

2.6 Logging performance data in the Process Server

This takes place in the Process Server Dispatcher component, and can be enabled or disabled in the Portrait Management Console as described in section 2.8.2. Times recorded at this level *include* any time spent waiting for dispatcher threads to become available.

2.7 Logging of performance data from Nodes

Logging of performance data is performed in the Data Access Node and Data Access Class components, at the point where Data Access Transactions are called.

The node activity is logged from the Process Server at the point of calling the node (that is, not within the DATs themselves) in order to implement it at a single common point. Hence the database access times are those observed at the Process Server, and include invoking the DATs and any network times between the Process Server and the database server. This is felt to be the most useful point at which to measure, since it records the total time involved in calling that DAT. It is not however possible to distinguish time spent in the DAT from time spent with the database without further customisation of specific DATs.

2.8 Configuration of Request Token tracing

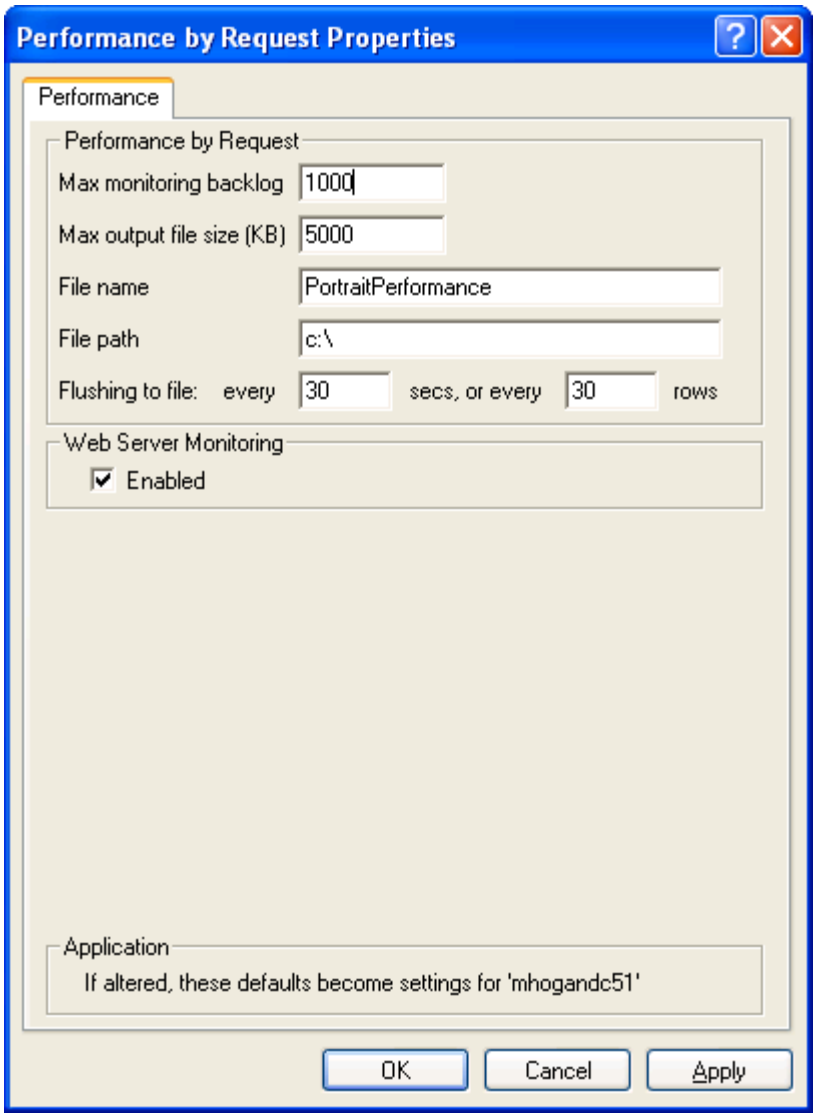
Configuration of this mechanism in the various layers of Portrait Foundation may be done using the Portrait Management Console.

2.8.1 Web Tier

Configuration in the web tier may be done through the following node in the Portrait Management Console:

Web Channel Enablers*serverName*\Performance By Request

Selecting Properties on this node brings up the following dialog, through which settings can be configured.



Max monitoring backlog – in order to minimise the impact on performance, logging takes place on a separate thread from the actual processing of requests and can therefore build up a backlog when the system is busy. If the backlog exceeds this number of entries, potential log entries will be discarded.

Max output file size (KB) – when the log file reaches this size it is archived and a new one started, as described in section 2.10.

File name – base file name for the log file.

File path – folder where the log file will be created – should be local to the server.

Flushing to file – log file entries will be written out when these thresholds are reached. Should be around 30 to avoid contention on the file, though while testing you might want much lower values so changes can be examined immediately.

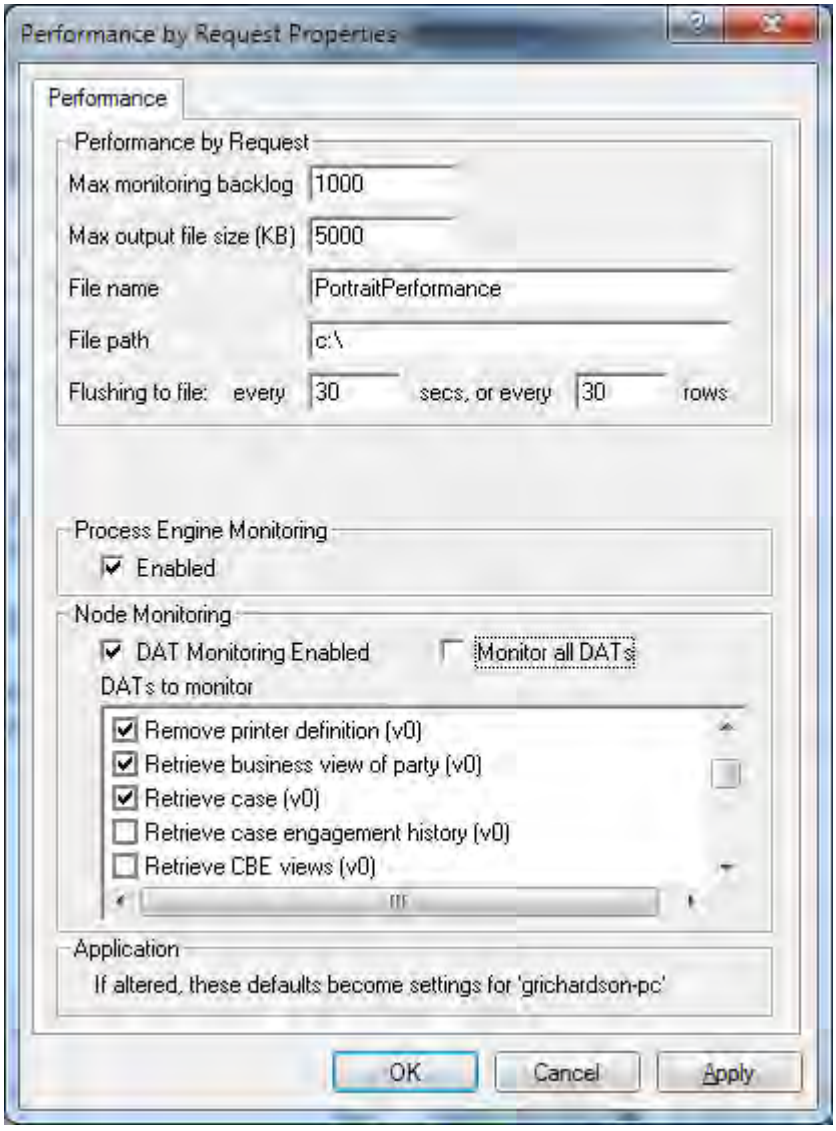
Enabled – tick here to enable logging at this level.

2.8.2 Process Engine and Nodes

Configuration of PPM at entry to the Process Engine and at nodes within models will be done through the following new node in the Portrait Management Console:

CRM Servers*serverName*\Performance By Request

Selecting Properties on this node brings the following dialog, through which settings can be configured.



The settings in the upper section are as described in section 2.8.1.

The two "Enabled" checkboxes enable logging at:

- Calls arriving at the Process Engine (the Dispatcher)
- DATs – enables logging at the DATs ticked in the list below

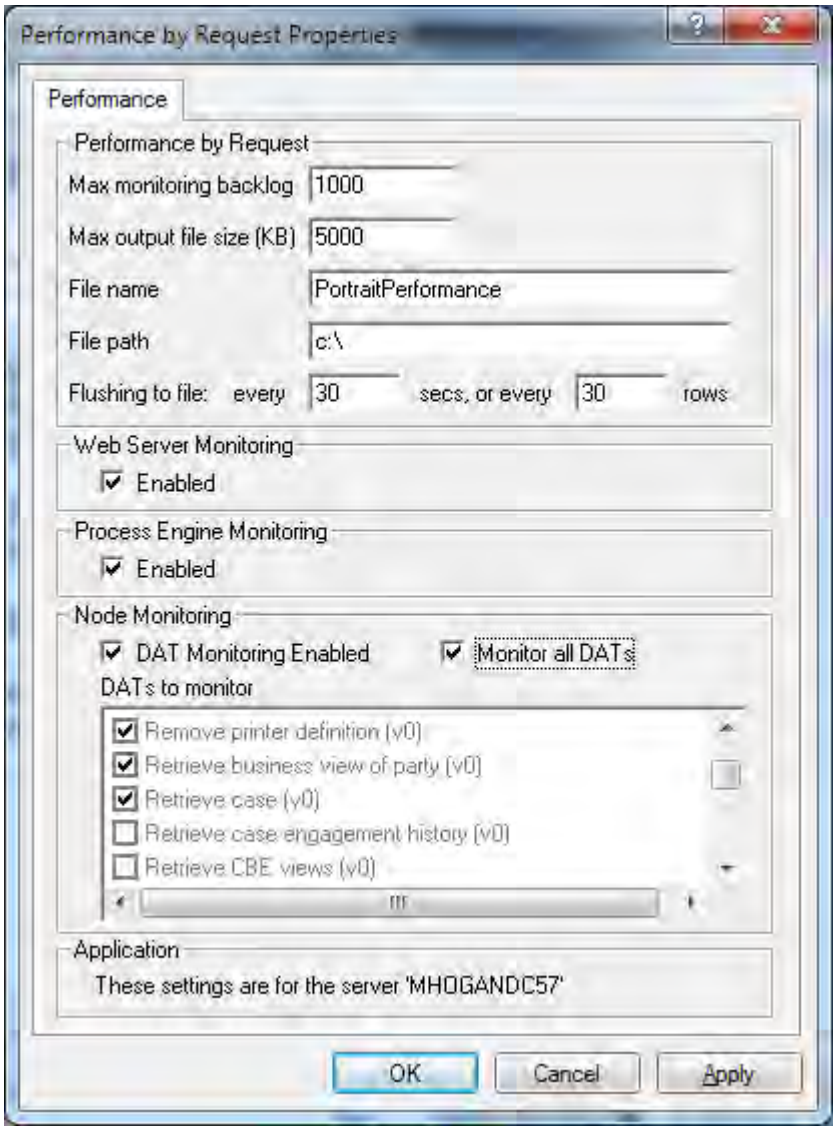
The *DATs to monitor* list box lists all Data Access Transactions currently deployed in the system.

- *If you check the Monitor* all DATs checkbox then all DATs in the system will be monitored and the *DATs to monitor* list box is disabled and ignored. Using this option has the advantage that any new DATs added subsequently will automatically be monitored.

2.8.3 Single-box environments

All logging that uses this mechanism on a given server is routed to a single file.

Therefore, in a single-box environment (for example, web server and process server on the same machine) it can be configured at every level through a single property page as shown below.



The settings on this page are as described in sections 2.8.1 and 2.8.2.

2.9 Data format

Files are written in CSV format. The first line in each file is a line of headers describing the data that follows.

The records written from each level in the software (Web, Dispatcher and Nodes) have the same overall format as described below, although not every field is written from each tier.

Records are not written in strictly chronological order due to the mechanisms employed for performance reasons.

Note: It will always be necessary to sort the data on *Time of Call* before exploring it further.

In some cases the range of times covered by two consecutive files will overlap.

Tips: if you open the file in Excel, select the first column and:

- In order to be able to see the entire date/timestamp, choose Format – Cells – Number tab – Custom – dd/mm/yyyy hh:mm:ss.000
- Sort the entire file on this first column, in order to see records in chronological order

Field Name	Value	Web tier	Dispatcher	Nodes	Notes
Time of Call	DateTime, format: dd-mm-yyyy hh:mm:ss.000	Yes	Yes	Yes	Time this call started. This will be recorded as local time for the server on which the call is logged.
ClientTag	String	Yes			Free-format text passed from the client application to indicate what the user is currently doing.
Activity Token	Double-GUID	Yes	Yes	Yes	This remains constant for a given user's Portrait Session.
Request Token	GUID	Yes	Yes	Yes	The request token, unique for each incoming request
Level	The level in the code from which this record was logged	Yes	Yes	Yes	"Web", "Dispatcher" or "Node"
Server	Network name of the server from which this record was logged	Yes	Yes	Yes	
Remote Username	Domain\UserId	Yes			Only available on requests that come through an authenticated web site, as opposed to one with anonymous access enabled.
Remote IP Address	IP address	Yes			Available as per Remote Username
ASP Page	The name of the ASP page being run.	Yes			Whenever this information is available
CRM Server	Network name of the CRM Server to which this call was routed and/or "UIECache"	Yes			If some custom controls (UIEs) were routed to a server and some satisfied from the UIE cache, contains "CRMServerName/UIECache"
Duration	Duration of the call in milliseconds	Yes	Yes	Yes	
Call Type	String to indicate the type of call being performed	Yes	Yes	Yes	Different values according to the tier from which the logging takes place. Eg: "Message:ExecuteOperation", "Message:RetrieveUIEData", "Message:ExecuteModel", "DAN" (DataAccessNode), "DAC" (DataAccessClass)
Resume	Flag to indicate if this is a resume operation	Yes			1 or 0. This is only logged from the web tier.

Field Name	Value	Web tier	Dispatcher	Nodes	Notes
Call Name	The system name of the operation, model, DAT or custom control (UIE) being performed	Yes		Yes	Eg “SelectCustomer”, “ContactCentre” etc. Multiple custom control requests (UIEs) will be separated by colons. In the case of DATs the version number will be appended, eg “SaveParty (v0)”
Call Details	Additional text to qualify the Operation Name	Yes		Yes	Only logged where useful extra data is available - Node level – contains a colon-separated call stack of ancestor models - Web level – if Resume is 0 (ie starting a model, operation or custom controls) contains the system name and version of the model invoked. No version shown means “latest”.
Thread Id	The thread Id	Yes	Yes	Yes	Helps to understand the following two per-thread processor time values by identifying which thread processed the request
Kernel Processor Time	Per-thread kernel-mode processor time, in milliseconds	Yes	Yes	Yes	NB: the granularity of the processor time measurements is the OS scheduling time-slice, ie about 15 ms.[2]
User Processor Time	Per-thread user-mode processor time, in milliseconds	Yes	Yes	Yes	

2.10 Data file management

Data is written to each logfile until it reaches a maximum size, which can be configured as specified in section 2.8.

Once the maximum size is attained, the file is closed and renamed for archive, and logging started to a new file. The archive filenames are constructed as follows:

yyyymmddhhmmssmmm_originalfilename

where *yyyymmddhhmmssmmm* represents the date and time of archiving.

[2] Where one thread spawns a number of others – for example to complete different execution paths within a model – the processor time recorded is for that thread alone and does not include time consumed by any spawned threads. It is important to realise that in some cases these processor times will not be particularly meaningful. For example, where a given thread does nothing except spawn some other threads to do the work, its processor time will not be very illuminating.

2.11 Measured performance impact

The impact of this mechanism has been measured as follows:

- With recording on at every level and for every DAT (not just those that call web services): less than 2 percent

However, the precise performance impact depends on your system configuration. For example if your system involves calling a lot of web services or accessing databases (and hence quite a lot of waiting in the Process Server) it may have no detectable impact at all. However, in a system with a high degree of processing the impact could be more significant.

3 Integration with external web services

Where DATs call external web services, the request token may be passed to them in a custom SOAP header in order to allow these calls to be linked back to the original web tier request that initiated them.

Conversely, where Portrait Foundation exposes web services that can be called by other systems a mechanism has been provided to pass in a request token that should be used instead of a new one being generated.

This allows external web services that are called by Portrait Foundation and then call back into Portrait Foundation again to pass the same request token through, so the whole sequence may be linked back to the original web tier request.

In all cases the SOAP header is optional, so it is not necessary for the external systems to support it or be changed in any way.

3.1 Portrait web service, Request Token and Client Tag

Where another system calls a Portrait Foundation web service it is permitted (but optional) to pass in a request token. This might either be an existing Portrait Foundation request token that was previously passed in a call to a web service, or a new request token generated by the calling system. If no such request token is provided, Portrait Foundation will generate a new one in the web tier.

In order to support efficient analysis of the data we recommend that this should be unique for each successive call, and it must be a valid GUID, for example:

```
3631EBB9-6CAF-4dd8-AEA0-F6D2C9F66F52
```

Similarly, a client tag may also be passed in if required (see section 2.2). If none is provided then one will be generated by Portrait Foundation to indicate which web service the request came through.

The mechanism used is a custom SOAP header called StateHeader which may be passed using XML similar to the following:

```
<soap:Header>
  <types:StateHeader id="id">
    <requestToken xsi:type="xsd:string">token</requestToken>
    <clientTag xsi:type="xsd:string">tag</clientTag>
  </types:StateHeader>
</soap:Header>
```

This SOAP header is supported by the Portraiit Generic Service and by all other web services generated by Portrait Foundation. If you generate .NET proxies for these web services they will include a member *StateHeaderValue* of class *StateHeader* to allow these values to be passed if required, eg:

```
service.StateHeaderValue = new StateHeader();
service.StateHeaderValue.requestToken = token;
service.StateHeaderValue.clientTag = tag;
// now call the web service method
```

This is an In/Out SOAP header: if no request token is passed in then in the response XML it will contain the request token generated by Portrait Foundation.

3.2 Passing the request token to external Web Services

The request token is available to external web services in a custom SOAP header as follows:

```
<soap:Header>
  <types:StateHeader id="id">
    <requestToken xsi:type="xsd:string">token</requestToken>
  </types:StateHeader>
</soap:Header>
```

The SOAP header is always passed with the **mustUnderstand** attribute set to 0, to indicate that it is optional whether the recipient processes it. This ensures that existing web services continue to run unchanged.

Web services are generally called using one of three general mechanisms:

- Custom DATs written in C++ or C#
- DATs using the Host Integration Framework and the XMLHTTP adapter
- DATs using the Generic Web Service Architecture (GWSA)

3.2.1 Passing Request Token in custom DATs (C++ or C#)

These may be modified by adding code to access the request token from Thread Local Storage (TLS) and adding the SOAP header to the request.

Classes have been added to the Portrait SDK to allow access the request token as follows.

Note: You can only gain access to this from a component that is called directly by a Portrait Foundation thread.

3.2.1.1 C++

```
#include "RequestToken.h"
CAmcRequestToken requestToken;
// Get the request token from thread-local storage
if (CAmcPortraitChannelData::GetRequestToken(&requestToken))
{
        // Get a string representation of the request token
        CComBSTR bstrRequestToken;
        hr = requestToken.GetAsBSTR( &bstrRequestToken );
        // Now add the request token in the SOAP header
        ...
```

You will also need to link your project to PortraitLibU.lib.

3.2.1.2 C# (or Visual Basic)

Your project will need a reference to AIT.Portrait.Interops.

```
using PortraitNETLib;
// Instantiate a ChannelData object
ChannelData cd = new ChannelDataClass();
// Get the request token from it
string sRequestToken = cd.RequestToken;
// Now add it to the SOAP header
...
```

3.2.2 Passing request token in DATs that use HIF XMLHTTP Adapter

A new adapter is available called **FoundationSOAPHeaderAdapter**. This scans the incoming XML for a SOAP envelope, and if it finds one it adds the StateHeader SOAP header, including the request token.

Thus, to pass the request token to a web service using this mechanism, all that is required is to add this to the adapter stack for this system, before the XMLHTTP adapter sends the request out via HTTP.

3.2.3 Passing the request token using GWSA

This is done automatically: the GWSA adapter adds the SOAP header in all cases to the outgoing request.

3.3 Receiving request token in external Web Services

Processing of these SOAP headers by a web service is optional, so existing web services will run correctly without change.

However, if you wish to modify them to receive and process the request token further you can do it as follows.

3.3.1 .NET Web Services

Web services written in .NET can be modified to handle the StateHeader SOAP header by adding a class:

```
using System.Web.Services.Protocols;
public class StateHeader : SoapHeader
{
        public string requestToken;
}
```

and a public member variable:

```
public StateHeader StateHeaderValue;
```

which should be allocated in the constructor:

```
StateHeaderValue = new StateHeader();
```

Then modify the declaration of the web service method(s) by adding the following attribute:

```
[SoapHeader("StateHeaderValue", Direction=SoapHeaderDirection.In)]
```

Within the body of the method you can then access the member StateHeaderValue to see if the header has been passed and to get its requestToken.

3.3.2 Web Services implemented using the SOAP Toolkit

To access a SOAP header from within a web service implemented using the SOAP Toolkit is slightly more involved.

1. Implement the interface IHeaderHandler in a COM object within your project (this may or may not be the same object that implements the web service).
2. Modify the generated .wsml file to indicate the class that implements the IHeaderHandler interface, for example:

<port name='PortName' headerHandler='HeaderHandlerClass'>

The IHeaderHandler interface will then be called for each SOAP header passed to the web service, and can extract the requestToken using standard XML DOM functionality.