Data Collectors

On this page:

Works with:

Data collectors allow you to supplement business transaction and transaction analytics data with application data. The application data can add context to business transaction performance issues. For example, they show the values of particular parameters or return value for business transactions affected by poor performance.

This data shows the business context affected by performance issues, such as the specific user, order, or product.

Types of Data Collectors

Data collectors come in two forms:

- Method invocation data collectors capture code data such as method arguments, variables, and return values.
- HTTP data collectors capture the URLs, parameter values, headers, and cookies of HTTP messages exchanged in a business transaction.

For information on how transaction analytics uses data collectors, see Collect Transaction Analytics Data.

View Collected Data

When applied to business transactions, the data collectors supplement the information shown in transaction snapshots. The information captured by HTTP data collectors appears in the HTTP DATA and COOKIES sections, while method invocation data appears in the Business Data section. To view collected data, navigate to the **Transaction Snapshots** window. Double click on the transaction you want to view data for, then click **Drill Down**.

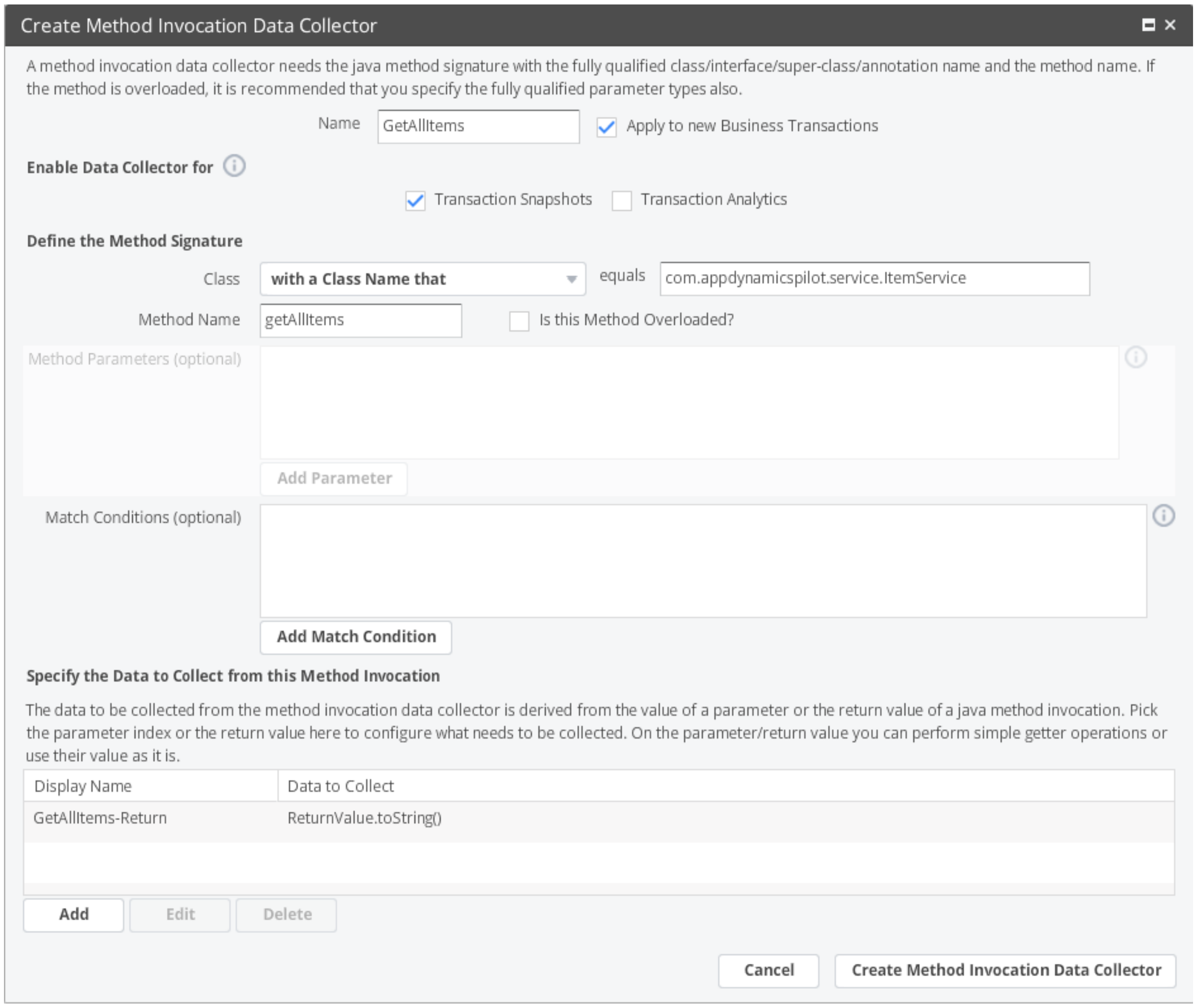
Configure a Data Collector

To configure data collectors, you need the **Configure Diagnostic Data Collectors** permission**.**

You can add a data collector directly from a transaction snapshot from the right-click menu for a method in a call graph.

Alternatively, you can add it manually from the **Configuration** > **Instrumentation** page. Click the **Data Collector** tab and click the **Add** button below the Method Invocation Data Collectors panel or the HTTP Request Data Collectors panel.

The new data collector configuration window appears as follows:


Create Method Invocation Data Collector

A method invocation data collector needs the java method signature with the fully qualified class/interface/super-class/annotation name and the method name. If the method is overloaded, it is recommended that you specify the fully qualified parameter types also.

Name: GetAllItems [✓ Apply to new Business Transactions]

Enable Data Collector for

[✓ Transaction Snapshots] Transaction Analytics

Define the Method Signature

Class: with a Class Name that equals com.appdynamicspilot.service.ItemService

Method Name: getAllItems Is this Method Overloaded?

Method Parameters (optional)

Add Parameter

Match Conditions (optional)

Add Match Condition

Specify the Data to Collect from this Method Invocation

The data to be collected from the method invocation data collector is derived from the value of a parameter or the return value of a java method invocation. Pick the parameter index or the return value here to configure what needs to be collected. On the parameter/return value you can perform simple getter operations or use their value as it is.

Display Name	Data to Collect
GetAllItems-Return	ReturnValue.toString()

Add Edit Delete

Cancel Create Method Invocation Data Collector

Note the following platform-specific considerations applicable to data collectors:

- For the C/C++ agent, you can create a method data collector using the `appd_bt_add_user_data function()`, not the Controller UI as described here. See Agent SDK for C and C++.
- For the Node.js agent, you can create a method data collector only using the `addSnapshotData()` Node.js API, not the Controller UI as described here. See Node.js Agent API Reference.
- For the Python agent, you can create a method data collector using the `add_snapshot_data()` Python agent API, not the Controller UI. See Python Agent API Reference.
- For PHP method data collectors, only the "with a Class Name that" option is valid. Also, you cannot add a method data collector to a standalone PHP function.

The general steps in configuring a data collector are:

1. Identifying the method on which to capture data. To do this, you define the method signature and optionally, filters based on the value of a code point in the method (such as return value or argument).
2. Specifying the actual code point that serves as the source of the data.
3. If the data collector applies to business transactions, choose the applicable business transactions.

Typically, creating a data collector requires knowledge of the code on which you are setting up the collector, whether based on access to the source code for the application or its documentation. For some application environments, including JVM 1.5 and .NET, you will need to restart the JVM or application server if your method invocation data collector results in modifications to instrumentation configuration (class name, method name, method parameters, and so on).

Configuration Notes

- The **Apply to new Business Transactions** option applies the collector to business transactions created after you have configured this collector. Otherwise, the data collector only applies to the business transactions you select in the subsequent data collector configuration screen.
- For **Class**, select the match condition that the data collector can use to identify the class, such as class name, implemented interface name, and so on. If matching by class name, use the fully qualified name of the class, as appropriate for the application platform. For example, the form of the equals field value would be:
  - In Java: `com.appdynamics.model.Item`
  - In .NET: `Bookstore.Item`
  - In PHP: `book`
- Is this Method Overloaded: If this is an overloaded method, add parameters that identify the signature. You will need to create a data collector definition for each form of the method for which you want to capture data. For example, given the overloaded method in the following table, to capture data only for the second two forms, you would need to create two data collectors. The parameters to configure would be as shown:

Signature	Parameters to configure
`getName()`	• None
`getName(int studentId)`	• Param Index 0: java.lang.int
`getName(int studentId, string name)`	• Param Index 0: java.lang.int • Param Index 1: java.lang.String

- You can refine the method selection to meet specific conditions. If you configure more than one match condition, all match conditions must be satisfied by the request for the data collector to be applied to the request.
- Once you identify the method, specify the code point from which you want to capture data, such as the method return value, argument, or a value captured by getter chain on the invoked object. You configure this code point in the **Specify the Data to Collect from this Method Invocation** section of the configuration settings.
- HTTP data collector can capture data from HTTP parameters, request attributes, cookies, and more. Notice that the Content-Length HTTP header is already captured in AppDynamics as the Average Request Size metric. However, you may choose to configure a data collector for this header to have the value appear in transaction snapshots, giving you insight, for example, on whether message size corresponds to slow performance.
- You can configure multiple data collectors. The effect of multiple data collectors is cumulative; if you add a custom data collector that does not include collection of the URL HTTP request attribute, for example, but keep the Default HTTP Request Data Collector configuration in which the URL is configured to be collected, the URL is collected.

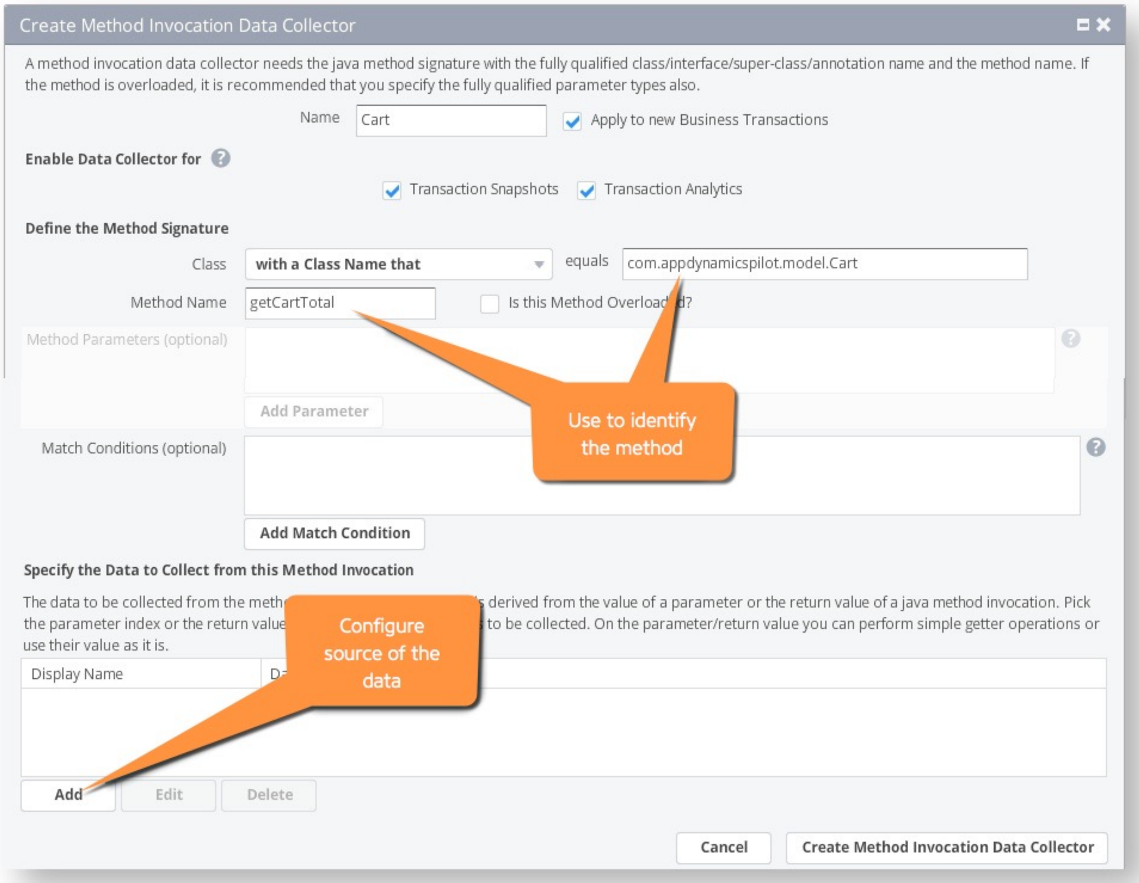
Java Data Collector Example

In this example, we'll set up a data collector on a Java application. We want to create a data collector on the method `getCartTotal()` , shown in the following snippet, along with a method we'll use later as a data source, `getUser()`.

```
package com.appdynamicspilot.model;
...
public class Cart implements java.io.Serializable {
    ...
 private Double fakeAmount = 0.0;
    ...
 private User user;
    ...
 public User getUser() {
  return user;
 }
 ...
    public Double getCartTotal() {
  if (fakeAmount == 0.0) {
   double total = 0;
   if (items != null) {
    for (Item item : items) {
     total += item.getPrice();
    }
   }
   return total;
  } return fakeAmount;
 }
   ...
}
```

To set up a sample data collector for this application:

1. Identify the method using the fully qualified name of the class and getCartTotal as the method.


Create Method Invocation Data Collector

A method invocation data collector needs the java method signature with the fully qualified class/interface/super-class/annotation name and the method name. If the method is overloaded, it is recommended that you specify the fully qualified parameter types also.

Name: Cart ✓ Apply to new Business Transactions

Enable Data Collector for

✓ Transaction Snapshots ✓ Transaction Analytics

Define the Method Signature

Class: with a Class Name that equals com.appdynamicspilot.model.Cart

Method Name: getCartTotal ☐ Is this Method Overloaded?

Method Parameters (optional)

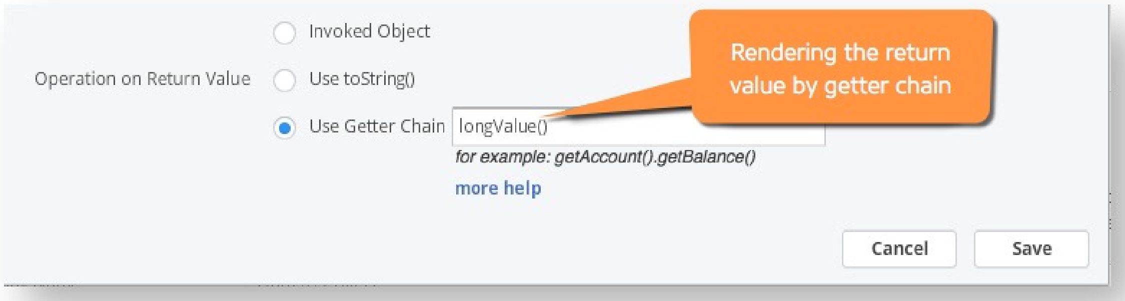
Match Conditions (optional)

Specify the Data to Collect from this Method Invocation

The data to be collected from the meth… derived from the value of a parameter or the return value of a java method invocation. Pick the parameter index or the return value… to be collected. On the parameter/return value you can perform simple getter operations or use their value as it is.

Add Parameter
Add Match Condition
Use to identify the method
Configure source of the data
Display Name
Add
Edit
Delete
Cancel
Create Method Invocation Data Collector

2. Configure the return value data source as follows:
3. Choose return value as the source type. Often, the return values are rendered as strings using the `toString()` method. Since the return value in our example is numeric (a `double`), we can use a method in the Java base packages to render the number as a long value:


Operation on Return Value: ○ Invoked Object ○ Use toString() ● Use Getter Chain: longValue()

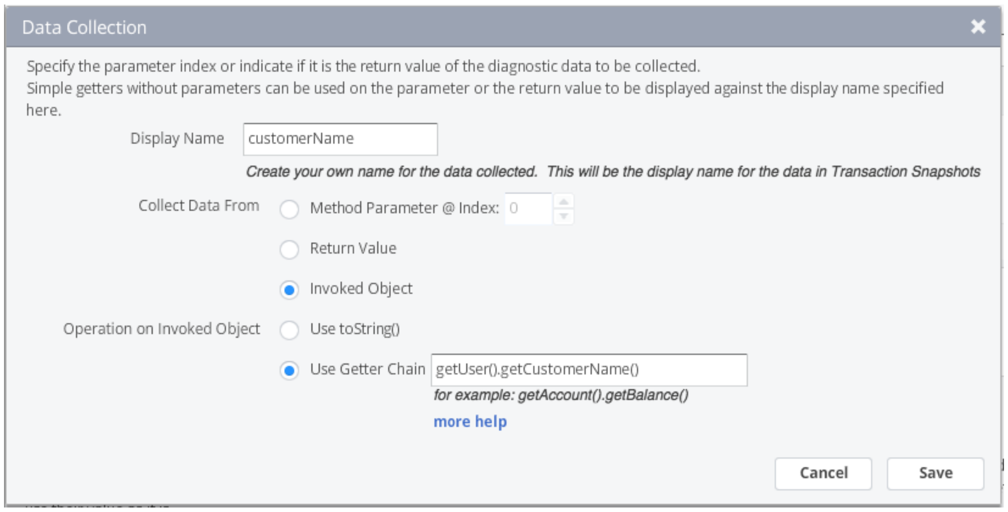
for example: getAccount().getBalance()

4. Capture the user name on the invoked object. The Cart class instantiates a User object based on the following class in the same package as `Cart`. Notice that the User class includes a method for returning the name of the user, `getCustomerName()`.

```
package com.appdynamicspilot.model;
...
public class User implements java.io.Serializable {
    ...
 private String customerName = null;
    ...

 public String getCustomerName() {
  return customerName;
 }
 public void setCustomerName(String customerName) {
  this.customerName = customerName;
 }
    ...
}
```

Using a getter chain, you can identify this method as another data source in the same data collector. For this case, select **Invo ked Object** as the source of the data. The getter chain `getUser().getCustomerName()` the operation on the invoked object:


Data Collection

Specify the parameter index or indicate if it is the return value of the diagnostic data to be collected.
Simple getters without parameters can be used on the parameter or the return value to be displayed against the display name specified here.

Display Name: customerName
Create your own name for the data collected. This will be the display name for the data in Transaction Snapshots

Collect Data From: Method Parameter @ Index: 0 / Return Value / [✓ Invoked Object]

Operation on Invoked Object: Use toString() / [✓ Use Getter Chain] getUser().getCustomerName()
for example: getAccount().getBalance()

For an HTTP data collector, the user name is sometime available as a variable value (e.g., userid) in the HTTP query or cookie. By creating a data collector on userid, you can similarly identify affected users in transaction snapshots for slow transactions with an HTTP data collector.

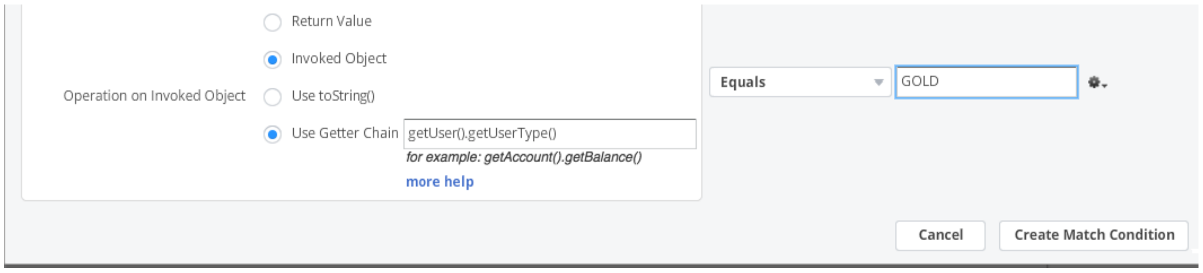
5. Optionally apply a capture condition. Our sample application differentiates users by types that correspond to service levels, diamond, platinum, and so on, as shown in the following snippet of the User class:

```
package com.appdynamicspilot.model;

public class User implements java.io.Serializable {
 public enum CUSTOMER_TYPE  {DIAMOND,PLATINUM,GOLD,SILVER,BRONZE};
    ...

 public CUSTOMER_TYPE getCustomerType() {
  return this.customerType;
 }
}
```

To capture the user name and the cart total for only gold level users, you could add a match condition that collects data from the Invoked Object and uses the following getter chain as the operation on the invoked object, using the condition Equals GOLD:


Operation on Invoked Object: Return Value / [✓ Invoked Object] / Use toString() / [✓ Use Getter Chain] getUser().getUserType()

for example: getAccount().getBalance()

Equals GOLD

6. For the PHP agent, if a method return value collected by MIDC is not stored in any variable then it is seen as null in both snapshot and Analytics data.

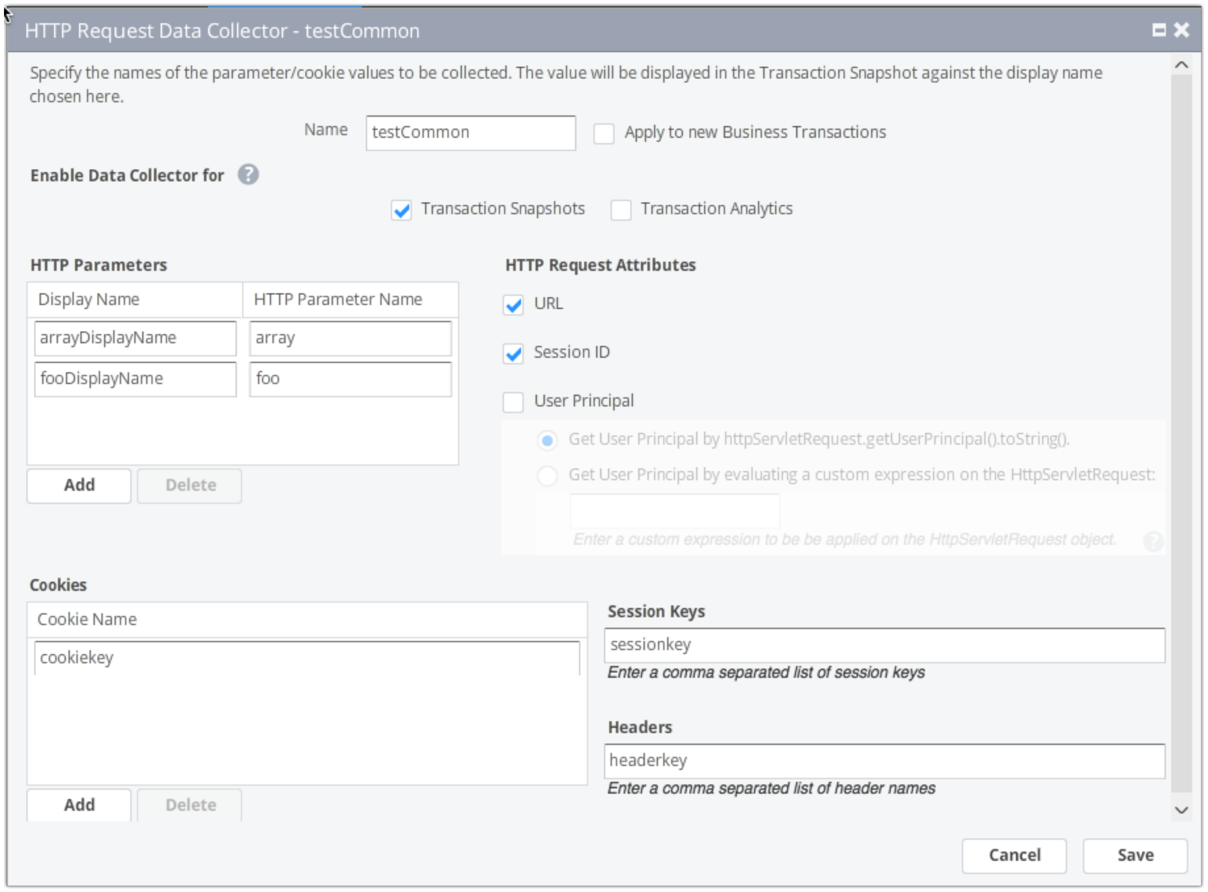
When complete, transaction snapshots for slow, very slow, and stalled transactions, the transaction snapshots will include user data that shows the user name and cart total amount for GOLD customers.

PHP HTTP Data Collector Example

This example shows how the PHP Agent can be configured to capture information from an HTTP request. Note that data collectors cannot be added to a standalone PHP function.

In the HTTP Data Collector configuration specify the request data that you want to display in the snapshot.

This configuration captures two HTTP parameters, the URL and session ID, the cookie name, the session keys and the headers:



HTTP Request Data Collector - testCommon

Specify the names of the parameter/cookie values to be collected. The value will be displayed in the Transaction Snapshot against the display name chosen here.

Name: testCommon ☐ Apply to new Business Transactions

Enable Data Collector for

☑ Transaction Snapshots ☐ Transaction Analytics

HTTP Parameters

Display Name	HTTP Parameter Name
arrayDisplayName	array
fooDisplayName	foo

HTTP Request Attributes

☑ URL

☑ Session ID

☐ User Principal

● Get User Principal by httpServletRequest.getUserPrincipal().toString().

○ Get User Principal by evaluating a custom expression on the HttpServletRequest:

Enter a custom expression to be be applied on the HttpServletRequest object.

Cookies

Cookie Name
cookiekey

Session Keys

sessionkey

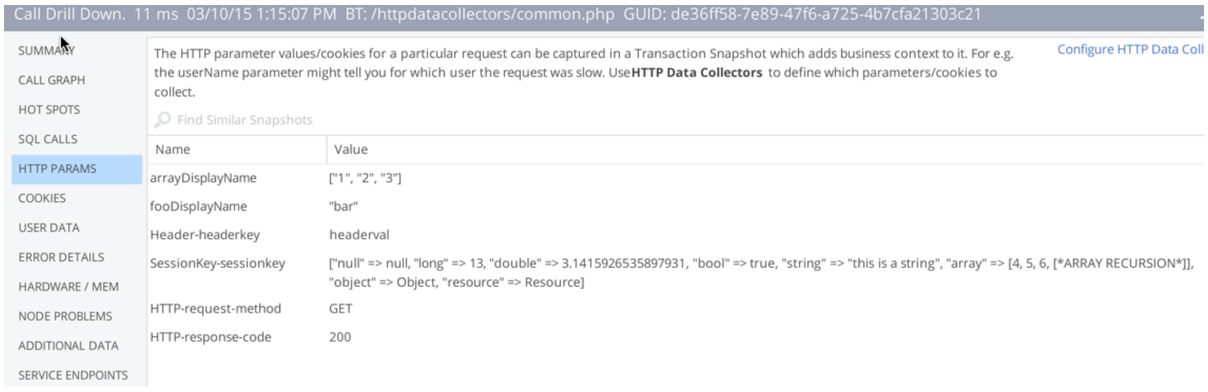
Enter a comma separated list of session keys

Headers

headerkey

Enter a comma separated list of header names

In the transaction snapshots, the URL and Session ID are displayed In the SUMMARY tab, and the configured HTTP parameters, session keys and headers in the HTTP PARAMS tab, along with the request method and response code:



Call Drill Down. 11 ms 03/10/15 1:15:07 PM BT: /httpdatacollectors/common.php GUID: de36ff58-7e89-47f6-a725-4b7cfa21303c21

The HTTP parameter values/cookies for a particular request can be captured in a Transaction Snapshot which adds business context to it. For e.g. the userName parameter might tell you for which user the request was slow. Use **HTTP Data Collectors** to define which parameters/cookies to collect.

Name	Value
arrayDisplayName	["1", "2", "3"]
fooDisplayName	"bar"
Header-headerkey	headerval
SessionKey-sessionkey	["null" => null, "long" => 13, "double" => 3.1415926535897931, "bool" => true, "string" => "this is a string", "array" => [4, 5, 6, [*ARRAY RECURSION*]], "object" => Object, "resource" => Resource]
HTTP-request-method	GET
HTTP-response-code	200

The cookie name and value are shown in the COOKIES tab.