AppDynamics App iQ Platform
AppDynamics Platform
Version 4.5.13
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic Jobs</td>
<td>369</td>
</tr>
<tr>
<td>Configure Synthetic Jobs</td>
<td>371</td>
</tr>
<tr>
<td>Synthetic Scripts</td>
<td>376</td>
</tr>
<tr>
<td>Write Your First Script</td>
<td>377</td>
</tr>
<tr>
<td>Locate DOM Elements</td>
<td>381</td>
</tr>
<tr>
<td>Wait for DOM Elements</td>
<td>382</td>
</tr>
<tr>
<td>Work with Screenshots</td>
<td>384</td>
</tr>
<tr>
<td>Add Logs to Troubleshoot</td>
<td>386</td>
</tr>
<tr>
<td>Import Python Screenshots</td>
<td>389</td>
</tr>
<tr>
<td>Verify Program Correctness</td>
<td>391</td>
</tr>
<tr>
<td>Select Client Certificates</td>
<td>393</td>
</tr>
<tr>
<td>Synthetic Agent Locations</td>
<td>395</td>
</tr>
<tr>
<td>Synthetic Credential Vault</td>
<td>399</td>
</tr>
<tr>
<td>Alerts for Browser Synthetic Monitoring</td>
<td>402</td>
</tr>
<tr>
<td>Synthetic Sessions</td>
<td>405</td>
</tr>
<tr>
<td>Synthetic Pages</td>
<td>412</td>
</tr>
<tr>
<td>Request Synthetic Snapshots On-Demand</td>
<td>414</td>
</tr>
<tr>
<td>Browser Synthetic Metrics</td>
<td>416</td>
</tr>
<tr>
<td>Browser Synthetic Licenses</td>
<td>421</td>
</tr>
<tr>
<td>Mobile Real User Monitoring</td>
<td>423</td>
</tr>
<tr>
<td>Hybrid Application Support</td>
<td>425</td>
</tr>
<tr>
<td>Set Up and Access Mobile RUM</td>
<td>428</td>
</tr>
<tr>
<td>Instrument iOS Applications</td>
<td>431</td>
</tr>
<tr>
<td>Install the iOS SDK</td>
<td>432</td>
</tr>
<tr>
<td>Instrument an iOS Application</td>
<td>435</td>
</tr>
<tr>
<td>Customize the iOS Instrumentation</td>
<td>446</td>
</tr>
<tr>
<td>Upload the dSYM File</td>
<td>481</td>
</tr>
<tr>
<td>Manage the dSYM Files with Bitcode Enabled</td>
<td>487</td>
</tr>
<tr>
<td>Troubleshoot the iOS Instrumentation</td>
<td>489</td>
</tr>
<tr>
<td>Instrument Android Applications</td>
<td>500</td>
</tr>
<tr>
<td>Instrument an Application with the Android Agent Installer</td>
<td>491</td>
</tr>
<tr>
<td>Instrument an Android Application Manually</td>
<td>494</td>
</tr>
<tr>
<td>Verify the Android Instrumentation</td>
<td>498</td>
</tr>
<tr>
<td>Customize the Android Build</td>
<td>500</td>
</tr>
<tr>
<td>Automatically Upload Mapping Files</td>
<td>501</td>
</tr>
<tr>
<td>Configure ProGuard to Prevent Obfuscation and Class Removal</td>
<td>504</td>
</tr>
<tr>
<td>Exclude Classes from Being Instrumented</td>
<td>505</td>
</tr>
<tr>
<td>Enable/Disable Instrumentation for Build Types</td>
<td>506</td>
</tr>
<tr>
<td>Enable/Disable Native Crash Reporting</td>
<td>509</td>
</tr>
<tr>
<td>Customize the Android Instrumentation</td>
<td>511</td>
</tr>
<tr>
<td>Manually Upload Mapping Files</td>
<td>533</td>
</tr>
<tr>
<td>Troubleshoot the Android Instrumentation</td>
<td>538</td>
</tr>
<tr>
<td>Instrument Xamarin Applications</td>
<td>540</td>
</tr>
<tr>
<td>Instrument a Xamarin Application</td>
<td>541</td>
</tr>
<tr>
<td>Customize the Xamarin Instrumentation</td>
<td>547</td>
</tr>
<tr>
<td>Instrument Cordova Applications</td>
<td>556</td>
</tr>
<tr>
<td>Customize the Cordova Instrumentation</td>
<td>561</td>
</tr>
<tr>
<td>Instrument React Native Applications</td>
<td>574</td>
</tr>
<tr>
<td>Customize the React Native Instrumentation</td>
<td>580</td>
</tr>
<tr>
<td>Configure the Mobile Agent Connected to the Controller</td>
<td>583</td>
</tr>
<tr>
<td>Configure the Controller UI for Mobile RUM</td>
<td>600</td>
</tr>
<tr>
<td>Configure Mobile Network Request Naming</td>
<td>601</td>
</tr>
<tr>
<td>Configure Mobile Network Request Thresholds</td>
<td>604</td>
</tr>
<tr>
<td>Configure Mobile Percentile Metrics</td>
<td>605</td>
</tr>
<tr>
<td>Configure Which Network Requests Are Sent to the Event Service</td>
<td>606</td>
</tr>
<tr>
<td>Configure Request IP Address Storage - Mobile</td>
<td>608</td>
</tr>
<tr>
<td>Configure Session Monitoring</td>
<td>610</td>
</tr>
<tr>
<td>Configure Mobile Crash Alerts</td>
<td>611</td>
</tr>
<tr>
<td>Configure Application Not Responding Thresholds</td>
<td>612</td>
</tr>
<tr>
<td>Correlate Business Transactions for Mobile RUM</td>
<td>613</td>
</tr>
<tr>
<td>Monitor Your Applications with Mobile RUM</td>
<td>616</td>
</tr>
<tr>
<td>Mobile App Dashboard</td>
<td>623</td>
</tr>
<tr>
<td>Mobile Sessions</td>
<td>625</td>
</tr>
<tr>
<td>Mobile Screenshots</td>
<td>631</td>
</tr>
<tr>
<td>Configure Mobile Screenshots</td>
<td>633</td>
</tr>
<tr>
<td>Take Mobile Screenshots</td>
<td>635</td>
</tr>
<tr>
<td>View Mobile Screenshots</td>
<td>638</td>
</tr>
<tr>
<td>Mobile Health Rules</td>
<td>641</td>
</tr>
<tr>
<td>Network Requests</td>
<td>646</td>
</tr>
<tr>
<td>Network Requests List</td>
<td>648</td>
</tr>
<tr>
<td>Network Request Analyze</td>
<td>650</td>
</tr>
<tr>
<td>Network Request Snapshots List</td>
<td>652</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Crashes</td>
<td>655</td>
</tr>
<tr>
<td>Crash Dashboard</td>
<td>656</td>
</tr>
<tr>
<td>Crash Analyze</td>
<td>658</td>
</tr>
<tr>
<td>Crash Snapshots</td>
<td>660</td>
</tr>
<tr>
<td>Crash Snapshot Properties</td>
<td>662</td>
</tr>
<tr>
<td>Get Human-Readable Crash Snapshots</td>
<td>663</td>
</tr>
<tr>
<td>Crash Alerts</td>
<td>665</td>
</tr>
<tr>
<td>Code Issues</td>
<td>667</td>
</tr>
<tr>
<td>Code Issues Dashboard</td>
<td>668</td>
</tr>
<tr>
<td>Code Issues Analyze</td>
<td>672</td>
</tr>
<tr>
<td>Code Issue Snapshots</td>
<td>676</td>
</tr>
<tr>
<td>Code Issue Alerts</td>
<td>679</td>
</tr>
<tr>
<td>Custom Data</td>
<td>680</td>
</tr>
<tr>
<td>Troubleshoot Mobile Applications</td>
<td>682</td>
</tr>
<tr>
<td>Troubleshoot Slow Network Requests from Mobile Applications</td>
<td>683</td>
</tr>
<tr>
<td>Troubleshoot Mobile Application Crashes</td>
<td>684</td>
</tr>
<tr>
<td>Get More Information about Mobile RUM</td>
<td>686</td>
</tr>
<tr>
<td>Mobile RUM Metrics</td>
<td>687</td>
</tr>
<tr>
<td>Mobile RUM Supported Environments</td>
<td>689</td>
</tr>
<tr>
<td>Mobile RUM Licenses</td>
<td>692</td>
</tr>
<tr>
<td>Network Request Limits</td>
<td>694</td>
</tr>
<tr>
<td>Mobile Agent Version and Deployment Support Matrix</td>
<td>696</td>
</tr>
<tr>
<td>Mobile Agent Feature Support</td>
<td>697</td>
</tr>
<tr>
<td>IoT Monitoring</td>
<td>700</td>
</tr>
<tr>
<td>Set Up and Access IoT Monitoring</td>
<td>707</td>
</tr>
<tr>
<td>Correlate Business Transactions for IoT Monitoring</td>
<td>709</td>
</tr>
<tr>
<td>Instrument Applications with the IoT C/C++ SDK</td>
<td>714</td>
</tr>
<tr>
<td>Instrument Applications with the IoT Java SDK</td>
<td>725</td>
</tr>
<tr>
<td>Instrument Applications with the IoT REST APIs</td>
<td>736</td>
</tr>
<tr>
<td>Confirm the IoT Application Reported Data to the Controller</td>
<td>745</td>
</tr>
<tr>
<td>Monitor Applications with the IoT Dashboards</td>
<td>746</td>
</tr>
<tr>
<td>Configure IoT Application Monitoring</td>
<td>755</td>
</tr>
<tr>
<td>IoT Monitoring Licenses</td>
<td>760</td>
</tr>
</tbody>
</table>
End User Monitoring

AppDynamics End User Monitoring (EUM) gives you end-to-end visibility on the performance of your web and mobile applications. EUM helps you troubleshoot problems such as slow web, Ajax, mobile network requests, or IoT application errors. EUM provides metrics on application performance and user activity, such as:

- How server performance impacts your web, mobile, and device performance
- How third-party APIs impact your web, mobile, and device performance
- Where your heaviest loads originate
- How your users connect to and navigate your application

### Installation
- Inject the JavaScript Agent
- Instrument iOS Applications
- Instrument Android Applications
- Instrument Applications with the IoT C/C++ SDK
- Instrument Applications with the IoT Java SDK

### Configuration
- Configure the Controller UI for Browser RUM
- Configure the Controller UI for Mobile RUM
- Configure IoT Application Monitoring
- Configure the JavaScript Agent

### Using EUM
- Browser Snapshots
- Pages & Ajax Requests View
- Synthetic Scripts
- Mobile App Dashboard
- Monitor Applications with the IoT Dashboards

### Extensibility and Reference
- Customize the iOS Instrumentation
- Customize the Android Instrumentation
- Browser RUM Metrics
- Browser Synthetic Metrics
- Mobile RUM Metrics
Overview of End User Monitoring

AppDynamics End User Monitoring (EUM) gives you visibility on the performance of your application from the viewpoint of the end user.

While Application Performance Monitoring (APM) measures user interaction starting at the web server or application server entry point, EUM extends that visibility all the way to the web browser, mobile, or IoT application. As a result, EUM reveals the impact the network and browser rendering time have on the user experience of your application.

The diagram below provides an overview of the different components, deployment models (SaaS/on-premises), and the Controller UI as seen by AppDynamics end users. The SaaS deployment employs services (Controller Service, Events Service, EUM Service, EUM Synthetic Monitoring Service) to collect, store, and process data, whereas, the on-premises deployment requires customers to install discrete components such as the Controller and servers (Events Server, EUM Server) that run processes to collect, store, and process data.

Annotations are provided for each label below the diagram.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Component(s)</th>
</tr>
</thead>
</table>

Copyright © AppDynamics 2012-2019
|   | The Browser, Mobile, and IoT Agents run in web/mobile/IoT applications, collect metrics, and then transmit that data to either a SaaS or an on-premises deployment of AppDynamics. | Browser/Mobile/IoT Apps  
EUM Agents |
|---|---|---|
| 2 | The AppDynamics SaaS Cloud, consisting of the components listed to the right, stores, processes, and analyzes data, and then delivers RUM metrics to the Controller UI. | **Controller Service** - stores data and metadata, makes calls to the EUM Server for raw data and the Events Service for analytic data.  
**Events Service** - stores short-term RUM data (such as sessions, network requests, snapshots) for heavier analysis.  
**EUM Service** - verifies, aggregates, and packages raw browser/mobile app metrics.  
**EUM Synthetic Monitoring Service** - schedules and executes Browser Synthetic jobs and returns session data to the Controller. |
| 3 | The on-premises deployment of AppDynamics has most of the same components and data as the SaaS model. In this model, DevOps install and administer their own Controller, Events Service, and EUM Server. The EUM Synthetic Monitoring Service and sessions data, however, are not available in the on-premises deployment. | **Controller**  
**Events Server**  
**EUM Server**  
See the descriptions for each component above in step 2. |
| 4 | AppDynamics users can go to the AppDynamics Controller UI to view and analyze RUM metrics as snapshots, pages, Ajax requests, sessions, network requests, or in the form of charts and graphs. | Controller UI  
See the description for the Controller UI in step 2. |

**Understand End User Activity**

Using EUM, you can determine:

- Where geographically your heaviest application load is originated.
- Where geographically your slowest End User response times occur.
- How performance varies by
  - location.  
  - client type, device, browser and browser version, and network connection for web requests.  
  - application and application version, operating system version, device, and carrier for mobile requests.  
- What your slowest web/Ajax requests are, and where the problem may lie.  
- What your slowest mobile and IoT network requests are, and where the problem may lie.  
- How application server performance impacts the performance of your web and mobile traffic.  
- Whether your mobile or IoT applications are experiencing errors or crashes and the root cause of the issues. For example, for mobile applications, EUM provides stack traces and event trails for the crash or error, helping you troubleshoot and optimize mobile applications.

**View EUM Data**

The performance information generated by EUM is distinct from the application monitoring data generated by app server agents.
EUM data appears in various locations in the Controller UI, including in the User Experience dashboard, Metric Browser, and AppDynamics Analytics pages.

When linked to application business transactions, EUM data gives you a complete view of your end users’ experience from the client request, through the application environment, and back to the client as the user response.

You can view EUM performance data in the Controller UI in the User Experience tab. From there, you can access information specific to browser applications, mobile applications, or connected devices (IoT applications).

**On-Premises EUM Deployments**

By default, EUM is configured to use an AppDynamics-hosted component called the EUM Cloud. For a fully on-premises installation, the EUM Server provides the functionality of the EUM Cloud. For information, see EUM Server Deployment.

Some functionality for EUM depends on the AppDynamics Platform Events Service. In a SaaS environment, this is managed by AppDynamics, but it is also possible to use this functionality in an on-premises form.

To host the Events Service on premises, see:

- Custom Install
- Events Service Deployment

If you are adding EUM to an existing on-premises Controller installation, you should evaluate your current configuration’s ability to handle the additional load imposed by EUM. For more information, see Additional Sizing Considerations.

**Access the SaaS EUM Server**

The SaaS EUM Server consists of the components listed below. Each component may have different endpoints depending on the region of your Controller.

- EUM Services - The Mobile Agents, JavaScript Agent, and IoT SDKs send data to the EUM Services. The Controller fetches data from the EUM Server.
- Events Service - The EUM Server sends analytics data to the Events Service. The Controller also queries the Events Service.
- Synthetic Services - The Synthetic Private Agent and Synthetic Hosted Agent send data to the Synthetic Services.

If your SaaS or on-prem deployment requires access to any of these components on the Internet, make sure the URLs given in SaaS Domains and IP Ranges are accessible from your network.

For on-prem deployments, however, the EUM Server can either be located on the Internet or hosted inside your own data center/network. On-premises access points are configured at installation or through the UI. See EUM Server Deployment and Events Service Deployment for more information.

**How EUM Works with other AppDynamics Products**

This section describes how other App iQ Platform products work with EUM to provide complete, full visibility on application health and user experience.

**EUM and Application Performance Monitoring**

Using APM with EUM provides you with greater insight into how the performance of your business application affects the end-user experience. To integrate APM with EUM, you correlate business transactions with browser snapshots. This enables you to trace bad user experiences to issues with your backends such as an unresponsive web service, bad database query or slow server response. To learn how to integrate APM with EUM, see Correlate Business Transactions for Browser RUM.

You can also use the server app agents running on business applications that serve your browser applications to inject JavaScript agent into the code that runs on the browser. This obviates the need to manually inject the JavaScript agent. For more information, see Automatic Injection and Assisted Injection.
EUM and Application Analytics

AppDynamics Application Analytics enables you to use the powerful AppDynamics Query Language (ADQL) to analyze different types of EUM data through complex queries. The Analytics components are based on the Events Service, which is also the source of data for Browser Analyze, Crash Analyze, Network Requests Analyze, and all IoT data. Analytics requires a license separate from the EUM licenses except for IoT Monitoring.

Learn More

For more information on the type of user monitoring you interested in, see:

- Browser Real User Monitoring
- Browser Synthetic Monitoring
- Mobile Real User Monitoring
- IoT Monitoring
EUM Data

On this page:
- Types of EUM Data
- How Data Is Stored and Retrieved
- Data Storage Details
- Controller Mapping of Data
- How to Access EUM Data
- Data and Deployment Models

Related pages:
- Using the Controller APIs

EUM collects, aggregates, and stores many different types of data that are displayed in different UI components of the Controller.

This page helps you understand the following:
- the types of EUM data
- how data is collected
- where data is stored
- where data is displayed in the Controller

Types of EUM Data

Metrics

Metrics are data that reflect your application's performance. Browser RUM captures metrics using your end-users' web browsers such as timing and Ajax metrics. Mobile RUM captures metrics from mobile applications, such as crash metrics and network request metrics. You can view and analyze EUM metrics in the Metric Browser.

To learn more about EUM metrics, see the following:
- Browser RUM Metrics
- Browser Synthetic Metrics
- Mobile RUM Metrics
- Extensions and Custom Metrics

Custom Data

Browser RUM

You can add user information that is specific to your application to a browser snapshot. The information is expressed as key-value pairs. The information appears in the User Data section of the snapshot and is available for page snapshots, Ajax requests, and virtual pages. To learn how to set custom user data, see Add Custom User Data to a Page Browser Snapshot.

Mobile RUM

Mobile RUM custom data such as Info Points, Custom Timers, and Custom Metrics are considered metrics and not records stored as mobile request event data. User data is another type of Mobile RUM custom data. Modifying user data affects the current state but does not generate event records. Changes to user data can be reported, however, through network request events because user data is attached to each network request.

To learn how to set user data, see User Data (iOS SDK) or User Data (Android SDK).

EUM Analytics Data
You can view EUM data in Application Analytics if you have an Application Analytics license separate from the EUM licenses. The EUM data found in Application Analytics consists of event data and is stored in the Events Service. The EUM Analytics data visualized in widgets is also from the Events Service.

EUM Analytics provides data for the following event types:

- Browser Requests Event Data
- Browser Sessions Event Data
- Mobile Requests Event Data
- Mobile Sessions Event Data
- Mobile Crash Report Event Data
- Synthetic Sessions
- Connected Device Data

For details about the limits and retention policy for EUM Analytics data, see the entries for Browser Analytics (SaaS), Browser Analytics (on-prem), Mobile Analytics (SaaS), and Mobile Analytics (on-prem) given on the License Entitlements and Restrictions page.

Cookies

Browser RUM uses two different kinds of short-lived cookies to help it collect data and correlate events. Neither type of cookie contains any personally identifiable information (PII) and are immediately deleted after being read.

- **The ADRUM cookie**: written by the JavaScript Agent, this cookie contains the referral page URL and some timing information to assist gathering First Byte Time for some browser types. When the agent loads on the subsequent page, it reads the information and then deletes the cookie. If there is no agent on that page, the cookie is deleted when the browser is closed. For privacy purposes, the URL of the referral page is hashed.
- **The ADRUM_BT cookies**: written by the server-side agent when the page is served from an instrumented server. These cookies help correlate browser data with related server-side performance data.
  
  - ADRUM_BTa - contains the backend transaction ID as well as timing info and is used to correlates end-user experience with the health of the backend app.
  - ADRUM_BTg - contains the backend transaction ID and is used as an alternative method to correlate end-user experience with the health of the backend app.
  - ADRUM_BT[1-5] - contains the business transaction numbers as well as timing and error info for the first five business transactions, such as ADRUM_BT1, ADRUM_BT2, etc.
  - ADRUM_BTs - contains a link from a browser snapshot to a server snapshot.
  - ADRUM_BTh - only written if there was a server-side error.

If Browser RUM detects that the page is HTTPS, the Security attribute is set for cookies. The Security attribute is a flag that forbids a cookie from being transmitted via an unencrypted HTTP connection.

Web Storage

Browser RUM stores key-value pairs in web storage to associate page views with a particular session and browser. The value for each key is a randomly generated ID.

The following lists the keys and the expiration time for each key-value pair:

- ADRUM_AGENT_INFO (never)
- ADRUM_CLIENTINFO (never)
- ADRUM_XD_AGENT_ID (never)
- ADRUM_XD_AGENT_INFO (1 week)

Mobile Local Storage

Mobile Agents use beacons to transmit metrics, metadata about the app, network requests, crashes, and custom data. When a beacon cannot be transmitted, the data is persisted in permanent storage within the container of the application and subject to the security configuration of the device and application. No encryption is currently being utilized. Once the network connection is restored, the beacons resume transmitting data. Because some of the data is provided by the developer's instrumentation of the app, such as breadcrumbs, user info, and the app (URLs, crash reports), some information is not explicitly collected by AppDynamics.

Mobile RUM also locally stores a randomly generated ID for tracking sessions and license usage. The ID is stored in the Events Service.
How Data Is Stored and Retrieved

Browser RUM

Browser RUM data is stored in the Controller, the Events Service, and the EUM Server. The JavaScript Agent sends raw data to the EUM Server, where the data is verified, aggregated, and packaged every minute. The EUM Server then sends the data to the Controller and the Events Service. The Controller UI makes requests to the Controller API, which fetches data from one of the three data stores (EUM Server, Events Service, Controller). Because the Controller UI only interacts with the Controller API, it doesn't need to know where the data is stored.

Mobile RUM

Mobile RUM data is stored in the Controller, the Events Service, and the EUM Server. The Mobile Agents send raw data to the EUM Server, where the data is verified, aggregated, and packaged every minute. The EUM Server then sends the data to the Controller and the Events Service. The Controller UI makes requests to the Controller API, which fetches data from one of the three data stores (EUM Server, Events Service, Controller). Because the Controller UI only interacts with the Controller API, the Controller UI doesn't need to know where the data is stored.
Data Storage Details

Browser RUM

The table below shows where different Browser RUM data is stored and how long the data is retained. Resource details are only available for those browser snapshots with resource timing.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Controller</th>
<th>Events Service</th>
<th>EUM Server (SaaS/On-Premises)</th>
<th>Retention Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browser Metrics</td>
<td>1 year</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Browser Snapshots</td>
<td>2 weeks</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resource Details</td>
<td>2 weeks</td>
<td>(filesystem)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EUM Page Configuration</td>
<td>N/A</td>
<td>(filesystem)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page View Events</td>
<td>8 days</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ajax Events</td>
<td>8 days</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Session Events</td>
<td>8 days</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metadata</td>
<td></td>
<td>(MySQL)</td>
<td></td>
<td>Indefinitely</td>
</tr>
<tr>
<td>Licenses (SaaS)</td>
<td></td>
<td>(MySQL)</td>
<td></td>
<td>Indefinitely</td>
</tr>
<tr>
<td>Licenses (On-Prem)</td>
<td></td>
<td>(MySQL)</td>
<td></td>
<td>Indefinitely</td>
</tr>
</tbody>
</table>
Browser RUM Default Data Limits per App Key

- Metrics: 100k
- Pages: 500
- Ajax Requests: 500
- Max Event Size: 1 MB

Mobile RUM

The table below shows where different EUM data is stored and how long the data is retained.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Controller</th>
<th>Events Service</th>
<th>EUM Server (SaaS/On-Premises)</th>
<th>Retention Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile RUM Metrics</td>
<td>Controller</td>
<td>Events Service</td>
<td>EUM Server (SaaS/On-Premises)</td>
<td>1 year</td>
</tr>
<tr>
<td>Network Request Snapshots</td>
<td>Controller</td>
<td>Events Service</td>
<td>EUM Server (SaaS/On-Premises)</td>
<td>2 weeks</td>
</tr>
<tr>
<td>Custom Data</td>
<td>Controller</td>
<td>Events Service</td>
<td>EUM Server (SaaS/On-Premises)</td>
<td>1 year</td>
</tr>
<tr>
<td>Crash Analyze</td>
<td>Controller</td>
<td>Events Service</td>
<td>EUM Server (SaaS/On-Premises)</td>
<td>1 year</td>
</tr>
<tr>
<td>Events</td>
<td>Controller</td>
<td>Events Service</td>
<td>EUM Server (SaaS/On-Premises)</td>
<td>2 weeks</td>
</tr>
<tr>
<td>Session Events</td>
<td>Controller</td>
<td>Events Service</td>
<td>EUM Server (SaaS/On-Premises)</td>
<td>8 days</td>
</tr>
<tr>
<td>Metadata</td>
<td>Controller</td>
<td>Events Service</td>
<td>EUM Server (SaaS/On-Premises)</td>
<td>Indefinitely</td>
</tr>
<tr>
<td>ProGuard/dSYM Files</td>
<td>Controller</td>
<td>Events Service</td>
<td>EUM Server (SaaS/On-Premises)</td>
<td>Indefinitely</td>
</tr>
<tr>
<td>Screenshot Files</td>
<td>Controller</td>
<td>Events Service</td>
<td>EUM Server (SaaS/On-Premises)</td>
<td>8 days</td>
</tr>
<tr>
<td>Licenses (SaaS)</td>
<td>Controller</td>
<td>Events Service</td>
<td>EUM Server (SaaS/On-Premises)</td>
<td>Indefinitely</td>
</tr>
<tr>
<td>Licenses (On-Prem)</td>
<td>Controller</td>
<td>Events Service</td>
<td>EUM Server (SaaS/On-Premises)</td>
<td>Indefinitely</td>
</tr>
</tbody>
</table>

Mobile RUM Default Data Limits per Mobile App and App Key

<table>
<thead>
<tr>
<th>Unit</th>
<th>Metric Limit</th>
<th>Network Requests</th>
<th>Max Event Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile App</td>
<td>N/A</td>
<td>500</td>
<td>1 MB</td>
</tr>
<tr>
<td>App Key</td>
<td>100,000</td>
<td>2000</td>
<td></td>
</tr>
</tbody>
</table>

Controller Mapping of Data

Browser RUM

The table below shows the relationship between the Controller UI components and their data sources.

<table>
<thead>
<tr>
<th>Controller Component</th>
<th>Storage Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>Controller</td>
</tr>
<tr>
<td>Geo Dashboard</td>
<td>Controller</td>
</tr>
<tr>
<td>Browser Snapshots</td>
<td>Controller (no resource details) / EUM Server (SaaS/On-Premises) (resource details)</td>
</tr>
<tr>
<td>Usage Stats</td>
<td>Controller</td>
</tr>
<tr>
<td>Sessions</td>
<td>EUM Server (SaaS/On-Premises), Events Service</td>
</tr>
</tbody>
</table>
## Mobile RUM

The table below shows the relationship between the Controller UI components and their data sources.

<table>
<thead>
<tr>
<th>Controller Component</th>
<th>Storage Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>Controller</td>
</tr>
<tr>
<td>Geo Dashboard</td>
<td>Controller</td>
</tr>
<tr>
<td>Usage Stats</td>
<td>Controller</td>
</tr>
<tr>
<td>Sessions</td>
<td>Events Service</td>
</tr>
<tr>
<td>Network Requests</td>
<td>Controller, Events Service (limited)</td>
</tr>
<tr>
<td>Network Requests Snapshots</td>
<td>Controller</td>
</tr>
<tr>
<td>Crashes</td>
<td>Events Service, EUM Server (SaaS/On-Premises)</td>
</tr>
<tr>
<td>Crashes Analyze</td>
<td>Events Service</td>
</tr>
<tr>
<td>Custom Data</td>
<td>Controller</td>
</tr>
<tr>
<td>Events</td>
<td>Controller</td>
</tr>
</tbody>
</table>

## How to Access EUM Data

In addition to accessing EUM data through the Controller UI, you can also access EUM event data through Application Analytics and the AppDynamics APIs. The AppDynamics APIs include the Analytics Events API and the Metric and Snapshot API that you can use to access EUM metric data.

If you have enabled Application Analytics and Browser RUM, you can use **Browser Analytics** to view data for the following event types:

- Browser Requests
- Browser Sessions

If you have enabled Application Analytics and Browser Synthetic Monitoring, you can use **Synthetic Analytics** to view data for the following event types:

- Synthetic Sessions

If you have enabled Application Analytics and Mobile RUM, you can use **Mobile Analytics** to view data for the following event types:

- Mobile Requests
- Mobile Sessions
- Mobile Crash Reports

If you have enabled IoT Monitoring, you can use **IoT Analytics** to view data for the event types below. You don't need to enable Application Analytics to use IoT Analytics:

- Connected Device Data

## Data and Deployment Models

AppDynamics offers the two deployment models SaaS and on premises. You can access most Browser and Mobile RUM data from either deployment model.
EUM Accounts, Licenses, and App Keys

On this page:
- View EUM License Information
- License Key
- EUM App Key
- Changes to the EUM Account Name or License Key

Licensing for EUM is separate from Controller licensing. EUM Accounts can be enabled for three distinct license types:

- Browser RUM
- Mobile RUM
- Browser Synthetic
- Connected Devices

If you have at least one of these license types, you have an EUM Account, which has a name and a license key associated with it.

View EUM License Information

1. Click the gear icon at the top right of the Controller UI.
2. Click License.
3. Scroll down to the User Experience section.

License Key

This is the unique identifier that AppDynamics uses to associate end-user data, Browser RUM, Browser Synthetic, Mobile RUM, and Connected Devices to your account. From a practical perspective, you only need to know this information for troubleshooting purposes. The same key applies to all three services. However, each product has its own types and metrics for allowed usage.

EUM App Key

This is the unique identifier that AppDynamics uses to associate end-user data to specific EUM applications. Each EUM application will be associated with one EUM App Key. For your applications to be monitored, they will need to include the EUM App Key when reporting data. The EUM App Keys are also associated with an EUM account name and license key.

Changes to the EUM Account Name or License Key

You must have separate licenses for the Synthetic Hosted Agent and the Synthetic Private Agent.
Changing your EUM account name or license key will affect existing EUM App Keys and the EUM applications using those EUM App Keys.

When the EUM Server detects a new EUM account name or license key, it regenerates new EUM App Keys. This is often a concern when renewing an EUM license, when you may try to make changes to the EUM account name or license key.
Browser Monitoring

On this page:
- Overview of the Controller UI for Browser Monitoring
- Differences between Browser RUM and Browser Synthetic Monitoring
- How to use Browser RUM and Browser Synthetic Monitoring Together

AppDynamics Browser Monitoring provides two products that can help you improve your user's experience: Browser Real User Monitoring (Browser RUM) and Browser Synthetic Monitoring. Browser RUM allows you to see how your web application is performing, using real user data to analyze application performance and user experience. Browser Synthetic Monitoring allows you to analyze application availability and performance, using scheduled testing to analyze website availability.

This page provides an overview of Browser RUM and Browser Synthetic Monitoring, the differences between the two products, and how to use them together to monitor your browser applications.

Overview of the Controller UI for Browser Monitoring

Browser RUM and Browser Synthetic Monitoring share two dashboards: Browser App Dashboard and Resource Performance Dashboard. Browser RUM and Browser Synthetic Monitoring also have separate UI sections: Real Users and Synthetic.

Browser App Dashboard

The Browser App Dashboard provides a high-level understanding of how your application's overall performance. When you first navigate to a browser application, you are defaulted to the Browser App Dashboard > Overview tab. The Overview tab contains widgets for both Browser RUM and Browser Synthetic Monitoring data.

Resource Performance Dashboard

The Resource Performance Dashboard provides a high-level understanding of how your resources affect the performance of your browser application. You can use this dashboard to pinpoint resource-related performance issues affecting the user experience. You can toggle between Browser RUM and Browser Synthetic Monitoring data.

Real Users

Browser RUM has three tabs for more detailed real-user data analysis:
Browser Synthetic Monitoring has three tabs for more detailed synthetic user data analysis:

- Synthetic Jobs
- Synthetic Sessions
- Synthetic Pages
- Synthetic On-Demand

Differences between Browser RUM and Browser Synthetic Monitoring

Although Browser Synthetic Monitoring and Browser RUM report similar metrics, you should be wary of comparing them because of these differences:

- hardware
- network connections
- different browsers
- browser caching is not present in synthetic sessions

If you see a sudden change in any of those metrics, however, you should compare the results of Browser Synthetic Monitoring and Browser RUM to see if there is an existing problem.

Performance Versus Workflows

Browser RUM excels at capturing the full breadth of performance that your real users experience. Browser Synthetic Monitoring gives you confidence that your key workflows are always working and performing.

Synthetic Metrics and Screenshots

Browser Synthetic Monitoring can collect certain data that Browser RUM cannot. For example, Browser Synthetic Monitoring provides screenshots, which help you see what is going on. You can also use the Visually Complete metric and other related metrics to understand how users experience page load time.

How to use Browser RUM and Browser Synthetic Monitoring Together

Identify Issues

Although Browser RUM can detect certain kinds of problems (like JavaScript exceptions), it cannot comprehensively test for functional correctness. For example, you may want to verify that your online store has reasonable prices a list of items. If your site is down entirely, then the JavaScript Agent will never be loaded, so errors or verifications will not be reported. Fortunately, Browser Synthetic Monitoring will keep running, discover the error, alert you, and provide detailed information about the problem.

Control Environmental Factors

Hardware, browsers, operating systems, and networks complicate performance analysis in Browser RUM. Browser Synthetic Monitoring uses consistent hardware, software, and network configurations, so if you see deviations in performance, you can be fairly certain a problem exists.
Browser App Dashboard

On this page:
- Access the Browser App Dashboard View
- Overview
- Geo Dashboard
- Browser Snapshots
- Usage Stats

The Browser App Dashboard gives you a high-level overview of how your application is performing, as described on this page.

Access the Browser App Dashboard View

1. Open the browser application in which you are interested.
2. On the left navigation bar, select Browser App Dashboard.
3. Click the tab for the view you want to access.

Overview

The Browser App Overview displays a set of configurable widgets, showing multiple graphs and lists featuring common high-level indicators of application performance. You can delete widgets using the X in the upper right corner and re-add them using the add widgets icon.

<table>
<thead>
<tr>
<th>Widgets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Page Requests</td>
</tr>
<tr>
<td>End User Response Time</td>
</tr>
<tr>
<td>Browsers</td>
</tr>
<tr>
<td>Devices</td>
</tr>
<tr>
<td>End User Response Time Distribution</td>
</tr>
<tr>
<td>Top 5 Pages by Total Requests</td>
</tr>
<tr>
<td>End User Response Time Trend</td>
</tr>
<tr>
<td>Page Requests per Minute</td>
</tr>
<tr>
<td>Synthetic Availability Trend</td>
</tr>
<tr>
<td>Top 5 Countries by Total Page Requests</td>
</tr>
<tr>
<td>Reset Dashboard</td>
</tr>
</tbody>
</table>

Revert the widgets on this dashboard to their default positions.
Geo Dashboard

The Browser App Geo Dashboard displays key performance metrics by geographic location, based on page loads.

The dashboard is divided into three panels:

- A main panel in the upper left displaying geographic distribution of end users and the status of their requests on a map, if you selected the map view icon, or on a grid if you clicked the grid view icon.
- A panel on the right displaying key timings.
- A lower panel with dynamics trend graphs of KPIs.

The metrics displayed throughout the dashboard are for the region currently selected on the map or in the grid. For example, if you zoom down from world view to France on the map, the summary panels and the trend graphs display data for France.

If you are using Synthetic with this application, you can choose to see either RUM or Synthetic data in the Geo Dashboard. See Browser RUM Metrics for definitions of the metrics.

Map View Labels

The map view displays load circles with labels for countries that are in the key timing metrics given in the right panel. Some countries and regions, however, are only displayed in the grid view.

Regions and Countries

Regions are subdivisions of a country, such as a state, province, or city. In the default map view, key performance metrics are displayed by country. You can check Show regions in Global View check box to view the key performance metrics by region. The grid view, like the map view, by default displays key performance metrics by country, but can also be configured to show the metrics by region. Because the map view displays fewer regions than the grid view, if you do not see a region displayed in the map view, switch to the grid view.

See Browser RUM Countries and Regions by Geo Dashboard for a list of the countries and regions available in the map and grid views.

Unknown Locations

An unknown location is one for which the agent cannot determine the country from which the request originated.

You may also see metrics reported for a location named “Anonymous Proxy”. The data for Anonymous Proxy represents the aggregated metrics from one or more private IP addresses that the agent cannot identify.

One of the effects of Unknown regions is that it possible for a country to display as slow (red circles) on the global map, but when you drill down to the country all its regions appear normal (green circles). Or a country may display as normal on the global map, but some subregions may display as slow when you drill down.

Browser Snapshots

Browser snapshots capture and display a broad set of metrics associated with a single request. You can drill down into errors, and, if your app server is instrumented with server-side app agents, see any server-side transaction snapshots associated with that request. For more detailed information, see Browser Snapshots.

Usage Stats

The Browser App Usage Stats presents aggregated page-load usage data based on your users' browser type and device/platform.

The Browsers dashboard helps you discover:

- The slowest browsers in terms of total end-user response time.
- The slowest browsers to render the response page.
- The browsers that most of your end users use.
- The browsers that most of your end users use in a particular country or region.

The Devices dashboard shows mobile access only via browsers on the device. The dashboard helps you discover:
• The slowest devices in terms of total end-user response time
• The slowest devices to connect to the server
• The devices that most of your end users use
• The devices that most of your end users use in a particular country or region
Browser Snapshots

On this page:

- Browser Snapshots Tab
- Open the Snapshot
- Browser Snapshot Types

Related pages:

- Transaction Snapshots
- Configure Browser Snapshot Collection
- Configure Browser Monitoring Snapshot Thresholds
- Add Custom User Data to a Page Browser Snapshot

When Browser RUM is enabled, the JavaScript Agent collects browser snapshots for:

- Every base page (including virtual pages), iframe, and Ajax request; these serve as a heartbeat snapshot
- The slowest page for every region, every device, and every browser
- Unique JavaScript errors; identified by script name and line number
- Unique Ajax errors; identified by the HTTP error code in the Ajax response

For more information about browser snapshot collection, see Configure Browser Snapshot Collection.

Browser Snapshots Tab

A list of available browser snapshots appears. You can change the time frame of your search by changing the Time Frame dropdown.

List Key

- Normal user experience
- Slow user experience
- Very slow user experience
- JavaScript errors. To see more information on the error, hover over the icon and a popup with summary information appears.
- Correlated server-side transaction snapshot exists
- Snapshot includes resource timing information

Open the Snapshot

To open a snapshot, do one of the following:

- Double-click the snapshot that you want to examine
- Select the snapshot that you want to examine and click Details.

Browser Snapshot Types

There are three browser snapshot types, depending on whether the original object was a page, an iframe, or an Ajax request.

- Page Browser Snapshots
- Ajax Request Browser Snapshots
- Iframe Browser Snapshots
Page Browser Snapshots

On this page:
- Waterfall Graph
- Correlated Transaction Snapshots and Business Transactions
- JavaScript Errors
- User Data
- Unknown Metrics in Browser Snapshots

Related pages:
- Page Browser Snapshots - Resource Details
- Ajax Request Browser Snapshots
- Iframe Browser Snapshots

Page-based snapshots give you a detailed look at individual page requested. The Summary tab covers the general characteristics of the page. For information on the Resource Details tab, see Page Browser Snapshots - Resource Details.

Waterfall Graph

The top of the page snapshot Summary tab displays a waterfall graph of the overall transaction timing for the page. You can hover over each of the metrics to see a popup definition for that metric.

For more information on what these metrics mean, see Browser RUM Metrics. Additional details associated with the snapshot, including any Related Snapshots and any custom User Data, are displayed as a table below the graph.

Generate a Synthetic Snapshot - SaaS Only

Synthetic snapshots use geographically distributed agents to create snapshots of web page download performance without the idiosyncratic and potentially skewing effects of real user last-mile performance. You can use this button to create an on-demand synthetic snapshot to compare with a problematic RUM snapshot.

Correlated Transaction Snapshots and Business Transactions

If server-side correlation has been set up, a link to any related business transaction is shown:
Clicking the link takes you to the flow map for that business transaction on the server side.

If the business transaction created a transaction snapshot, links to the business transaction, tier, and node flow maps are displayed, along with a direct link to the transaction snapshot flow map. Cross-app correlation is supported.

In this case, the transaction snapshot also has a link back to the browser snapshot.

JavaScript Errors

If there are JavaScript errors on the page, they are displayed below the waterfall graph:

You can configure errors to ignore if you are seeing too many errors that are not of interest. See Configure JavaScript and Ajax Error
Detection.

When an error occurs in a script that is hosted on a domain other than the domain of the current page, most browsers prevent the JavaScript Agent from recording any details of the error. In such cases, the string CROSSDOMAIN is displayed as the Script Origin.

User Data

If you have set up custom user data, it appears in the Details section:

<table>
<thead>
<tr>
<th>User Data</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>algo&lt;String&gt;</td>
<td>LOG(N)</td>
</tr>
<tr>
<td>Random Number&lt;String&gt;</td>
<td>1938299132</td>
</tr>
<tr>
<td>hotel&lt;String&gt;</td>
<td>AppDynamics Resort</td>
</tr>
</tbody>
</table>

Unknown Metrics in Browser Snapshots

Browser RUM captures metrics using your end-users' web browsers. Occasionally, you may see Unknown data reported for one or metrics in a browser snapshot. This occurs on older or less sophisticated browsers that do not support the collection of a given metric.

See Browser Monitoring Metrics for details about which metrics may not be captured based on browser capabilities.
Page Browser Snapshots - Resource Details

On this page:
- Overview Panel
- Resource Waterfall

Related pages:
- Ajax Request Browser Snapshots
- Iframe Browser Snapshots

Page-based snapshots give you a detailed look at an individual page request. If the browser that was used supports the Resource Timing API, the Resource Details tab gives you a detailed breakdown of the performance of resources—scripts, CSS files, fonts, SVGs, and images—as they are loaded into the page. For information on the Summary tab, see Page Browser Snapshots.

Overview Panel

The Overview panel gives you a quick summary of the number of each type of resource being loaded and the domains from which they have been requested.

This example is based on a request by a real user, not a synthetic agent.

Resource Waterfall

The Resource Waterfall provides a snapshot of when, relatively, each resource was loaded in the page load process (before First Byte Time, before DOM Ready Time, before Onload), including the name of the resource, the domain from which it was fetched, the type of resource, and how long it took. You can filter the items by type using the upper-right dropdown and search using the search box.

To see more details, click any of the blue timelines. These are from the primary domain or cross-domains that have set the Timing-Allow-Origin HTTP header. See Resource Timing Metrics for descriptions and explanations about how the metrics are calculated.
Grey timelines indicate cross-domains that have not set the Time-Allow-Origin HTTP header. Working with your CDN provider to add this header can mean you get better information on shared and CDN-served content.
Ajax Request Browser Snapshots

On this page:
- Waterfall Graph
- Details

Related pages:
- Page Browser Snapshots
- Iframe Browser Snapshots

Ajax request browser-based snapshots give you a detailed look at an individual Ajax request. The Summary tab covers the general characteristics of the page.

**Waterfall Graph**

The top of the Ajax snapshot displays a waterfall graph of the overall transaction timing for the Ajax request. The snapshot is labeled either a Real User Snapshot or a Synthetic Snapshot based on the origin of the request. You can hover over each of the metrics to see a popup definition for that metric.

For a detailed description of what these metrics mean, see Browser RUM Metrics.

**Details**

Much of the information is the same as you see for page-based browser snapshots. The main differences are:

- If there is an Ajax error, the error code returned with it is listed.
- The parent page from which the Ajax call originates is listed.

You can configure errors to ignore if you are seeing too many errors that are not of interest. See Configure JavaScript and Ajax Error Detection.
Iframe Browser Snapshots

Related pages:
- Page Browser Snapshots
- Ajax Request Browser Snapshots

Iframe-based browser snapshots give you a detailed look at an individual iframe request. The display is identical to Page Browser Snapshots except in the Details section, where the parent page may also be displayed.

<table>
<thead>
<tr>
<th>Details</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Experience</td>
<td>Normal</td>
</tr>
<tr>
<td>Time</td>
<td>12/15/16 2:58:16 PM</td>
</tr>
<tr>
<td>End User Response Time (ms)</td>
<td>552 ms</td>
</tr>
<tr>
<td>Application Server Time</td>
<td>0 ms</td>
</tr>
<tr>
<td>Name</td>
<td><code>www.myapp.com/visual.html</code></td>
</tr>
<tr>
<td>URL</td>
<td><code>https://www.myapp.com/visual.html</code></td>
</tr>
<tr>
<td>Page Title</td>
<td><code>visual.html</code></td>
</tr>
<tr>
<td>Parent Page</td>
<td><code>www.myapp.com/visual.html</code></td>
</tr>
<tr>
<td>IP Address</td>
<td>10.134.1.162</td>
</tr>
<tr>
<td>Page Referrer</td>
<td><code>/referrer/visual.html</code></td>
</tr>
</tbody>
</table>
The Resource Performance Dashboard gives you a high-level overview of how your resources affect the performance of your browser application. You can use this dashboard to pinpoint resource-related performance issues affecting the user experience, such as the following:

- A prioritized list of resource performance issues by comparing their performance against thresholds
- Changes in the number of resources
- Large resources (images, JavaScript, CSS, etc.)
- Size increase of resources impacting performance. For example, a page banner might be replaced with an uncompressed image, slowing down the page load.
- Slow CDNs
- Resources that haven't been compressed
- Comparison of real user and synthetic resource performance

Once you navigate to a browser application, the Resource Performance Dashboard is located on the left-side panel.

For the Resource Performance Dashboard to be effective, it is highly recommended that you set the Time-Allow-Origin HTTP header in all of your CORS domains to enable access to resource timing information. Without this header, the JavaScript Agent cannot capture the resource size, and, of the supported Resource Timing Metrics, only the resource load time can be calculated.

Overview Tab

The **Overview** tab displays widgets providing high-level indicators of resource performance over a specified time period. The dashboard can be filtered to Real User or Synthetic data. The widgets only show a small number of resources, but you can click **See More** to view up to 100 resources per widget.
Violations Tab

The Violations tab shows a list of pages, resource types, and domain or resource violations that have exceeded performance thresholds. You can use the Violations tab to not only find problematic resources, but also to become aware of sudden changes that negatively impacted the performance of a resource. Clicking a specific violation leads to the Resources tab, and the data is filtered with that violation for further diagnostics. The configured violation rules are evaluated every 10 minutes for the last 30 minutes.
Violation Types

The supported violation types include the following:

- Median Domain Load Time
- Average Domain Size
- Average Page Size
- Median Resource Load Time
- Average Resource Size
- Median Resource Type Load Time
- Average Resource Type Size

Resources Tab

You can use the Resources tab to diagnose a problematic resource. You can also add criteria as a filter to the widgets. All use cases to troubleshoot a resource leads to the Resources tab where you can learn more about an individual resource's impact on an application. See Use Cases for examples.
Use Cases for the Resource Performance Dashboard

On this page:

- Troubleshoot a Resource from Sessions
- Discover Top Resource Violations
- Analyze Resources by Region
- Diagnose Resources

This page describes top use cases for the Resource Performance Dashboard.

Troubleshoot a Resource from Sessions

In this scenario, you have discovered poor application performance from the Browser App Dashboard. Now you want to investigate a particular session and see if a resource is causing the poor performance. Follow these steps to find and analyze a resource from Sessions:

1. Navigate to Sessions.
2. Add criteria and set the page experience to Very Slow.
3. Double-click on the a slow session record.
4. In the Session Summary select the page with the longest End User Response Time.
5. In the Waterfall tab, click a problematic resource to open the Resource Details dialog.
6. Click Analyze Resource.
7. You are redirected to the Resource Performance > Resources tab. The resource becomes the filter for the dashboard widget data.

8. In the widgets, look for patterns or possible factors such as geographic location, domains, etc. See Diagnose Resource Issues to learn some strategies for diagnosing resource issues.

9. Next steps:
If you want to rule out environmental factors such as network connections, browser, etc., create a synthetic job that calls a page with the resource to confirm the resource is the problem.

Discover Top Resource Violations

In this scenario, you want to analyze top violations with the most occurrences in your application. Follow these steps to learn more about a resource with the most performance violations:

1. Navigate to the **Resource Performance** > **Violations** tab.
2. Select the **Occurrences** column so that number of occurrences is in descending order.
3. Double-click the top-occurring resource.

4. You are redirected to the **Resources** tab with information for the resource in violation.
5. Next steps:
   - Create a search to track and monitor this resource by following the instructions in Perform Diagnostics on Resources.

**Analyze Resources by Region**

In this scenario, you want to view a resource's performance from a particular region. Follow these steps to analyze and monitor a resource based on a region:

1. Navigate to the Resources tab.
2. From the Add Criteria drop-down, select Load Time and specify the value to be greater than 10000 (10 seconds).

3. Click the Save icon ( ).
4. In the Create dialog, enter “Resources Loading Slower Than 10 Seconds,” and click OK.
4. Scroll down to the “Resource Load Time by Region” widget.

5. Diagnose Resources

In this scenario, you want to perform root-cause analysis for a poor-performing resource. Follow these steps to diagnose why certain image resources are taking longer than normal to load:

1. From theViolations tab, double-click on a resource. You are redirected to the Resources tab with the resource applied as a data filter.
2. Select criteria fields to help you understand possible causes, such as Region, Cached, Compressed, Load Time, Domain, etc.

- Example: Confirm that resources are cached or compressed.
- Example: Add criteria that may indicate issues with the server where the resource is coming from, such as DNS Lookup Time, Connection Time, Wait Time, and Request Time.

3. You can look for patterns that indicate the cause within the widgets, providing insights such as:
   - The resource load time speed in certain regions.
   - The top pages using the resource.
• The median load time breakdown to show that load time slows down at certain times.
• The resource size increases.

4. **Next steps:**
   • Run a synthetic job to remove possible environmental factors. See Configure Synthetic Jobs.
Configure Resource Violation Rules

On this page:
- Pages Violation Rules
- Domain Violation Rules
- Resource Type Violation Rules
- Individual Resource Violation Rules

This page describes how to configure resource violation rules for the Resource Performance Dashboard. You can configure violation rules for pages, domains, resource types, and individual resources. You can use or edit the default rules per category or create your own rules. Resource Performance configuration is only available to admin-level users.

To configure resource violation rules, navigate to Configuration > Resource Performance.

Pages Violation Rules

Page violations help monitor your most important pages and ensure they are light and efficient for end users. You can set violation rules for all pages, individual pages, or by pages matching query criteria within an application. The average page size can be set to megabytes, kilobytes, or bytes. This helps you troubleshoot application performance based on pages.
Use Case

In this scenario, your home page size average is 1500 KB, and you want to be notified when it surpasses that threshold. You can set your home page at a maximum size of 1500 KB. If the violation rule is triggered, you will see the violation in the Violations tab as well as in the “Resource Performance” widget in the Browser App Dashboard.

Domain Violation Rules

Domain violations help monitor the effects of domain performance on your application. You can set violation rules for all affected domains, individual domains, or by domains matching query criteria within an application. The median load time per page view can be set to milliseconds or seconds, and the average domain size can be set to megabytes, kilobytes, or bytes. This helps you troubleshoot poor application performance based on domains.

Use Case

In this scenario, you want to monitor the performance of your CDN. You can set your CDN domains to a maximum median load time of 1000 ms for all domains in the application. If the violation rule is triggered, you will see the violation in the Violations tab as well as in the “Resource Performance” widget in the Browser App Dashboard.

Resource Type Violation Rules

Resource type violations help monitor the effects of resource types, such as images, html, css, etc., within your application. You can set violation rules for all affected resource types or by individual resource type within an application. You can also set the rule to one page, pages matching query criteria, or all pages in the application. The median load time per page view can be set to milliseconds or
seconds, and the average domain size can be set to megabytes, kilobytes, or bytes. This helps you troubleshoot poor application performance based on resource types.

**Use Case**

In this scenario, you want to monitor all the images in all pages across your application. You can set all images to maximum average size of 500 KB for all pages in the application. If the violation rule is triggered, you will see the violation in the Violations tab as well as in the "Resource Performance" widget in the Browser App Dashboard.

**Individual Resource Violation Rules**

Individual resource violations help monitor the effects of one resource across your application. You can set violation rules for all affected resources or by resources matching query criteria within an application. The median load time per page view can be set to milliseconds or seconds, and the average domain size can be set to megabytes, kilobytes, or bytes. This helps you troubleshoot poor application performance based on individual resources.

**Use Case**
In this scenario, you want to monitor all resources on your home page. You can set all resources to a maximum load time per page view at 2 ms within the scope of the home page. If the violation rule is triggered, you will see the violation in the Violations tab as well as in the "Resource Performance" widget in the Browser App Dashboard.
## Resource Timing Metrics

Resource timing metrics are the median times in milliseconds. The resource timing metrics shown in the Resource Performance Dashboard are raw data. For a visual representation of the resource timing, see the Resource Timing Overview diagram.

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>How It’s Calculated</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browser Wait</td>
<td>startTime or redirectEnd to fetchStart</td>
<td>Time from the redirection or request start until the browser begins to fetch the resource.</td>
</tr>
<tr>
<td>Redirect Time</td>
<td>redirectStart to redirectEnd</td>
<td>Time for all redirects associated with fetching resources.</td>
</tr>
<tr>
<td>DNS Wait Time</td>
<td>fetchStart to domainLookupStart</td>
<td>Time from when the browser starts to fetch the resource until the browser starts the domain name lookup for the resource.</td>
</tr>
<tr>
<td>DNS Time</td>
<td>domainLookupStart to domainLookupEnd</td>
<td>Time to complete the domain lookup portion of the server connection time.</td>
</tr>
<tr>
<td>TCP Wait Time</td>
<td>domainLookupEnd to connectStart</td>
<td>Time for the browser to find the domain name for the resource before a connection is made.</td>
</tr>
<tr>
<td>Connection Time</td>
<td>connectStart to connectEnd</td>
<td>Time to complete the establish the TCP connection for a resource: the equivalent of one network roundtrip of latency.</td>
</tr>
<tr>
<td>SSL Time</td>
<td>secureConnectionStart to connectEnd (if secureConnectionStart exists)</td>
<td>Time to establish a secure connection to the server.</td>
</tr>
<tr>
<td>Request Wait Time</td>
<td>connectEnd to requestStart</td>
<td>Time for the request is made for a resource from the server, cache, or local resource before the connection is closed.</td>
</tr>
<tr>
<td>Request Time</td>
<td>requestStart to responseStart</td>
<td>Time for the browser to complete the request for a resource from the server, cache, or local resource.</td>
</tr>
<tr>
<td>Response Time</td>
<td>responseStart to responseEnd</td>
<td>Time for the browser to download the complete HTML document content. In the context of an Ajax request, the time for the browser to download the complete Ajax response.</td>
</tr>
<tr>
<td>First Byte Time</td>
<td>startTime to responseStart</td>
<td>Time from when the request is made until the first byte of the resource is received by the browser.</td>
</tr>
<tr>
<td>Resource Load Time</td>
<td>startTime to responseEnd</td>
<td>Time for the browser to start a request and receive a response for a resource.</td>
</tr>
<tr>
<td>Browser Queue Time</td>
<td>Browser Wait + DNS Wait Time + Request Wait Time</td>
<td>Total wait time of the resource.</td>
</tr>
<tr>
<td>TCP Time</td>
<td>secureConnectionStart to connectStart</td>
<td>Time spent to establish a secure connection to the server (same as SSL Time).</td>
</tr>
</tbody>
</table>
Browser Real User Monitoring

On this page:
- Monitor Your Application
- How Browser RUM Works
- Set Up and Configure Browser RUM
- License and Enable Browser Monitoring

Related pages:
- End User Monitoring

Search the Browser RUM topics:

Browser Real-User Monitoring (Browser RUM) allows you to see how your web application is performing from the point of view of a real end user. You can answer questions like:

- Which 1st or 3rd-party Ajax or iframe calls are slowing down page load time?
- How does server performance impact end-user experience in aggregate or in individual cases?

You can drill into the data to explore how users experience your application in their web browsers.

Monitor Your Application

Browser RUM offers multiple ways to look at your data in real time. You can:

- Understand and improve your web page's performance
  - Know how your pages, Ajax requests (XHR, Fetch API calls), and IFrames are performing over time. See The Pages & Ajax Requests View.
  - Gain insight into individual requests, with detailed charts on how your pages, Ajax requests, and IFrames load and build in your end user's browsers, with links, if enabled, to reports on server-side performance. See Browser Snapshots.
  - Find your worst performing pages by multiple common metrics. See Top Pages.
- Reduce errors
  - Learn which pages are loading with JavaScript errors, including the script file and line number that are creating the problem. See Browser Snapshots.
- Learn about your users
  - See how your web users are connecting to your application by device/platform and browser. See Browser App Dashboard.
  - Find out where geographically your web users are and how your application is performing across countries and regions. See Browser App Dashboard.
  - Understand how users are navigating through your website and what actions they are taking. See Browser RUM Sessions.

For more information on using Browser RUM, see Overview of the Controller UI for Browser RUM.

How Browser RUM Works

Browser RUM works in the following way:

1. An end user requests the first page from your web application.
2. Your web application executes whatever business logic that particular page requires.
3. Your web application creates the response page to return to the end user. The response page includes:
   a. application-specific information,
   b. a copy of a small JavaScript script that knows how to collect relevant performance information about that page. This script is called the JavaScript Agent.
4. The page, with the JavaScript Agent included, is returned to the end user.
5. As the page is being constructed in the browser, the script collects relevant information about the page's performance.
6. At approximately the same time as the `onload` event for the page fires, a copy of a somewhat larger JavaScript file, the JavaScript Agent extension, is downloaded asynchronously by the injected agent.
7. This second script packages the collected performance information and sends it via a web beacon to the EUM Server collector for processing.
8. The two scripts work together to collect and send performance information as the end user navigates through the instrumented pages of your application.

**Set Up and Configure Browser RUM**

Browser RUM is easy to set up. It is also highly configurable. You can:

- Set up and enable Browser RUM. See Set Up and Access Browser RUM.
- Instrument your application to work with Browser RUM. For more information, see Configure the JavaScript Agent.
- Set up how your information appears in the Controller UI. For more information, see Configure the Controller UI for Browser RUM.

**License and Enable Browser Monitoring**

Browser RUM requires a separate license and must be enabled before it is available for use.

For information about licensing, including a description of the types of licenses, Lite and Pro, see Browser RUM Licenses.

For information on enabling or disabling Browser RUM, see Enable and Disable Browser RUM.
Set Up and Access Browser RUM

On this page:

- EUM Account
- Access Browser Monitoring Configuration
- Browser RUM Prerequisites
- Enable and Disable Browser RUM
- Inject the JavaScript Agent for Browser RUM into Your Application Pages
- External Access Locations
- Additional Browser RUM Configurations

Related pages:

- Browser RUM Licenses
- Inject the JavaScript Agent

You must have an EUM account and a Browser RUM license to access Browser RUM. See EUM Account.

EUM must be enabled through the Controller UI. You can enable and then disable EUM if you decide not to use the feature. See Enable and Disable EUM.

Metrics are collected on your end users' experience in their Web browsers using a special JavaScript agent for Browser RUM. Your web application must be configured to insert this agent into the web pages it serves for them to be monitored. This process is called injection. See Inject the JavaScript Agent.

EUM Account

A special EUM account is required. An EUM account is created when you have a license for any of Browser RUM, Browser Synthetic, or Mobile RUM. There is an account license key associated with this account.

For on-premises customers, the account license key is set up within the Controller license file. If you added EUM after your initial installation, you may need to upload and install the new Controller license file. If you are a SaaS customer, the EUM license is set up in the SaaS Controller for you.

Each Browser RUM license provides a certain number of page views per year. For information on how to examine your current page view usage, see Browser RUM Licenses.

Access Browser Monitoring Configuration

Your data is displayed in the Controller UI in the context of an EUM application, which can be any name you choose. An EUM application name is associated with an EUM App Key, which identifies data coming into the Controller.

1. If you do not yet have an EUM application, click the User Experience tab, the Browser Apps tab, and the +Add button. You can create an application name either by using the Getting Started Wizard or by simply choosing an application name.
2. Open the EUM application you are interested in by double-clicking on the application name in the Browser Apps list.
3. In the left navigation bar of the application UI, click Configuration.

Browser RUM Prerequisites

To turn on Browser RUM, you need to:

- enable Browser Monitoring.
- inject the JavaScript Agent into your application pages.

If you are an on-prem customer, your JavaScript Agent version should be equal to or less than your EUM Server version.

Enable and Disable Browser RUM

At the top of the Configuration > Instrumentation page, toggle the Browser Monitoring:
To disable Browser RUM, toggle the switch to **OFF**.

**Inject the JavaScript Agent for Browser RUM into Your Application Pages**

The JavaScript Agent for Browser RUM collects Browser RUM metrics. See [Browser RUM Metrics](#) for more detail.

The JavaScript Agent for Browser RUM must be inserted into the headers of the pages for which you want to see these metrics. This process is called *injection*. There are several ways to accomplish this. See [Inject the JavaScript Agent](#).

**External Access Locations**

Browser RUM is made up of several components, which, in various configurations, can either be located on the Internet or hosted inside your own data center/network. On-premises access points are configured at installation or through the UI. If your installation requires access to any of these components on the Internet, see [Access the SaaS EUM Server](#) for more information.

If you are using the AppDynamics CDN to fetch the JavaScript Agent (adrum.js) and JavaScript Agent extension (adrum-ext.js), you also need to make sure that cdn.appdynamics.com is accessible from your network.

**Additional Browser RUM Configurations**

You can also configure:

- Page Identification and Naming
- JavaScript and Ajax Error Detection
- Browser Snapshot Thresholds
- Browser Snapshot Collection Rules
Inject the JavaScript Agent

On this page:

- How the JavaScript Agent Works
- JavaScript Agent hosting options
- Cross-domain session correlation
- Set the Geo Server URL
- Custom Configuration
- JavaScript Agent Limitations

Related pages:

- Overview of Injection Types
- Manual Injection of the JavaScript Agent
- Automatic Injection of the JavaScript Agent
- Assisted Injection
- Undo Injection

To instrument your application for Browser RUM, you must configure your JavaScript Agent and insert it into the page that is returned to the end user as part of the normal process it follows. The act of inserting the agent is called *injection*.

There are several ways to inject the JavaScript Agent for Browser RUM into your web pages. You also have several JavaScript hosting options to choose from that offer differing degrees of control, simplicity, and reliability.

This page provides an overview of configurations for the JavaScript Agent. For instructions on how to configure the JavaScript Agent, see the following:

- Manual Injection of the JavaScript Agent
- Automatic Injection of the JavaScript Agent
- Assisted Injection

How the JavaScript Agent Works

The JavaScript Agent consists of two files: `adrum.js` and `adrum-ext.js`.

1. The file `adrum.js` is injected into each instrumented page, as close to the top as possible, as the page is served. The file loads synchronously at around 90 KB (30 KB with compression).
2. This first JavaScript file starts a timer and does some quick configuration and setup.
3. At the `onload` event, `adrum-ext.js` is fetched *asynchronously*, specifically to not block the page from loading. This is the code that does most of the heavy lifting. Once it has been fetched, it is cached for 24 hours on the browser.
4. When cross-domain session correlation is enabled, the first time a user visits a domain, the browser additionally loads the file `adrum-xd.html` to generate or load a piece of common information that enables cross-domain sessions. Future visits will not load `adrum-xd.html` again.
5. When the page has completed loading, the collected data is bundled into a beacon and sent to the EUM Server by `adrum-ext.js`.
6. The data is processed by the EUM Server and then made available for pickup by the Controller.

**JavaScript Agent hosting options**

The `adrum.js` file is inserted into the page when it is downloaded from your web application. The `adrum-ext.js` (and `adrum-xd.html` when cross-domain session correlation is enabled) file is loaded asynchronously by the `adrum.js` file. By default, `adrum-ext.js` and `adrum-xd.html` are fetched by the highly available Amazon CloudFront CDN infrastructure.

There are three hosting options for the JavaScript Agent:

- **AppDynamics CDN** - All the JavaScript Agent files are from the host `cdn.appdynamics.com`. This is the simplest hosting...
option and ideal for testing.
- **Self-Hosting** - You host all of the JavaScript Agent files. This option gives you the most control and is recommended for production.
- **Shared Hosting** - You host the main file `adrum.js`, but the other files that are loaded asynchronously are from the AppDynamics CDN. This choice offers control of the most important file of the JavaScript Agent and is ideal for small to medium-sized businesses that don’t have the resources or prefer not to host the entire JavaScript Agent.


### Cross-domain session correlation

By default, Browser RUM sessions are restricted to one domain. Thus, when an end user navigates to a page in another domain or even subdomain, the session for that user is ended. You can, however, configure Browser RUM to enable sessions to continue across subdomains. Thus, when an end user navigates from http://example1.com to http://example2.com, the session will continue as long as the configured session inactivity time has not expired.

**How Sessions Are Correlated Across Multiple Domains**

For sessions to be correlated across domains, each page is required to use HTTPS to load the file `adrum-xad.html` from the same URL. If your pages are fetching the files `adrum-ext.js` and `adrum-xad.html` files from the AppDynamics CDN (this includes the shared hosting use case), sessions will automatically be correlated across domains because the pages will be fetching the file `adrum-xad.html` from the AppDynamics CDN.

For those **self-hosting** the JavaScript Agent files, you will need to configure the JavaScript Agent to use HTTPS to load the `adrum-ext.js` file from the same URL. The `adrum-xad.html` file is served from the location of the `adrum-ext.js` file. See the sections below for configuration instructions for the two use cases.

### Configuration for Session Correlation Across Multiple Domains

#### AppDynamics CDN / Shared Hosting

When using the AppDynamics CDN or shared hosting, your JavaScript Agent configurations must enable cross-domain session correlation and use HTTPS to call the `adrum-ext.js` file. Thus, in your JavaScript Agent configuration make sure you have the following lines:

```javascript
...  
config.adrumExtUrlHttps = 'https://cdn.appdynamics.com';  
config.xd = {enable : true};  
...  
```

#### Self-Hosting

When using self-hosting, your JavaScript Agent configurations must enable cross-domain session correlation and use HTTPS to call the `adrum-ext.js` file from the same URL. Thus, in your JavaScript Agent configuration make sure you have the following lines and that the value for `adrumExtUrlHttps` is the same for all pages requiring cross-domain session correlation.
... 
// HTTPS is needed to fetch adrum-xd.html.
config.adrumExtUrlHttps = 'https://<your-adrum.ext-host>'; 
// This config enables cross-domain session correlation.
  config.xd = {enable : true}; 
...

Set the Geo Server URL

By default, end-users' locations are resolved using public geographic databases. You can host an alternate geo server for your countries, regions, and cities instead of using the default geo server hosted by AppDynamics. See Install and Host a Custom Geo Server for Browser RUM.

Custom Configuration

You can add configuration for adrum.js to customize and extend the functionality of the JavaScript Agent. See Configure the JavaScript Agent for instructions and examples on adding custom configuration.

JavaScript Agent Limitations

The JavaScript Agent cannot monitor and report the activity of service workers or web workers. For example, if you use a service worker to make an XHR request, the JavaScript Agent will not be able to report it.
Overview of Injection Types

On this page:
- Manual Injection
- Automatic Injection
- Assisted Injection
- Container-Based Injection
- Choosing an Injection Method

You will need to inject the JavaScript Agent into your Web pages to use Browser RUM. There are several injection types that you can use depending on your use case. The following sections discuss the different injection types, how they work, and when to use each type.

**Manual Injection**

Manual injection is supported on all platforms and frameworks. To set up a manually injected page:

- Choose a JavaScript Agent hosting option. See [JavaScript Agent Hosting Options](#) to learn what hosting option best suits your use case.
- Manually configure each page you wish to instrument to find the location of the JavaScript Agent, so that it can be executed as the page is constructed by the browser.

For detailed instructions on using manual injection in your application, see [Manual Injection of the JavaScript Agent](#).

**Automatic Injection**

If you are using a Java or .Net app agent on the server-side, and your application is running in an environment that supports the Apache Jasper JSP compiler (for Java) or ASP.NET or ASPX (for .NET), you may be able to use automatic injection, where the server-side app agent completely manages injecting the code at runtime. For detailed instructions on using automatic injection in your application, see [Automatic Injection of the JavaScript Agent](#).

**Assisted Injection**

Assisted injection is available in two variants. In both cases, some of the work is done manually by you and some of the work is done by the server-side Java or .Net app agent.

**Assisted Injection Using Injection Rules (Java Only)**

In this type of assisted injection, you configure rules that define which app server Java classes and methods write to the output stream of your application and the writer object that is used to do that writing. AppDynamics intercepts the method and injects the JavaScript Agent into the output stream. You also specify which server-side business transactions you wish to have instrumented in this way.

For detailed information on using this form of assisted injection, see [Assisted Injection](#).

**Assisted Injection Using Attribute Injection**

In this type of assisted injection, you copy small code snippets appropriate to your framework into your page templates or other code that creates your pages. This snippet contains the two variables `JS_HEADER` and `JS_FOOTER`, which the app agent replaces with the appropriate information in the response object at runtime.

For detailed information on using this form of assisted injection, see [Using Attribute Injection](#).

**Container-Based Injection**

If you are using Nginx or Apache as a web container or as a reverse proxy in front of your web container, you can use directives to inject the agent into the response object. See [Injection Using Nginx](#) or [Injection Using Apache](#) for more information.

**Choosing an Injection Method**

If you are uncertain which procedure to use to inject the agent into your web pages, follow these guidelines in the given order:

- If you want to use Browser RUM and do not have any app agents on the server side, use manual injection. See [Manual Injection of the JavaScript Agent](#).
- If automatic injection is available and works for your framework, use automatic injection. Automatic injection requires the least
amount of effort because you do not have to manually instrument every page. Check the matrices at Browser RUM Supported Environments to see if automatic injection has been tested in your environment. Also, see Automatic Injection of the JavaScript Agent.

- If you cannot use automatic injection, and you can edit the source code of your web pages, use manual injection. See Manual Injection of the JavaScript Agent.
- If you cannot use automatic injection, and you can edit the source code of your web application, use one of the kinds of assisted injection. See Using Injection Rules (Java Only) or Assisted Injection-Using Attribute Injection.
Manual Injection of the JavaScript Agent

On this page:
- Access Configuration Panel
- Use a Cloud Hosted Version of the JavaScript Agent
- Host the JavaScript Agent
- Only Host the JavaScript Agent Extension
- Verify Manual Injection
- JavaScript Agent Placement

Related pages:
- Automatic Injection of the JavaScript Agent
- Assisted Injection
- Undo Injection

For manual injection, use any of the JavaScript Agent hosting options and then manually copy a snippet of JavaScript into the header of the web pages for which you want to collect Browser RUM data.

Because the steps are slightly different depending on the JavaScript hosting option, the instructions below are organized by the JavaScript hosting option. See JavaScript Agent Hosting Options to learn what option best fits your needs.

Access Configuration Panel

1. Open the browser application in which you are interested.
2. From the left navigation menu, select Configuration > Configure JavaScript Agent.

Use a Cloud Hosted Version of the JavaScript Agent

1. For the JavaScript hosting option, select AppDynamics hosts all JavaScript agent files from cdn.appdynamics.com.
2. For the Advanced section:
   - Determine if you want to enable cross-domain session correlation.
   - If you want to use your own Geo Server, set the Geo Server URL.
   - Add custom configuration if desired.
3. Click Save Config & Generate HTML Snippet.
4. Copy the HTML snippet and paste it into the header of the pages that you want to monitor, right after the <head> tag and any <meta> tags. This HTML snippet location ensures that the JavaScript Agent loads immediately, which improves timing accuracy.

Using Previous Versions of the JavaScript Agent

Previous versions of the agent can be found at the following location, where VERSION is the version number (for example, 4.5.0.0) you want to access. For versions before 4.5, please contact support.

http://cdn.appdynamics.com/adrum/adrum-VERSION.js

Use the adrum-latest.js version if you are using a SaaS EUM Cloud and want a hosted version of adrum.js. If you are using an on-prem EUM Server and want to use a hosted version of adrum.js, you need to match the version to the version of your EUM Server.

Host the JavaScript Agent

1. For the JavaScript hosting option, select I will host all the JavaScript agent files.
2. Click Download to download the JavaScript Agent.
3. Copy or move the directory holding the JavaScript Agent files to the download location you specified. For example, if you
configured the host URL to be myhost.com, move the directory adum to the web root of the server of that host.
4. Under Specify the URLs where the files can be referenced:; enter the URL to your host for both HTTP and HTTPS. Do not include the file path or file name in the URL.
5. For the Advanced section, determine if you want to:
   - enable cross-domain session correlation.
   - use your own Geo Server, set the Geo Server URL.
   - add custom configuration.
6. Click Save Config & Generate HTML Snippet.
7. Copy the HTML snippet and paste it into the header, right after the <head> tag and any <meta> tags of the pages that you want to monitor.
8. Modify the HTML snippet so that script src attribute points to where the file path or URL path of adrum.js.

Configure Self-Hosting for On-Premises

If you are self-hosting and don’t want to use the default CDN, you can follow these instructions to provide your own CDN for hosting all of the JavaScript Agent files:

1. From the Controller Admin, navigate to the Controller Settings page.
3. For the values of those two settings, enter the URLs to where the JavaScript Agent files are being hosted.
4. Confirm that these URLs are used in the HTML snippet that you manually inject in the page.

Only Host the JavaScript Agent Extension

1. For the JavaScript hosting option, select I will host the main adrum-[version].js, but all other files should be loaded from cdn.appdynamics.com.
2. Click Download to download the file adrum.js, which is one of the JavaScript Agent files.
3. Copy or move the JavaScript Agent extension file to the download location you specified. For example, if you configured the host URL to be myhost.com, move the file adrum-ext.js to the web root of the server of that host.
4. Under Specify the URLs where the files can be referenced:; enter the URL to your host for both HTTP and HTTPS. Do not include the file path or file name in the URL.
5. For the Advanced section, determine if you want to:
   - enable cross-domain session correlation.
   - use your own Geo Server, set the Geo Server URL.
   - add custom configuration.
6. Click Save Config & Generate HTML Snippet.
7. Copy the HTML snippet and paste it into the header, right after the <head> tag and any <meta> tags of the pages that you want to monitor.
8. Modify the HTML snippet so that it points to the file path or URL path of adrum.js.

Configure Shared-Hosting for On-Premises

If you are using shared-hosting and don’t want to use the default CDN, you can follow these instructions to provide your own CDN for just hosting the main JavaScript Agent file (adrum-[version].js):

1. From the Controller Admin, navigate to the Controller Settings page.
3. For the values of those two settings, enter the URLs to where the JavaScript Agent files are being hosted.
4. Confirm that these URLs are used in the HTML snippet that you manually inject in the page.

Verify Manual Injection

Once the agent is injected, it can take the AppDynamic controller a few minutes to discover and recognize the page, which must happen before data will begin to appear. If, however, you have configured your page using manual injection and are not seeing Browser RUM metrics after running load for a while, check the web page to confirm that the JavaScript Agent for Browser RUM is present in the page. If it is not, try injecting the script again.

If after two attempts you still do not see Browser RUM metrics, try one of the other injection schemes if they are available for your platform. See Troubleshoot Browser RUM for more troubleshooting information.

JavaScript Agent Placement

The preferred location for the JavaScript Agent is at the top of the <head> section. This is because one of the most important things that adrum.js does is capture the time as the page begins to load, and it is measured against the time that other timings are
calculated, for all browsers. Many—but not all—modern browsers, however, support the Navigation Timing API (see this list), and for these browsers, timings can be acquired via the API. So locating the JavaScript Agent somewhere else on the page is possible, but useful timing information, in that case, is only available for those NavTime capable browsers.
Automatic Injection of the JavaScript Agent

On this page:

- Server-Side Agents That Support Automatic Injection
- Access the User Experience App Integration Panel
- Enable Business Transaction Correlation
- Enable Automatic Injection
- Configure Automatic Injection

Related pages:

- Injection Using Nginx
- Injection Using Apache

Automatic injection uses AppDynamics server-side agents to automatically add the `adrum` header and footer to each of your web pages.

Server-Side Agents That Support Automatic Injection

Automatic injection is available only for server-side applications built on Jasper-supported JSP (Java), ASP.NET, or ASPX (.NET) frameworks. Thus, only the [Java Agent](#) and [.NET Agent](#) support automatic injection.

Although the [Apache Agent](#) does *not* support automatic injection, you can [configure Apache](#) or [Nginx](#) to inject the JavaScript Agent.

If the server-side application does not return well-formed HTML, the JavaScript Agent may not be able to instrument the page. For example, the JavaScript Agent may not be able to instrument HTML pages that are missing elements or have unclosed tags.

Access the User Experience App Integration Panel

1. From the [Applications](#) page, open the `business application` that you want to automatically inject the JavaScript Agent into your browser application.
2. From the left navigation bar, select [Configuration](#).
3. Click [User Experience App Integration](#).

Enable Business Transaction Correlation

From the [Business Transaction Correlation](#), check the [Enable Business Transaction Correlation](#) check box to correlate Pages and Ajax requests with server-side business transactions.

Specify Business Transactions to Include Correlation Headers

You can also specify which business transactions will include or exclude correlation headers. If you do not add request match rules or request exclude rules, correlation headers will be added to all requests.

To add a request rule:

1. Click the plus icon. The [Create HTTP Request Match Rule](#) will appear.
2. Check the [Method](#) checkbox and select an HTTP method that you want to match.
3. Check the [URI](#) checkbox and enter your criteria.
4. Click [Save](#).

Enable Automatic Injection

1. Select the [JavaScript Agent Injection](#) tab.
2. From the [Inject the JavaScript Agent configured for this Browser App](#) dropdown, select the browser application you want to use automatic injection.
3. Check the [Enable Automatic Injection of JavaScript](#) check box to enable automatic injection.
4. If you enabled automatic injection, click [OK](#) in the [Confirm Enabling Automatic Injection](#) dialog.
5. Click [Save](#) in the outer configuration screen.
**Configure Automatic Injection**

After you have enabled automatic injection:

- You must specify the server-side business transactions for which automatic JavaScript injection is enabled.
- You can limit which pages to inject by creating custom match and exclude rules for automatic injection. If you do not configure these rules, by default AppDynamics injects all pages visited by the enabled business transactions.

Use these rules to fine-tune which business transactions to include or exclude from injection based on match criteria. For example, you can exclude all business transactions that have a certain string in their URLs or set a certain cookie. The configurations for include rules and exclude rules are similar. It depends on your situation whether it is more convenient to restrict transactions based on inclusion or exclusion.

**Specify Business Transactions for Automatic Injection**

You must select at least one business transaction for automatic injection.

1. From the list on the right shown in the screenshot below, select one or more business transactions. (If you don’t see any business transactions, click **Refresh List**.) Not all your business transactions may appear here—the list includes only those transactions that AppDynamics can parse for automatic injection, those based on Jasper-compiled JSPs or .Net ASP or ASPX pages.
2. Click **Add** to move the business transaction to the list on the left.
3. Repeat until all the transactions you wish to enable are on the left and those you do not wish to enable are on the right.
4. Click **Save**.

**Create Match Rules for Automatic Injection**

You may not wish to instrument every page in your application. For example, if your application has a very large number of pages, you might want to instrument only the key ones for your business, to avoid hitting licensing limits. Or when you are in the setup and test phase, you might only want to instrument a few pages to keep your initial sample manageable.

Use match rules to include or exclude certain pages:

1. Expand **Only enable Automatic Injection for certain Pages** if it is closed.
2. Click the **plus** icon to create a match rule or an exclude rule. The **Create HTTP Request Match Rule** dialog appears.

3. Select one or more criteria to match. If you select multiple criteria, **all** must match for the rule to come into effect. Use the **gear** icon to set a **NOT** condition.  
   See **Using Regular Expressions** for general information about match rules.

4. Click **Save**.

5. Click **Save** in the outer configuration screen.

You can later edit or remove a match rule by selecting it in the list and clicking the edit or delete icon.
Injection Using Nginx

On this page:
- Download the Agent
- Configure Nginx with ngx_http_sub_module
- Configure Nginx without ngx_http_sub_module

Related pages:
- Manual Injection of the JavaScript Agent
- Automatic Injection of the JavaScript Agent
- Using Attribute Injection

If you are using Nginx as your web container, or you are currently using, or willing to use, Nginx as a reverse proxy (see HowTo: Use Nginx As Reverse Proxy Server for more information), you can use a container substitution module to automatically inject the JavaScript Agent into your pages. The module intercepts the response object as it is being returned and makes a string substitution.

Download the Agent

You must first download the JavaScript Agent from the Configuration screen.

1. Open the browser application in which you are interested.
2. From the left navigation menu, select Configuration.
3. Click the Configure and download JavaScript Agent.
4. For the JavaScript hosting option, select I will host all the JavaScript agent files.
5. Click Download to download the JavaScript Agent.
6. Place the file somewhere accessible to the Nginx instance. The name of the saved file should be adrum.js.

Configure Nginx with ngx_http_sub_module

The ngx_http_sub_module module is a filter that modifies a response by replacing one specified string by another. You can use this feature to have the server automatically inject the header portion of the JavaScript Agent into a served page. For more information on the process, see the Nginx documentation, Module ngx_http_sub_module.

For example, modify the location context to replace the <head> tag with the <head> tag and the JavaScript Agent scripts.

Sample Nginx Configuration

```nginx
location / {
    sub_filter      <head>
    '<head><script>window["adrum-app-key"]="<EUM_APP_KEY>";window["adrum-star
t-time"]=new Date().getTime();</script><script type="text/javascript" src="//cdn.appdynamics.com/adrum/adrum-latest.js"></script> הפי
    sub_filter_once on;
}
```

Where /adrum.js is the path to a copy of the adrum file that is accessible to the server. Note the timer initialization: `<script>window['adrum-start-time'] = new Date().getTime();</script>`. Keeping this as close as possible to the top of the document, preferably right after the <head> tag, ensures the best possible timings.

If you use <meta> tags, you should place them right after your <head> tag, and then place the JavaScript Agent directly after the last <meta> tag. This can avoid issues with some versions of IE.
You may need to escape some characters, depending on your platform. For example, on Mac OS:

```bash
location / {
    sub_filter <head>
        '<head><script>window["adrum-start-time"] = new Date().getTime();</script><script src="/adrum.js"></script>';
    sub_filter_once on;
}
```

This Nginx module is often used for adding the Google Analytics script. Be careful not to overwrite any existing GA script when you do this.

Possible variations on the script string can be found in Configure the JavaScript Agent.

**Configure Nginx without ngx_http_sub_module**

You can configure Nginx without the `ngx_http_sub_module` module if you prefer. Insert the required script into the `conf.d/default.conf` location.

**Sample Nginx Configuration**

```bash
location / {
    sub_filter <head>
        '<head><script>window["adrum-app-key"]="<EUM_APP_KEY>";window["adrum-start-time"]=new Date().getTime();</script><script type="text/javascript" src="/cdn.appdynamics.com/adrum/adrum-latest.js"></script>';
    sub_filter_once on;
}
```
Injection Using Apache

On this page:
- Download the Agent
- Configure Apache
- Other Alternatives

Related pages:
- Manual Injection of the JavaScript Agent
- Automatic Injection of the JavaScript Agent
- Using Attribute Injection
- Using Injection Rules

If you are using Apache as your web container, or you are currently using, or willing to use, Apache as a reverse proxy, you can use a container substitution module to automatically inject the JavaScript Agent into your pages. The module intercepts the response object as it is being returned and makes a string substitution.

This method uses the two Apache modules `mod_substitute` and `mod_filter`.

Download the Agent

You must first download the JavaScript Agent from the Browser Monitoring configuration screen.

1. Open the browser application in which you are interested.
2. From the left navigation menu, select Configuration.
3. Click the Configure and download JavaScript Agent.
4. For the JavaScript hosting option, select I will host all the JavaScript agent files.
5. Click Download to download the JavaScript Agent.
6. Place the file somewhere accessible to the Apache instance. The name of the saved file should be `adrum.js`.

Configure Apache

The basic set-up takes four steps:

1. Make Sure the Modules are Loaded
2. Create an Adrum Configuration File
3. Add the Location of the Adrum Configuration File to httpd.conf
4. Restart the Web Server

*Make Sure the Modules are Loaded*

Check your global Apache `httpd.conf` file and make sure that the following two `LoadModule` commands are in the file:

```bash
LoadModule substitute_module modules/mod_substitute.so
LoadModule filter_module modules/mod_filter.so
```

*Create an Adrum Configuration File*

Create a file named `adrum.conf` with contents similar to the example below, based on your Apache version. (See this Stack Overflow post for details.) In this case, the substitution rule covers the location of the entire site (Location /) but you can also recursively select a specific directory and its subdirectories by using `/somedirectory`.

*Sample adrum.conf Apache 2.2*

In the sample code below, replace `<EUM_APP_KEY>` with your own EUM App key.
<Location "/">
    SetOutputFilter INFLATE;SUBSTITUTE;DEFLATE
    AddOutputFilterByType SUBSTITUTE text/html
    Substitute "s#<head>#<head><script>window['adrum-start-time'] = new Date().getTime();</script><script>(function(config){config.appKey='"EUM_APP_KEY";})(window['adrum-config'] || (window['adrum-config'] = {}));</script><script src='./adrum.js'></script>#inq"
</Location>

If you use <meta> tags, you should place them right after your <head> tag, and then place the JavaScript Agent directly after the last <meta> tag and before other <script> tags. This can avoid issues with some versions of IE and improve the accuracy of the resource timing.

**Sample adrum.conf Apache 2.4**

In the sample code below, replace `<EUM_APP_KEY>` with your own EUM App key.

<Location "/">
    AddOutputFilterByType SUBSTITUTE text/html
    Substitute "s#<head>#<head><script>window['adrum-start-time'] = new Date().getTime();</script><script>(function(config){config.appKey='"EUM_APP_KEY";})(window['adrum-config'] || (window['adrum-config'] = {}));</script><script src='./adrum.js'></script>#inq"
</Location>

Where `./adrum.js` is the path to a copy of the adrum file that is accessible to the server. The flags after the `#` are:

- i - matching is case-insensitive
- n - pattern is treated as a fixed string (removing the n means the pattern is treated as a regular expression)
- q - module does not flatten the buckets after each substitution - this can speed performance.

For more information, see the Apache module docs [here](#).

If your `<head>` tag has an attribute like `<head lang="en">`, you can use a regex in the substitution string and omit the n flag. In the sample code below, replace `<EUM_APP_KEY>` with your own EUM App key.

Substitute "s#(<head[^>]*>)#$1<script>window['adrum-start-time'] = new Date().getTime();</script><script>(function(config){config.appKey='"EUM_APP_KEY";})(window['adrum-config'] || (window['adrum-config'] = {}));</script><script src='./adrum.js'></script>#iq"
(Optional) Adjust for gzipped resources

If your page is compressed, the substitution won’t work unless the content is INFLATEd, the substitution is made, and then the content is DEFLATEd. There are multiple ways to do this. For example, in the FilterProvider line:

```
FilterProvider AdrumFilter INFLATE;SUBSTITUTE;DEFLATE
resp=Content-Type $text/html
```

See this Stack Overflow post for more information. If you are using Apache as a proxy, you can also instruct it not to accept gzip-encoded content.

Add the Location of the Adrum Configuration File to httpd.conf

Add the following line to your global Apache httpd.conf file:

```
Include [absolutePathTo]/adrum.conf
```

Alternatively, you can add the directives to the httpd.conf file directly instead of creating a separate adrum file.

Restart the Web Server

To pick up the new configuration, restart:

```
sudo apachectl -k restart
```

If you get a warning “Useless use of AllowOverride in line 2 of [absolutePathTo]/adrum.conf”, it can be ignored. It simply means AllowOverride is redundant. You can remove it if you wish.

Possible variations on the script string can be found in Configure the JavaScript Agent.

Other Alternatives

If you are setting up your automatic injection using an Apache instance that is configured as a reverse proxy, you must use the Location directive based method described above, with the ProxyPass and ProxyPassReverse directives also in the Location directive. If you
are using an Apache instance that is your primary web container you have two additional options for describing the actual substitution step:

- Using the Directory Directive
- Using .htaccess

**Use the Directory Directive**

You can use the Directory directive instead of the Location directive.

**Use .htaccess**

Add lines similar to this to an .htaccess file in the base document directory for your site, replacing your filter name for <MyFilter>. If you don’t have an .htaccess file, create one:

Apache 2.2

In the code samples below, replace <EUM_APP_KEY> with your own EUM key.

```
Substitute "s#<head><head><script>window['adrum-start-time'] = new Date().getTime();</script><script>(function(config){config.appKey='<EUM_APP_KEY>';})(window['adrum-config'] || (window['adrum-config'] = {}));</script><script src='./adrum.js'></script>"#inq"
```

Apache 2.4

In the code samples below, replace <EUM_APP_KEY> with your own EUM key.

```
Substitute "s#<head><head><script>window['adrum-start-time'] = new Date().getTime();</script><script>(function(config){config.appKey='<EUM_APP_KEY>';})(window['adrum-config'] || (window['adrum-config'] = {}));</script><script src='./adrum.js'></script>"#inq"
```
AddOutputFilterByType SUBSTITUTE text/html
Substitute "s#<head>#<head><script>window['adrum-start-time'] = new Date().getTime();</script><script>(function(config){config.appKey='EUM_APP_KEY'})(window['adrum-config'] || (window['adrum-config'] = {}));</script><script src='./adrum.js'></script>#inq"

Where ./adrum.js is the path to a copy of the adrum file that is accessible to the server. Make sure .htaccess is world-readable.
Assisted Injection

On this page:
- Types of Assisted Injection
- Injection Rules (Java Only)
- Attribute Injection

Related pages:
- Automatic Injection of the JavaScript Agent
- Injection Using Nginx
- Injection Using Apache

Assisted injection is when your server-side application injects the JavaScript Agent into your browser application. The following sections show you how to configure your business applications to inject the JavaScript Agent into your browser applications.

**Types of Assisted Injection**

Assisted injection can be performed with rules or through attributes. The table below summarizes the platforms supported for each type of assisted injection and the process for performing the assisted injection.

<table>
<thead>
<tr>
<th>Type of Assisted Injection</th>
<th>Supported Platforms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Injection Rules</td>
<td>Java</td>
<td>This type of assisted injection uses rules to configure which Java classes should be intercepted. You create the injection rules in the <strong>User Experience App Integration Panel</strong> of the Controller UI for the business application that will inject the JavaScript Agent.</td>
</tr>
<tr>
<td>Attribute Injection</td>
<td>Java, .NET</td>
<td>This type of assisted injection relies on templates that tell the app agent where to inject information. You enable attribute injection in the <strong>User Experience App Integration Panel</strong> of the Controller UI for the business application that will inject the JavaScript Agent. In addition, you add code snippets in the page templates that determine where the JavaScript Agent is injected.</td>
</tr>
</tbody>
</table>

**Injection Rules (Java Only)**

To have your server-side application use assisted injection of the JavaScript Agent using injection rules, you define rules to configure:

- The Java classes and methods that should be intercepted
- The Java writer object and method to use to add the agent to the response object

Assisted injection using rules is available for Java frameworks only.

Access the User Experience App Integration Panel

1. From the **Applications** page, open the business application that you want to automatically inject the JavaScript Agent into your browser application.
2. From the left navigation bar, select **Configuration**.
3. Click **User Experience App Integration**.

Create JavaScript Injection Rules

1. In the **JavaScript Agent Injection** tab, select a browser application from the **Inject the JavaScript Agent configured for this Browser App** dropdown.
2. From the **Configure JavaScript Injection** tab, expand **Create Injection Rules**.
3. Click the + icon to open the **Create Manual Injection Rule** dialog.
4. From the **Where to Inject JavaScript** tab:
   a. In the **Name** field, enter a name for the rule.
   b. Check **Enable**.
   c. In the **Class and Method to Intercept** section, define match conditions for the class and method that write to the output stream in your application. This is the class that server-side agent intercepts for injection.
   d. If the write method is overloaded:
      i. Check the **Is this Method Overloaded?** checkbox.
      ii. Click **Add Parameter**.
      iii. Add the parameters that define the method.
   b. In the **Pointer to the writer** section:
      i. Select how to obtain a reference to the writer object using either the selected method with a configured number of parameters, return value, or invoked object.
      ii. Specify a getter chain.
   c. In the **Injection options** section, specify:
      - the output stream write method the server-side agent should use to inject the JavaScript Agent
      - when the injection should occur: when the method begins or when the method ends
      - which part of the script should be injected: the header or the footer
      - optional prefix to output before writing the header or footer, such as `<DOCTYPE...>`

5. Click **Create Injection Rule**.

**Attribute Injection**

To have your server-side application use assisted injection of the JavaScript Agent using attribute injection, you:

- Enable attribute injection
- Copy code snippets into your page template

Only **Servlet containers** supported assisted injection.

**Access the User Experience App Integration Panel**

1. From the **Applications** page, open the business application that you want to automatically inject the JavaScript Agent into your browser application.
2. From the left navigation bar, select **Configuration**.
3. Click **User Experience App Integration**.

**Access the JavaScript Injection Configuration Panel**

1. From the **User Experience App Integration** page, click the **JavaScript Agent Injection** tab.
2. In the **JavaScript Agent Injection** tab, select a browser application from the **Inject the JavaScript Agent configured for this Browser App** dropdown.

**Copy Code Snippets into Your Page Template**

The following examples show code snippets that can be copied directly into your page templates or other code creating your page. These code snippets tell the app agent where to inject information. The header value must be injected at the very top of the `<head>` section and the footer value must be added at the very end of the code creating the page.

If you have already injected the header portion of the agent using manual injection, you can use these code snippets to automatically inject the footer data portion only. In this case, add only the `JS_FOOTER` values shown in the sections below.

**JSF**
JSP

```jsp
<% if (request.getAttribute("AppDynamics_JS_HEADER") != null) { %>
  <%=request.getAttribute("AppDynamics_JS_HEADER").toString();
<% } %>
<% if (request.getAttribute("AppDynamics_JS_FOOTER") != null) { %>
  <%=request.getAttribute("AppDynamics_JS_FOOTER").toString();
<% } %>
```

Servlet

```java
if (request.getAttribute("AppDynamics_JS_HEADER") != null) {
    out.write(request.getAttribute("AppDynamics_JS_HEADER").toString());
}
if (request.getAttribute("AppDynamics_JS_FOOTER") != null) {
    out.write(request.getAttribute("AppDynamics_JS_FOOTER").toString());
}
```

Groovy

```groovy
<%if test="${AppDynamics_JS_HEADER}">
  ${AppDynamics_JS_HEADER}
</%if>

<%if test="${AppDynamics_JS_FOOTER}">
  ${AppDynamics_JS_FOOTER}
</%if>
```
Velocity Template

```text
#if ($AppDynamics_JS_HEADER)
    $AppDynamics_JS_HEADER
#end
#if ($AppDynamics_JS_FOOTER)
    $AppDynamics_JS_FOOTER
#end
```

ASP.NET C#

```csharp
<% if (Context.Items.Contains("AppDynamics_JS_HEADER")) %>
    Response.Write(Context.Items["AppDynamics_JS_HEADER"]); %>
<% if (Context.Items.Contains("AppDynamics_JS_FOOTER")) %>
    Response.Write(Context.Items["AppDynamics_JS_FOOTER"]); %>
```

MVC Razor

```razor
@if(HttpContext.Current.Items.Contains("AppDynamics_JS_HEADER"))
{ @Html.Raw((string)HttpContext.Current.Items["AppDynamics_JS_HEADER"]) }
@if(HttpContext.Current.Items.Contains("AppDynamics_JS_FOOTER"))
{ @Html.Raw(HttpContext.Current.Items["AppDynamics_JS_FOOTER"].ToString()) }
```
Undo Injection

On this page:

- Reverse Manual Injection
- Undo Automatic Injection
- Undo Assisted Injection Using Attribute Injection
- Undo Assisted Injection Using Injection Rules
- To undo assisted injection using injection rules:

Related pages:

- Manual Injection of the JavaScript Agent
- Automatic Injection of the JavaScript Agent
- Assisted Injection

If you try one way to inject and it does not work, AppDynamics recommends that you undo the current injection configuration before implementing another one.

**Reverse Manual Injection**

To undo manual injection, simply delete the JavaScript Agent from your web pages.

**Undo Automatic Injection**

To undo automatic injection:

1. Navigate to the Application Dashboard of the application that is injecting the JavaScript Agent.
2. Click Configuration > User Experience App Integration.
3. From the Inject the JavaScript Agent configured for this Browser App dropdown, select your browser application.
4. From the Automatic JavaScript Injection tab, clear the Enable Automatic Injection of JavaScript check box.

**Undo Assisted Injection Using Attribute Injection**

To undo assisted injection using attribute injection:

1. Navigate to the Application Dashboard of the application that is injecting the JavaScript Agent.
2. Click Configuration > User Experience App Integration.
3. From the Inject the JavaScript Agent configured for this Browser App dropdown, select your browser application.
4. From the Configure JavaScript Injection tab, clear the Request Attribute Injection check box.

**Undo Assisted Injection Using Injection Rules**

To undo assisted injection using injection rules:

1. Navigate to the Application Dashboard of the application that is injecting the JavaScript Agent.
2. Click Configuration > User Experience App Integration.
3. From the Inject the JavaScript Agent configured for this Browser App dropdown, select your browser application.
4. From the Configure JavaScript Injection tab, double-click the rule that you’d like to disable.
5. Clear the Enabled check box.

You can also simply delete rules by selecting the rule and then clicking the Delete icon.
Upgrade the JavaScript Agent
On this page:
Compatibility Issues
Injection Code Changes
Injection Code for Different Hosting Options
Related pages:
Manual Injection
Automatic Injection
Assisted Injection
Undo Injection
If you are a 4.4 SaaS customer and have an EUM license, you can and are recommended to upgrade to the 4.5 JavaScript
Agent. Upgrading to the latest version of the JavaScript Agent will allow you to use the latest features and get the latest bug fixes.
The following sections will discuss possible compatibility issues, show you a new way to include the JavaScript Agent, and provide you
with manual injection code examples for the different JavaScript Agent hosting options.

Compatibility Issues
4.4 SaaS Deployment

If you're using the 4.4 SaaS deployment, you will not have any compatibility problems using the 4.5 JavaScript Agent because the EUM
Cloud has been updated to 4.5.
On-Premises Deployment

Your JavaScript Agent version should be equal or less than your EUM Server version. For example, if you have deployed the EUM
Server 4.4, the latest version of the JavaScript Agent you can use is 4.4. Therefore, you are highly recommended to upgrade your EUM
Server to the latest version, so you can take advantage of the latest JavaScript Agent features.

Injection Code Changes
If the URL to the server hosting your JavaScript Agent is the same for both HTTP and HTTPS, you should no longer use document.wr
ite to inject the <script> tag. Instead, just hardcode the <script> tag with the following syntax:

<script src="//cdn.appdynamics.com/adrum/adrum.js"></script>

Injection Code for Different Hosting Options
The injection code will vary slightly depending on your deployment (SaaS/on-premises). The following provides you with code snippets
for each type of deployment. Be sure to place the code snippets below right after the <head> tag.

AppDynamics CDN
Replace the string <EUM_APP_KEY> in the code below with your own EUM app key.

Copyright © AppDynamics 2012-2019

Page 73


<script charset='UTF-8'>
window['adrum-start-time'] = new Date().getTime();
(function(config){
    config.appKey = '<EUM_APP_KEY>';  
    config.adrumExtUrlHttp = 'http://cdn.appdynamics.com';
    config.adrumExtUrlHttps = 'https://cdn.appdynamics.com';
    config.beaconUrlHttp = 'http://eum-col.appdynamics.com';
    config.beaconUrlHttps = 'https://eum-col.appdynamics.com';
    config.xd = {enable : false};
})(window['adrum-config'] || (window['adrum-config'] = {}));
</script>
<script src='//cdn.appdynamics.com/adrum/adrum-latest.js'></script>

Self-Hosting

In the code snippet below, replace the string <EUM_APP_KEY> with your own EUM app key and <your-cdn.com> with the URL to your server hosting the JavaScript Agent.

<script charset='UTF-8'>
window['adrum-start-time'] = new Date().getTime();
(function(config){
    config.appKey = '<EUM_APP_KEY>';  
    config.adrumExtUrlHttp = 'http://<your-cdn.com>';
    config.adrumExtUrlHttps = 'https://<your-cdn.com>';
    config.beaconUrlHttp = 'http://eum-col.appdynamics.com';
    config.beaconUrlHttps = 'http://eum-col.appdynamics.com';
    config.xd = {enable : false};
})(window['adrum-config'] || (window['adrum-config'] = {}));
</script>
<script src='//<your-cdn.com>/adrum/adrum.js'></script>

Shared Hosting

In the code snippet below, replace the string <EUM_APP_KEY> with your own EUM app key and <your-cdn.com> with the URL to where you're hosting the JavaScript Agent.
<script charset='UTF-8'>
window['adrum-start-time'] = new Date().getTime();
(function(config){
    config.appKey = '<EUM_APP_KEY>,'
    config.adrumExtUrlHttp = 'http://cdn.appdynamics.com';
    config.adrumExtUrlHttps = 'https://cdn.appdynamics.com';
    config.beaconUrlHttp = 'http://eum-col.appdynamics.com';
    config.beaconUrlHttps = 'https://eum-col.appdynamics.com';
    config.xd = {enable : false};
})(window['adrum-config'] || (window['adrum-config'] = {}));
</script>
<script src='//<your-cdn.com>/adrum/adrum.js'></script>
Configure the JavaScript Agent

On this page:
- Add Custom Configuration for the JavaScript Agent
- Set the Beacon URL for On-Premises Deployments

Related pages:
- Overview of Injection Types
- Manual Injection of the JavaScript Agent
- Automatic Injection of the JavaScript Agent
- Assisted Injection
- Injection Problems
- Inject the JavaScript Agent

For most situations, the JavaScript Agent works correctly with the default settings. You can, however, customize adrum.js in specific situations, either by adding on to the manual injection scripts, or, in the case of automatic or assisted injection, by having your web application insert the appropriate code.

Add Custom Configuration for the JavaScript Agent

You can add the custom configuration to the Controller UI and then generate the HTML snippet with your configuration.

To do so, follow the steps below:
1. Open the Configuration page.
2. Click Configure and download JavaScript Agent.
3. Expand the Advanced section.
4. Enter your custom configuration in the text area.
5. Click Save Config & Generate HTML Snippet.
6. Copy the HTML snippet to your clipboard or download the JavaScript file.

Set the Beacon URL for On-Premises Deployments

The beacon URL is used to transmit metrics about your application to the EUM Server. When you create a custom configuration, the URLs assigned to the variable config.beaconUrlHttp and config.beaconUrlHttps in the HTML snippet must be the same as the URL assigned to the EUM property appdynamics.controller.eum.beacon.hostname and appdynamics.controller.eum.beacon.https.hostname in the Controller Setting page of the Controller Admin.

For example, the Controller Settings page shown below displays the URLs appdynamics.controller.eum.beacon.hostname and appdynamics.controller.eum.beacon.https.hostname for the properties appdynamics.controller.eum.beacon.hostname and appdynamics.controller.eum.beacon.https.hostname respectively.

<table>
<thead>
<tr>
<th>Controller Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>eum.beacon.host</td>
</tr>
<tr>
<td>eum.beacon.https.host</td>
</tr>
</tbody>
</table>

In the HTML snippet containing your custom configuration, the variables config.beaconUrlHttp and config.beaconUrlHttps shown below must have the same values as given in the Controller Settings of the Controller Admin. In the case that you're using multiple instances of the EUM Server, for example, one SaaS EUM Server and one on-premises EUM Server, you can modify the values to point to one of the EUM Servers.
<script type='text/javascript' charset='UTF-8'>
(function(config){
    config.appKey = '<EUM_APP_KEY>';
    config.adrumExtUrlHttp = 'http://cdn.appdynamics.com';
    config.adrumExtUrlHttps = 'https://cdn.appdynamics.com';
    config.beaconUrlHttp = 'http://col.eum-appdynamics.com';
    config.beaconUrlHttps = 'https://col.eum-appdynamics.com';
    config.xd = {enable : true};
})(window['adrum-config'] || (window['adrum-config'] = {}));

(function (cfg) {
    if (cfg.beacon) cfg.beacon.neverSendImageBeacon = true;
    else cfg.beacon = { neverSendImageBeacon: true };;
})(window['adrum-config'] || (window['adrum-config'] = {}));
</script>
<script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
Add Custom User Data to a Page Browser Snapshot

You can add user information that is specific to your application to a browser snapshot. The information is expressed as key-value pairs that are attached to the JavaScript Agent configuration and later included in the beacons sent to the EUM Server.

The JavaScript Agent initializes user data differently depending on the page type (base, Ajax, virtual) and the method used (immediately invoked function expressions, function pointers, and literals). See Methods for Setting Custom User Data to learn when the JavaScript Agent initializes the user data for each page type and method.

The maximum size allowed for user data in a page is 2048 characters (CORS beacon) or 100 bytes (image beacon). The maximum size includes the keys-value pairs, syntax characters such as braces, and quotation marks.

View Custom User Data

Custom user data appears in Browser Analyze, Browser Snapshots, and in Sessions. View the following tabs to learn more about how user data is viewed and used in the Controller UI.

Browser Analyze

In the following screenshot of Browser Analyze, the user data `vehicleYear` is used to sort the records. You can also select fields to view user data.
Browser Snapshots

The following screenshot of the **Browser Snapshots** tab shows you how to filter results with user data.
You view user data in the **Session Summary** of the **Session Details** dialog as shown in the following screenshot. In sessions, the custom user data is only included when the base page with the custom user data is the first page of the session.

### Methods for Setting Custom User Data

You can use several methods to set custom data for each page type (base, Ajax, virtual). Each method has its own syntax, execution time, and use case. The following table outlines the execution time and potential use case for each method and page type.

<table>
<thead>
<tr>
<th>Method Type</th>
<th>Page Type</th>
<th>Execution Time</th>
<th>Potential Use Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediately Invoked Function Expressions (IIFE)</td>
<td>Base</td>
<td>These are JavaScript functions that run as soon as they are defined. They're also known as self-executing anonymous functions.</td>
<td>If your base/virtual page or Ajax request depends on information from different resources, you can set static information for custom user data with IIFE. For example, if two different server scripts generate content for the base page, you could use the IIFE on the client to set the static user data.</td>
</tr>
<tr>
<td></td>
<td>Ajax</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Virtual</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function pointers</td>
<td>Base</td>
<td>The JavaScript Agent executes function pointers when the <code>onload</code> event is triggered.</td>
<td>Data extracted from cookies, the page, and meta data.</td>
</tr>
<tr>
<td></td>
<td>Ajax</td>
<td>This event is triggered when an Ajax call is made. Custom user data for the Ajax events are attached to AJAX requests.</td>
<td>Meta data regarding the Ajax request such as the URL, HTTP method, or the request payload.</td>
</tr>
<tr>
<td></td>
<td>Virtual</td>
<td>This event is triggered when the virtual page is created. The virtual page is a dynamically recreated version of the base page, and custom user data set for <code>VPageView</code> events are attached to the virtual page records.</td>
<td>Use if the information is derived or found somewhere on the page because of the creation of the virtual page. User-specific fields or user data set based on the URL and DOM changes.</td>
</tr>
<tr>
<td>Literals</td>
<td>Base</td>
<td>You can simply use literal values to set custom data. The values, as with IIFE, are set as soon as the values are defined.</td>
<td>Constants, static data extracted and set on the server.</td>
</tr>
<tr>
<td></td>
<td>Ajax</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Virtual</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### User Data Types

For each event callback that is triggered, you can add user data by returning the following objects containing key-value pairs of different
data types. Each user data type is an object containing properties of a specific data type.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>userData</td>
<td>This data type is for setting one or more properties of type string.</td>
<td>{ &quot;username&quot;: &quot;john_thompson&quot;, &quot;email&quot;: &quot;<a href="mailto:jthompson@gmail.com">jthompson@gmail.com</a>&quot; };</td>
</tr>
<tr>
<td>userDataLong</td>
<td>This data type is for setting one or more properties of type long.</td>
<td>{ &quot;numberOfProducts&quot;: 17, &quot;numberOfSales&quot;: 1213 };</td>
</tr>
<tr>
<td>userDataDouble</td>
<td>This data type is for setting one or more properties of type double.</td>
<td>{ &quot;monthlyVisitFrequency&quot;: 0.13333333, &quot;avgCustomerConverstion&quot;: 0.0361922 };</td>
</tr>
<tr>
<td>userDataBoolean</td>
<td>This data type is for setting one or more properties of type boolean.</td>
<td>{ &quot;returnCustomer&quot;: true, &quot;subscriber&quot;: false };</td>
</tr>
</tbody>
</table>
Syntax of User Data Objects

You add user data as objects for each page type to the JavaScript Agent configuration.

The base page, Ajax, and virtual pages have the following corresponding user data objects:

- PageView
- Ajax
- VPageView

The PageView, Ajax, and VPageView objects all have the same syntax consisting of the user data types. In the JSON Syntax tab, the property names for each user data type are arbitrary and are simply used for illustration.

```
{  
    userData: {  
        dataPropertyOne: "String",  
        dataPropertyTwo: "String",  
        ...  
    },  
    userPageName: "String",  
    userDataDate: {  
        dateProperty: Date  
    },  
    userDataLong: {  
        longProperty: Long  
    },  
    userDataDouble: {  
        doubleProperty: Double  
    },  
    userDataBoolean: {  
        booleanProperty: Boolean  
    }  
}
```
Capture Ajax Data for Setting User Data

When setting user data for the Ajax event, the JavaScript Agent configuration provides a context object that has properties for the HTTP method, the request URL, and the request payload of the Ajax request. You can use this information to set values for the user data configuration object.

Ajax Context Object

To access the URL and HTTP method of the Ajax request, you can simply access the method and url properties of the context object. For the request payload of the Ajax request, you need to first match the payload parameters (HTTP method or Ajax URL) to access the data property. See Match the Ajax Payload Parameters to learn how to define the filters to match the payload parameters.

The context object has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Data Type</th>
<th>Requirement to Access Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>The HTTP method used to make the Ajax request.</td>
<td>string</td>
<td>XHR call</td>
</tr>
<tr>
<td>url</td>
<td>The URL used to make the Ajax request.</td>
<td>string</td>
<td>XHR call</td>
</tr>
</tbody>
</table>
The request payload attached to the Ajax request. Any data type that can be passed as the body to `xhr.send`.

**Match the Ajax Payload Parameters**

To access the request payload, you need to use `xhr.payloadParams` array to match the HTTP method and/or the Ajax URL. To match the Ajax URL, you can specify one or more patterns. To match the HTTP method, you include one or more objects specifying HTTP methods.

In the "Match HTTP Methods" example, the `payloadParams` array is used to match the HTTP methods "POST" and "GET". In the "Matching URLs" example, the `payloadParams` array is used to match URLs with the following string: `*/xhr/uptime`.

### Match HTTP Methods

```javascript
<script type='text/javascript' charset='UTF-8'>
window['adrum-config'] = {
    xhr: {
        payloadParams: [{method: 'POST'}, {method: 'GET'}],
    },
    ...
}
</script>
```

### Match URLs

```javascript
window['adrum-config'] = {
    xhr: {
        payloadParams: [{urls: [{pattern: '*/xhr/uptime'}]}],
    },
    ...
}
```

**Ajax Payload Filter Examples**

**Conventional Ajax Requests**

In the following examples, the data from a conventional Ajax request payload is used to set custom user data. For capturing the request...
payload for Fetch API calls, see Ajax Requests Using the Fetch API.

**Match HTTP Methods Example**

This example only sets the user data configuration objects with data from the Ajax request payload if the HTTP method is **POST**.
<script type='text/javascript' charset='UTF-8'>
window['adrum-config'] = {
  xhr: {
    payloadParams: [{method: 'POST'}]
  },
  userEventInfo: {
    "Ajax": function(context) {
      if (context.data &&
          context.data.indexOf("purchase") > -1) {
        // The URLSearchParams API does not work on
        IE/Edge
        var payload = new
        URLSearchParams(context.data);
        return {
          userData: {
            username: payload.get("username"),
            email: payload.get("email"),
            phone: payload.get("phone")
          },
          userDataLong: {
            customerId: Number(payload.get("customer_id")),
            totalPurchases: Number(payload.get("total_purchases"))
          },
          userDataDouble: {
            purchaseTotal: Number(payload.get("total"))
          },
          userDataBoolean: {
            discounted: Boolean(payload.get("discounted")),
            vipMember: Boolean(payload.get("member"))
          },
          userDataDate: {
            purchaseTime: Date.now()
          }
        }
      }
    }
  }
};
</script>
<script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
Match URLs Example

This example only sets the user data configuration objects with data from the Ajax request payload if the Ajax URL matches the pattern ".*/transaction". For example, the URL http://example.com/transaction would match the pattern and cause the user data configuration to be set with data from the Ajax request payload.
window['adrum-config'] = {
  xhr: {
    payloadParams: [{urls: [{pattern: '.*/transaction'}]}]
  },
  userEventInfo: {
    "Ajax" : function(context) {
      if (context.data &&
        (context.data.indexOf("username") > -1) &&
        (context.data.indexOf("customer_id") > -1) &&
        (context.data.indexOf("phone") > -1)) {
        // The URLSearchParams API does not work on
        IE/Edge
        var payload = new
        URLSearchParams(context.data);
        return {
          userData: {
            username: payload.get("username"),
            email: payload.get("email"),
            phone: payload.get("phone")
          },
          userDataLong: {
            customerId:
            Number(payload.get("customer_id"))
            totalPurchases:
            Number(payload.get("total_purchases") || 0)
          },
          userDataDouble: {
            purchaseTotal:
            Number(payload.get("total") || 0)
          },
          userDataBoolean: {
            discounted:
            Boolean(payload.get("discounted") || false,
            vipMember:
            Boolean(payload.get("member") || false)
          },
          userDataDate: {
            purchaseTime: Date.now()
          }
        }
      }
    }
  }
};
</script>
<script src='//cdn.appdynamics.com/adrum/adrum-latest.js'
type='text/javascript' charset='UTF-8'></script>
Ajax Requests Using the Fetch API

The example below demonstrates how to use the JavaScript Agent configuration to match the HTTP method, the request URL, and the request body for Ajax calls using the Fetch API. The Fetch API call is also provided to add context. The JavaScript Agent cannot access the request body, however, if you supply your own Request object to the fetch method.

Fetch API Call

This Fetch API example sets the from the Ajax request payload if the HTTP method is POST.

```javascript
<script type='text/javascript' charset='UTF-8'>
// Note: URLSearchParams() is not supported in IE/Edge
let queryParams = new URLSearchParams(window.location.search.substr(1));

fetch('https://my-json-server.typicode.com/typicode/demo/posts',
{
    method: 'post',
    body: queryParams.toString()
}).then(function(response) {
    return response.json();
}).then(function(data) {
    console.log('Posted content:', data.request_url);
});
</script>
```

Match HTTP Method and URL Example

This example only sets the user data configuration objects with data from the Ajax request payload if the HTTP method is POST and the Ajax URL matches the pattern ".*/posts".
Custom User Data Examples

The following examples show you how to set custom data for different page types and using the different methods:

- Setting User Data with Function Pointers
- Setting User Data with an Anonymous Function
- Setting User Data with Multiple Methods
Setting User Data with Function Pointers

The example below assigns a value to the `userPageName` property of the `PageView` object by executing the function `extractUserData`, which extracts data from the cookies.

```javascript
<script type='text/javascript' charset='UTF-8'>
(function(config){
    (function(info) {
        info.PageView = extractUserData;
    })(config.userEventInfo || (config.userEventInfo = {}))
})(window['adrum-config'] || (window['adrum-config'] = {}));

function extractUserData() {
    var cookies = document.cookie.split(';');
    for (var i = 0; i < cookies.length; i++) {
        var cookie = cookies[i].trim();
        if (cookie.indexOf("role=") === 0) {
            var role = cookie.substring(5);
        }
    }
    return {
        userPageName: role
    };
}
</script>
<script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
```

Setting User Data with an Anonymous Function

You can also use anonymous functions that return an object as shown in this example for setting user data for virtual pages.
Setting User Data with Multiple Methods

You might also want to use a combination of literal values, named and anonymous functions as this example does for setting user data for the Ajax event. Note, the context object is only available for Ajax events, and this object has the properties data (stores the request payload), method (HTTP method used to make Ajax request), and url (the Ajax request URL).
<script type='text/javascript' charset='UTF-8'>
(function(config){
    (function(info) {
        info.Ajax = function(context) {
            return {
                userAgent: $('title').get(0).text,
                userData: extractUserData(context)
                userDataBoolean: {
                    "visited": true
                }
            }
        }
    })(config.userEventInfo || (config.userEventInfo = {}))
})(window['adrum-config'] || (window['adrum-config'] = {}));

function extractUserData(context) {
    var cookies = document.cookie.split(';');
    for (var i = 0; i < cookies.length; i++) {
        var cookie = cookies[i].trim();
        if (cookie.indexOf("email=") === 0) {
            var role = cookie.substring(5);
        }
    }
    return {
        role: role,
        url: context.url,
        method: context.method
    };
}
</script>
<script src='//cdn.appdynamics.com/adrum/adrum-latest.js'
type='text/javascript' charset='UTF-8'></script>
Set Custom Page Names

**Related pages:**
- Configure Page Identification and Naming
- Configure Page Naming for Ajax Requests
- Configure Virtual Page Naming

In the Configuration > Instrumentation page, you can configure rules that name pages, iframes and Ajax requests based on various parts of the page URL. See Configure Page Identification and Naming.

You can also configure the JavaScript Agent to use any arbitrary string, not necessarily a part of the URL, to name a page or an iframe, but not an Ajax request. To do so, you assign a string of 760 characters or less to the `userPageName` property of the PageView object. If the string length exceeds 760 characters, the page name will not be set.

For example, the configuration shown below would set the page name to “My Custom Page”. That page name would then be used to identify and group pages in the Pages & AJAX Requests page in the Controller UI.

```html
<head>
  <script type='text/javascript' charset='UTF-8'>
    (function (config) {
      (function (info) {
        info.PageView = {
          userPageName: "My Custom Page"
        }
      })(config.userEventInfo || (config.userEventInfo = {}));
    })(window['adrum-config'] || (window['adrum-config'] = {}));
  </script>
  <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
  ....
</head>
```

The default page name consists of the hostname, port, and path. For example, if the page URL is `http://example.com:8080/yourpath`, the default page name would be `example.com:8080/yourpath`. 
Set Custom Virtual Page Names

To name virtual pages with the JavaScript Agent, you need to enable SPA2 monitoring. You can configure the JavaScript Agent to use any arbitrary string, not necessarily a part of the URL, to name a virtual page. The virtual page name must consist of a string of 760 or fewer alphanumeric characters. If the string length exceeds 760 characters, the page name will not be set.

To name a virtual page, you execute the method `setVirtualPageName` with `ADRUM.command` as shown below. The `ADRUM` object is globally accessible after you load the JavaScript Agent (adrum-latest.js).

```javascript
ADRUM.command("setVirtualPageName", "myCustomVPName");
```

When you name a virtual page with `setVirtualPageName`, the custom name will be applied to the next virtual page you manually mark with `markVirtualPageBegin`. Thus, the virtual page currently being monitored will not get the custom name.

The code example below shows how to call the methods to manually mark the beginning and end of the virtual page as well as execute `setVirtualPageName` to name the next virtual page that is monitored. See Report Virtual Pages to learn how to manually mark the beginning and end of virtual pages with the JavaScript API.
.controller('VPNaming', ['$scope', '$http', function ($scope, $http) {

    $scope.startVirtualPageWithCustomUrl = function () {
        console.log("Marking the beginning of the virtual page and waiting for the end to be manually marked.");
        ADRUM.markVirtualPageBegin("homepage", true);
    }

    $scope.startVirtualPageNotWaitingForMarkVirtualPageEnd = function () {
        console.log("Marking the beginning of the virtual page and allowing the JS Agent to mark the end of the virtual page.");
        ADRUM.markVirtualPageBegin("homepage", false);
    }

    $scope.endVirtualPage = function () {
        console.log("Manually marking the end of the virtual page.");
        ADRUM.markVirtualPageEnd();
    }

    $scope.setVirtualPageName = function () {
        console.log("Setting a custom name for the next virtual page to be monitored: have it's beginning and end marked.");
        ADRUM.command("setVirtualPageName", "myCustomVPName");
    }
};
Set Ajax Request Names Based on Captured POST Parameters

On this page:
- Configure the JavaScript Agent to Capture Parameters
- Create an Ajax Include Rule
- View Results in Pages & Ajax Requests

Related pages:
- Configure Page Naming for Ajax Requests
- Configure Virtual Page Naming
- Page, Ajax, and Iframe Dashboards
- Set Custom Page Names

You can configure the JavaScript Agent to capture POST parameters and then use the parameter(s) to name the Ajax request in the Pages & Ajax Requests page. This enables you to identify and sort Ajax requests from the same page based on POST parameter(s).

For example, customers on your home page of your website can either register an account or log in. The Ajax request may pass the parameter action to reflect one of these user actions. By capturing the action parameter, you can differentiate the Ajax requests to monitor performance and debug issues.

Configure the JavaScript Agent to Capture Parameters

You use the xhr object to configure the JavaScript Agent to capture POST parameters. The xhr object has the property parameter that is used to match resource URLs and define a callback for parsing the request body and return the desired results.

In the example below, the configuration sets a pattern to match the URL http://www.mystore.com, parses the request body, and returns an object containing the parameter action. As mentioned earlier, you could use a configuration like this to differentiate Ajax requests that are sending requests to register or log in users.
Create an Ajax Include Rule

To name an Ajax request based on a captured request parameter, you need to define an include rule that specifies the captured POST parameter. See Configure Page Naming for Ajax Requests to learn how to create include rules for Ajax requests.
The example include rule below uses the full domain and the captured POST parameter to name the Ajax request in the **Pages & Ajax Requests** page. For example, if an Ajax request is made to **www.mystore.com** and the value of the **action** parameter is **register**, the Ajax request will be named **www.mystore.com/register**.

---

### Include Rule

- **Enabled**
- **Rule Name**: Capture All Ajax Requests
- **Criteria**
  - This Include Rule applies to any URL that is not empty
  - **GET**
  - **POST**
  - **PUT**
  - **DELETE**
- **Name Pages**
  - using parts of the URL
  - **GET**
  - **POST**
  - **PUT**
  - **DELETE**
  - **HTTP Method** (GET, POST, PUT, or DELETE)
  - **Show Protocol** (Ex: http, https, etc)
  - **Show Domain** (Ex: mywebsite.com)
    - **Show Full Domain**
    - **Show Sub-domain**
- **Path Segments**
  - **Don't use path segments**
  - **Use first 1 segments**
  - **Use last 1 segments**
  - **Use segment numbers**
- **Query String Parameters to use in Page Name (Optional)**
- **POST Parameters to use in Page Name (Optional)**: action
- **What part of anchor should be used in Page Name**
  - **Don't use the anchor**
  - **Use first 1 segments**
  - **Use last 1 segments**
  - **Use segment numbers**

---

**View Results in Pages & Ajax Requests**

From the **Pages & Ajax Requests** page, you can view the Ajax requests that are named based on your include rule.

Using the configuration and include rule shown above, you might see the following Ajax request with the name **mystore.com/register**.

---

### Pages & Ajax Requests

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Requests</th>
<th>Requests per Minute</th>
<th>End User Response Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="mystore.com/register" /></td>
<td>mystore.com/register</td>
<td>983,613</td>
<td>57</td>
<td>70</td>
</tr>
</tbody>
</table>
Handle the window.onerror Event

Related pages:
- JavaScript Errors
- Configure JavaScript and Ajax Error Detection
- Visualize JavaScript Errors

If any script on your monitored web pages, including library code, sets the JavaScript `window.onerror` event, add the following method to the page immediately after setting `window.onerror`:

```html
<script>
ADRUM.listenForErrors()
</script>
```

The JavaScript Agent (`adrum.js`) sets `window.onerror` to listen for uncaught JavaScript errors. If this listener is overwritten, errors are not reported.

The agent invokes your original `onerror` handler.
Disable Browser Monitoring Programmatically

For pages in which the JavaScript Agent was injected manually, you can disable the agent programmatically by adding a script to the header.

To disable Browser Monitoring add the snippet below before `adrum.js` agent is injected.

```javascript
window["adrum-disable"] = true
```

For example:

```html
<head>
    <script type='text/javascript' charset='UTF-8'>
        // before adrum.js
        window["adrum-disable"] = true
    </script>

    <script>
        window["adrum-start-time"] = new Date().getTime();
    </script>
    <!-- adrum.js injection -->
    <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>

    ...
</head>
```

Old Browser Monitoring data is preserved, but no new monitoring data is collected while the agent is disabled.

You can re-enable monitoring by removing the disable script statement.
Set the Exact Current Domain in the JavaScript Agent Cookie

The JavaScript Agent itself writes a session cookie to the page, for timing purposes. This cookie is set when the user clicks a link and the unload event is fired. By default, the cookie is set to the broadest possible version of the originating domain (e.g., *.domain.com) to increase the likelihood that the next page opened in the same domain can access that cookie. For more on how Browser Monitoring uses cookies, see More on Cookies and Browser Monitoring Data.

In some cases, however, it may be necessary to limit the cookie to the full exact domain name. To do this, add a flag in the "adrum-start-time" script to the header of each page right after the <head> tag and before the entry that includes the location of the agent (adrum.js). The flag should read: window["adrum-use-strict-domain-cookies"] = true.

After you include the start-time line, the strict domain flag, and the agent, the <head> section in your monitored web pages should look something like this:

```html
<head>
  <script type='text/javascript' charset='UTF-8'>
    window["adrum-start-time"] = new Date().getTime();
    window["adrum-use-strict-domain-cookies"] = true;
  </script>
  <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
  ...
</head>
```
Limit Beacon Types

**Related pages:**
- Alter or Eliminate the Page Title Captured in the Beacon

The JavaScript Agent sends the information it collects to the EUM Server via a beacon, either through Cross-Origin Resource Sharing (CORS) or an image request which returns a small transparent .gif file. If you want the agent to only use CORS to transport the beacon you can turn off the image request mechanism.

Add the following snippet to your page, before you inject the adrum.js script.

```html
<head>
  <script type='text/javascript' charset='UTF-8'>
    (function (cfg) {
      if (cfg.beacon) cfg.beacon.neverSendImageBeacon = true;
      else cfg.beacon = { neverSendImageBeacon: true };"
    })(window['adrum-config'] || (window['adrum-config'] = {}));
  </script>
  <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
  ...
</head>
```
Alter or Eliminate the Page Title Captured in the Beacon

In a standard Browser EUM beacon, the value for `document.title` is collected as part of the data to be sent in the beacon. You may want to alter or eliminate the page title for security or privacy reasons.

You can choose to:

- Not collect a page title at all
- Use a title that is created by a function with 0 arguments
- Use a title that is an arbitrary string

The page title is different than the page name. The page title is just additional information included in browser snapshots, whereas, the page name is used to identify and group records in the Pages & AJAX Requests page.

Don't Use a Page Title

To remove the page title entirely, add the following snippet to your page before you inject the adrum.js script:

```html
<head>
  <script type='text/javascript' charset='UTF-8'>
    (function (config) {
      (function (page) {
        page.captureTitle = false;
      })(config.page || (config.page = {}));
    })(window['adrum-config'] || (window['adrum-config'] = {}));
  </script>
  <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
  ...
</head>
```

Use a Page Title Created by a Function

To use a function with no arguments to create a page title, before you inject the adrum.js script, define your function and call it, as in this example:
Something like this could be used, for example, to remove sensitive data from the page title.

**Use an Arbitrary String as a Page Title**

To use any arbitrary string as the page title, before you inject the `adrum.js` script, set `page.title`.

```html
<head>
  <script type='text/javascript' charset='UTF-8'>
    function title() { return 
      document.title.split('-').slice(1,3).join('-'); } // define a function
    (function (config) {
      (function (page) {
        page.title = title; // call your function
      })(config.page || (config.page = {}));
    })(window['adrum-config'] || (window['adrum-config'] = {}));
  </script>
  <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'/>
  ...
</head>
```

This would set the page title in the beacon to "My Special Page Title".
Modify Resource Sampling Options

Some pages use a very large number of resource files, more than can usefully be monitored. By default, the JavaScript Agent decides which resource files to monitor based on two factors:

- How the resources are ordered (the sampling algorithm used)
- The maximum number of resources to be evaluated

Both of these factors can be modified.

To modify the sampling algorithm, use `resTiming.sampler`. Possible values are:

- Top N: the resources that take the most time to load, up to the maximum number
- First N: the resources that load first, up to the maximum number
- Relevant N: the resources that are deemed "most relevant" by an algorithm that takes into account both overall load time and when in the sequence the resource loads. So, for example, something that takes a long time to load and is early in the load sequence is "more relevant" than something that takes exactly the same time to load, but does so later in the sequence, on the assumption that the former would have more impact on the overall user experience. This is the default.

To modify the maximum number of resources to be evaluated, use `resTiming.maxNum`.

For example, to sample based on Top N and to set the max number at 100, you could add the following snippet to your page, before you inject the `adrum.js` script.

```html
<head>
  <script type='text/javascript' charset='UTF-8'>
    (function (config) {
      config.resTiming = {
        sampler: "TopN",
        maxNum: 100
      };
    })(window['adrum-config'] || (window['adrum-config'] = {}));
  </script>
  <script src='//cdn.appdynamics.com/adrum/adrum-latest.js'
    type='text/javascript' charset='UTF-8'></script>
  ...
</head>
```
Set the Origin Location of the Request

Related pages:
- Geo Dashboard

There are some cases where it would be useful to be able to set a specific IP address/location as the origin of the request by modifying the JavaScript Agent itself. Add the following snippet to your page, before you inject the adrum.js script.

```html
<head>
  <script type='text/javascript' charset='UTF-8'>
    window['adrum-config'] = (function(config) {
      config.geo = {
        localIP: "192.168.200.255",
        city: "San Francisco",
        region: "California",
        country: "USA"
      };
      return config;
    })(window['adrum-config'] || {});
  </script>
  <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
</head>
```

You can use this snippet in two ways:

- Specify only the IP address, in which case this IP address has the highest precedence for use by the geo server, local or cloud-based, in resolving the request location.
- Specify all of IP, country, region, and city, in which case the request location is set by the script, and is not resolved by the Geo Server.

If you specify only some of the location parameters, the entire configuration is ignored. For example, if you only specified country, but not region, the entire configuration is ignored.

The value for city is only displayed in snapshots and can be any string. Using this value it is possible to specify location more precisely, as desired, for example, San Francisco, Sutter Street.
Hide All or Parts of the URL Query String

By default, we capture URLs with their query strings. Because query strings may result in very long URLs or contain information that you don’t want to expose, you may want to prevent all or parts of the query string from being displayed in the Controller UI.

You can do this by configuring the JavaScript Agent to remove query strings from URLs for the following:

- pages
- virtual pages
- XHR calls
- referrers
- scripts
- resources

Filter for URL Query Strings

The filter `filterURLQuery` is used to remove query strings from URLs. You can use it to remove all query strings or specific key-value pairs from query strings. The table below specifies the supported values and describes the result of the filter.

<table>
<thead>
<tr>
<th>Filter</th>
<th>Value</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filterURLQuery</td>
<td>true</td>
<td>No</td>
<td>Removes query strings from all URLs.</td>
</tr>
<tr>
<td>filterURLQuery</td>
<td>false</td>
<td>Yes</td>
<td>Retains the query strings for all URLs.</td>
</tr>
<tr>
<td>filterURLQuery</td>
<td>array of strings</td>
<td>No</td>
<td>Each string represents a key in the query string. The given keys are removed from the query string. Strings that do not match keys in the query string will have no effect on the filtering.</td>
</tr>
</tbody>
</table>

Unsupported values will result in the default behavior of retaining the entire query string for all URLs.

Remove All Query Strings

The configuration below will remove all URL query strings, so you’ll only see the domain name and the URL path in the Controller UI.

```html
<head>
  <script type='text/javascript' charset='UTF-8'>
    (function (config) {
      config.urlCapture = {
        filterURLQuery: true
      }
    })(window['adrum-config'] || (window['adrum-config'] = {}));
  </script>
  <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
  ....
</head>
```

Remove Specific Keys From Query Strings
This configuration will remove the key-value pairs from the query string where the keys match the strings "name", "page", and "id" given in the array.

```html
<head>
    <script type='text/javascript' charset='UTF-8'>
      (function (config) {
        config.urlCapture = {
          filterURLQuery: ['name', 'page', 'id']
        }
      })(window['adrum-config'] || (window['adrum-config'] = {}));
    </script>
</head>
```
Configure the JavaScript Agent to Use HTTPS

By default, the JavaScript Agent chooses the same transport (protocol) used to load the base page, but you can configure the JavaScript Agent to use HTTPS for a more secure network connection.

When configured to use HTTPS, the JavaScript Agent will use HTTPS to do the following:

- fetch adrum-ext.js
- fetch geolocation data
- send CORS beacons
- send image beacons

**Set the useHTTPSAlways Flag**

Following the code example below, you can set the boolean useHTTPSAlways to true to configure the JavaScript Agent to always use HTTPS.

```html
<head>
    <script type='text/javascript' charset='UTF-8'>
        (function(cfg) {
            cfg.useHTTPSAlways = true;
        })(window['adrum-config'] || (window['adrum-config'] = {}));
    </script>
    <script src='https://cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
    ...
</head>
```
Configure the Number and Length of URL Segments

On this page:
- Segment Definition
- Truncation Rules and Examples
- Default Values for Segments
- Set the Maximum Number of Segments
- Set the Maximum Length of Segments

Related pages:
- Hide All or Parts of the URL Query String

When the number of segments or the length of a segment exceeds the default maximum, the URL will appear truncated in the Controller UI. For example, the resources shown in the Resource Details tab or the Session Details dialog may appear truncated because of the number of URL segments or the length of the segments in the URL.

You can set configurations for the JavaScript Agent that set the maximum number of segments and the maximum number of characters for each segment. This allows you to view more or even less of the resource URLs in the Controller UI.

Segment Definition

A segment is a fragment of a URL. The following constitute one segment:
- `<protocol>://<domain-name>:<port>`
- `?key=value&key=value` (query string)
- `#someAnchor` (anchor)
- `/file-path/` (file path specified between two forward slashes)

Thus, the following URL consists of five segments: http://example.com:8090/a/b.html#someAnchor?test=true&segments=5

Truncation Rules and Examples

The maximum number of segments determines how many segments will be displayed in the Controller UI. The last segment will always be shown, and then the first $n$ number of segments until the maximum number is reached.

For example, if the original URL is http://example.com:8090/a/b.html#someAnchor?test=true&segment=5 and the configured maximum number of segments is 3, then the Controller UI would display the following: http://example.com:8090/a/...?test=true&segment=5

The maximum length of segments determines how many characters of a segment will be displayed in the Controller UI. The truncation rules are slightly more complicated as they depend on the type of segment:
- For the segment `<protocol>://<domain-name>:<port>` the `<protocol>` and `<port>` are displayed, and if the remaining amount of the maximum length is greater than the length of `<domain-name>`, then the entire domain name is displayed. If the remaining amount of the maximum length is less than the length of `<domain-name>`, then the Controller UI would only display http... 8090.
- For query strings, anchors, and file paths, the first character (?, #, /) is displayed as well as the last $n$ number of characters until the maximum length is reached. For example, for the segment `?test=true&segment=5`, if the maximum length is 10, then the segment will be displayed as the following: ?...segment=5

Default Values for Segments

The variable for setting the maximum number of segments to display is $\text{maxResUrlSegmentNumber}$. The variable for setting the maximum length of segments is $\text{maxResUrlSegmentLength}$.

The following are the default values for the two variables:
- $\text{maxResUrlSegmentNumber}=10$
- $\text{maxResUrlSegmentLength}=64$

Set the Maximum Number of Segments

The code snippet below sets the maximum number of segments to display at 15.
Set the Maximum Length of Segments

The code snippet below sets the maximum number of characters of segments to display at 100.

```javascript
<head>
  <script type='text/javascript' charset='UTF-8'>
    (function(config){
      config.maxResUrlSegmentLength = 100;
    })(window['adrum-config'] || (window['adrum-config'] = {}));
  </script>
  <script src='//cdn.appdynamics.com/adrum/adrum-latest.js'
    type='text/javascript' charset='UTF-8'></script>
  ...
</head>
```
Exclude Virtual Pages from Being Monitored

You can exclude virtual pages from being monitored based on the virtual pages' URLs. You define exclude filters to match URLs of virtual pages that you don't want to be monitored. You cannot, however, exclude virtual pages that you are manually reporting with the SPA2 JavaScript API.

Virtual Page Object

The vp object contains the exclude filter that specifies an array of patterns to match the URLs of virtual pages to exclude. By defining an exclude filter, you automatically enable SPA2 monitoring.

```json
"spa": {
  "spa2": {
    "vp": {
      "exclude": {
        "urls": ["pattern": 'api'], "pattern": "resources"]
      }
    }
  }
}
```

Virtual Page Filter Example

To use the virtual page filter, you set the JavaScript Agent with the exclude filter before you inject the adrum.js script.

For example, in the code snippet below, the pattern would match and exclude all virtual pages with URLs that have the strings "contact", "api", or "api" followed by one or more characters.
Limit the Number of Ajax Requests

By default, the JavaScript Agent limits the Ajax requests (using XHR or the Fetch API) sent for base or virtual pages to 250 with SPA2 monitoring enabled or 50 when it is not. Although base or virtual pages normally do not send large amounts of Ajax requests, in some cases, a page may send redundant Ajax requests, especially those containing error reports.

For example, a series of redundant Ajax requests containing a dead loop of errors and exceptions with no new information could overload the EUM Server. To prevent an overload of Ajax requests, you can configure the JavaScript Agent to limit the number of Ajax requests sent for base and virtual pages.

The code example below limits the number of Ajax request per base or virtual page to 7.

```html
<head>
    <script type='text/javascript' charset='UTF-8'>
        window['adrum-start-time'] = new Date().getTime();
        window['adrum-config'] = {
            xhr: {
                maxPerPageView: 7
            }
        };
    </script>
    ...
    <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
    ...
    </head>
```
Capture Resource Timing Data Without Loss

Most browsers stop capturing resource timing data in a page when the number of resources reaches 150 because of the limit set by the Resource Timing API. You can configure the JavaScript Agent for conventional web pages (non-single page applications) and single-page applications (SPAs), however, to overcome this limitation.

Because the JavaScript Agent configuration is different for non-SPA and SPA, see the following sections for details and instructions:

- Set and Clear the Resource Timing Buffer for Non-SPAs
- Disable the Clearing of the Resource Timing Buffer for SPAs

Set and Clear the Resource Timing Buffer for Non-SPAs

For non-SPAs, you can configure the JavaScript Agent to use a buffer to capture resource timing data and then set a flag to clear the buffer once the beacon has transmitted the resource timing data to the EUM Server. Clearing the buffer clears the browser's resource buffer array and ensures that new resources will be sent in the next beacon.

JavaScript Configuration Example

The `resTiming` object is used for configuring the buffer size (maximum number of resources to return) and whether the buffer is reset once the beacon has been transmitted to the EUM Server. The code snippet below shows you how to specify the buffer size and set the flag for clearing the buffer with the properties `bufSize` and `clearResTimingOnBeaconSend`.

```html
<head>
  <script type='text/javascript' charset='UTF-8'>
    window['adrum-config'] = {
      resTiming: {
        bufSize: 200,
        clearResTimingOnBeaconSend: true
      }
    };
  </script>
  <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
</head>
```

Disable the Clearing of the Resource Timing Buffer for SPAs

For SPAs, the JavaScript Agent by default clears the resource timing buffer after it's full and saves the data in a local buffer. You can configure the JavaScript Agent so that the resource timing buffer is not cleared to capture resource timing data.

If the resource timing buffer is not cleared, the resource timing correlation may miss some resources due to the buffer limit of the browser.
JavaScript Configuration Example

The default for clearResTiming is true. Setting clearResTiming to false, as shown below, disables the automatic clearing of the resource timing buffer. The resource timing buffer for virtual pages will also not be cleared.

```javascript
window['adrum-config'] = {
  ...
  "spa": {
    "spa2": {"clearResTiming": false}
  }
}
```
Filter XHR Calls by URLs

If you have many XHR calls (using XMLHttpRequest or the Fetch API) from your page that you do not need to monitor, you can use XHR filters to limit the agent, so that it monitors only a regex-defined list of specified calls.

When XHR calls are monitored, the absolute path of the XHR calls is provided to the JavaScript Agent. In the Controller UI, those monitored XHR calls will be displayed as configured by naming rules. If there are no naming rules, however, the absolute path of the XHR call will be displayed.

**XHR Filters**

You can use the XHR filters below to include or exclude XHR calls. The filters can be in the form of an XHR filter object or an array of XHR filter objects.

- `xhr.include`
- `xhr.exclude`

**Filter Object Structure**

The XHR filter object has the two properties `urls` and `method`. The `urls` property is an array of objects containing a `pattern` property that specifies a regular expression for matching URLs. The `method` property is a string that specifies an HTTP method. You can use one or both of the properties. If you just use the `urls` array, the matched URLs will be either included or excluded for all HTTP methods. If you just specify `method`, all calls using the specified HTTP method will be either included or excluded.

The following is the general structure for the filter object:

```
{
  urls:
  [
    {
      pattern: ".*foo.*"
    },
    {
      pattern: ".*bar.*"
    }
  ],
  method: 'GET'
}
```

**XHR Filter Example**

To use XHR filters, you must assign XHR filters to `xhr.include` and `xhr.exclude` before you inject the `adrum.js` script. It's important to note that the exclude patterns override the include patterns, so those URLs that are matched by both the include and exclude patterns will ultimately be excluded.
For example, in the code snippet below, the include pattern would match all HTTP requests to `http://somedomain/app_status/user-profile.jsp`, but the exclude pattern would exclude POST calls to that URL.

```html
<head>
  <script type='text/javascript' charset='UTF-8'>
    (function(config){
      (function(xhr) {
        xhr.include = {
          urls: [  
            {
              pattern: ".*ajax_info.txt"
            },
            {
              pattern: ".*app_status.*"
            }
          ]
        }
      xhr.exclude = {
          urls: [  
            {
              pattern: ".*user-profile.*"
            }
          ],
          method: "POST"
        }
      })(config.xhr || (config.xhr = {}));
    })(window["adrum-config"] || (window["adrum-config"] = {}));
  </script>
  <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
  ...
</head>
```
Report Events with the JavaScript API

On this page:
- Notify the Agent of Events
- Report Virtual Pages
- Report Events
- Correlate Ajax Requests

The JavaScript API enables you to manually report events to the agent so that that it can time the various parts of your virtual page loads and correlate Ajax calls to those page loads. You can also capture and report errors using this API.

Notify the Agent of Events

Events are reported to the JavaScript Agent (ADRUM) by calling the `ADRUM.report` method and passing in an event tracker object.

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ADRUM.report(tracker: eventTracker);</code></td>
<td>Notifies the agent of an event.</td>
</tr>
</tbody>
</table>

Report Virtual Pages

SPA1 Monitoring

For SPA1 monitoring, use the event tracker `VPageView` to manually report virtual pages.

SPA2 Monitoring

You are required to enable SPA2 monitoring to use the API below to manually report virtual pages for SPAs. These APIs will also work in non-SPAs if you enable SPA2 monitoring.

To report virtual pages, you mark the beginning and end of virtual pages with the methods below. Both methods are called from the `ADRUM` object. See also Set Custom Virtual Page Names.

<table>
<thead>
<tr>
<th>API</th>
<th>Parameter(s)</th>
<th>Default Value</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>markVirtualPageBegin(VPName: string, manuallyMarkVPEnd?: boolean)</code></td>
<td>VPName</td>
<td>N/A</td>
<td>Used to set the label for the virtual page. This label will be displayed in the Controller UI. If the string length exceeds 760 characters, the page name will not be set.</td>
</tr>
<tr>
<td></td>
<td>manuallyMarkVPEnd</td>
<td>true</td>
<td>A flag that indicates whether you or the JavaScript Agent mark the end of the virtual page. When set to true, you need to call <code>markVirtualPageEnd</code> to report the virtual page. When set to false, the JavaScript Agent will automatically mark the end of the virtual page.</td>
</tr>
<tr>
<td><code>markVirtualPageEnd()</code></td>
<td>N/A</td>
<td>N/A</td>
<td>Calling this method marks the end of the virtual page and triggers the JavaScript Agent to send a beacon with the virtual page information.</td>
</tr>
</tbody>
</table>

How the API Works

The steps below describe the process of manually reporting virtual pages with the API:

1. Start monitoring a virtual page by manually marking the beginning of the virtual page with `ADRUM.markVirtualPageBegin(VPName, manuallyMarkVPEnd)`. A beacon with the set virtual page name is sent to the EUM Server. If `manuallyMarkVPEnd` is set to true, the JavaScript Agent will wait for you to call `ADRUM.markVirtualPageEnd` to report the virtual page. If `manuallyMarkVPEnd` is set to false, the JavaScript Agent will automatically mark the end of the virtual page.
2. The JavaScript Agent will automatically mark the end of the virtual page.
3. You either call `ADRUM.markVirtualPageEnd()` to mark the end of the virtual page or the JavaScript Agent automatically marks the end of the virtual page.
4. The JavaScript Agent reports the virtual page metrics to the EUM Server.

**Example of Reporting Virtual Pages**

The Angular example below shows both ways to mark the beginning of a virtual page. The function `manualMarkVPEnd` calls `ADRUM.markVirtualPageBegin` that uses the default requiring you to manually mark the end of the virtual page. The function `allowJSAgentMarkVPEnd` passes the value `false` as the second parameter, so that the JavaScript Agent will automatically mark the end of the virtual page for you.

```javascript
angular.module('myApp.controllers', []
    .controller('VPCtrl', ['$scope', '$http', function ($scope, $http) {
        $scope.manualMarkVPEnd = function () {
            console.log("Mark the beginning of the virtual page and wait for markVirtualPageEnd() to be called.");
            ADRUM.markVirtualPageBegin("VPExample-ManuallyMarkEnd");
        }
        $scope.allowJSAgentMarkVPEnd = function () {
            console.log("Mark the beginning of the virtual page and allow the JS Agent to mark the virtual page end.");
            ADRUM.markVirtualPageBegin("virtualPageExample-JSAgentMarksEnd", false);
        }
        $scope.endVirtualPage = function () {
            console.log("Mark the end of the virtual page.");
            ADRUM.markVirtualPageEnd();
        }
    ...])
);    
```

**Report Events**

Events are reported to the agent using event trackers. There are three different kinds of event trackers:

<table>
<thead>
<tr>
<th>Event Tracker</th>
<th>Enabled for SPA2 Monitoring?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPageView</td>
<td>No</td>
<td>Used to track the stages of a virtual page view.</td>
</tr>
<tr>
<td>Ajax</td>
<td>Yes</td>
<td>Used to track Ajax requests.</td>
</tr>
<tr>
<td>Error</td>
<td>Yes</td>
<td>Used to track errors.</td>
</tr>
</tbody>
</table>

**Common Properties**

There are also two properties that are common to all tracker types:

- Gets or sets a URL

| API       | Description |
|-----------|-------------|-------------|
|           |             |             |
url(url?: string) | Gets or sets a URL.

- Gets or sets the parent event identifier

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent(parent?: object)</td>
<td>Gets or sets the parent event identifier.</td>
</tr>
</tbody>
</table>

**VPageView**

The following is the page view load flow in SPA1 monitoring. You'll use the SPA1 monitoring API below to set timing marks to match the below workflow as closely as possible in your own single page app framework. For SPA2 monitoring, see Report Virtual Pages: SPA2 Monitoring to learn about the SPA2 APIs for manually reporting virtual pages.

Based on the marks you set, AppDynamics derives the following key timing metrics. Marks should be called in the order in which they occur in the flow. The following table describes which marks used to calculate each metric.

<table>
<thead>
<tr>
<th>Full Metric Name</th>
<th>Short Metric Name</th>
<th>How Calculated</th>
</tr>
</thead>
<tbody>
<tr>
<td>End User Response Time (not used for waterfall UI)</td>
<td>PLT</td>
<td>virtualPageStart to virtualPageEnd</td>
</tr>
<tr>
<td>HTML Download Time</td>
<td>DDT</td>
<td>viewChangeStart to viewChangeEnd</td>
</tr>
<tr>
<td>HTML Download and DOM Building Time</td>
<td>DRT</td>
<td>viewChangeStart to viewDOMLoaded</td>
</tr>
<tr>
<td>DOM Building Time</td>
<td>DPT</td>
<td>viewChangeEnd to viewDOMLoaded</td>
</tr>
<tr>
<td>DOM Ready Time (used instead of PLT for waterfall UI)</td>
<td>DOM</td>
<td>viewChangeStart to viewDOMLoaded</td>
</tr>
</tbody>
</table>

Instantiate using ADRUM.events.VPageView().

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start()</td>
<td>Indicates when a virtual page starts. It automatically calls:</td>
</tr>
<tr>
<td></td>
<td>- startCorrelatingXhrs()</td>
</tr>
<tr>
<td></td>
<td>- markVirtualPageStart()</td>
</tr>
<tr>
<td>end()</td>
<td>Indicates when a virtual page ends. It automatically calls:</td>
</tr>
<tr>
<td></td>
<td>- stopCorrelatingXhrs()</td>
</tr>
<tr>
<td></td>
<td>- markVirtualPageEnd()</td>
</tr>
</tbody>
</table>
**startCorrelatingXhrs()**

Correlates the Ajax requests sent after this call with the virtual page view event. The last tracker calling this method wins.

This method is called automatically in the VPageView constructor. When a VPageView is created, the AJAX requests made after that call are automatically correlated to that VPageView. Use this separate call only when you want to set up manual correlation.

**stopCorrelatingXhrs()**

Stops correlating Ajax requests to the virtual page view event.

Use this separate call only when you wish to set up manual correlation.

### Setters

The default value for these is the time when the API is called.

- **markViewChangeStart()**
  Sets the view change start time.

- **markViewChangeEnd()**
  Sets the view change end time.

- **markViewDOMLoaded()**
  Sets the view DOM loaded time.

- **markXhrRequestsCompleted()**
  Sets the XHR requests completed time.

- **markViewResourcesLoaded()**
  Sets the view resources loaded time. This includes images, CSS files, and scripts.

- **markVirtualPageStart()**
  Sets the virtual page start time.

- **markVirtualPageEnd()**
  Sets the virtual page end time.

### Getters

- **getViewChangeStart()**
  Gets the view change start time.

- **getViewChangeEnd()**
  Gets the view change end time.

- **getViewDOMLoaded()**
  Gets the view DOM loaded time.

- **getXhrRequestsCompleted()**
  Gets the XHR requests completed time.

- **getViewResourcesLoaded()**
  Gets the view resources loaded time.

- **getVirtualPageStart()**
  Gets the virtual page start time.

- **getVirtualPageEnd()**
  Gets the virtual page end time.

### Ajax

Instantiate using ADRUM.events.Ajax().

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Property Setters/Getters</strong></td>
<td>Call this without a parameter to get the value and with a parameter to set the value.</td>
</tr>
<tr>
<td><strong>method(method?: string)</strong></td>
<td>Gets or sets the method (&quot;GET&quot; or &quot;POST&quot;) of the Ajax.</td>
</tr>
<tr>
<td><strong>markSendTime(sendTime?: number)</strong></td>
<td>The default value for these is the time when the API is called.</td>
</tr>
<tr>
<td><strong>markFirstByteTime(firstByteTime?: number)</strong></td>
<td>Sets the time the request is sent.</td>
</tr>
<tr>
<td><strong>markFirstByteTime</strong></td>
<td>Sets First Byte Time.</td>
</tr>
</tbody>
</table>
markRespAvailTime(respAvailTime?: number)  Sets Response Available Time.
markRespProcTime(RespProcTime?: number)  Sets the time the response is completely processed.

Timing Getters
getSendTime()  Gets the time the request was sent.
getFirstByteTime()  Gets First Byte Time.
getRespAvailTime()  Gets Response Available Time
getRespProcTime()  Gets the time the response was completely processed.

Errors
Instantiate using ADRUM.events.Error().

API  Description
Property Setters/Getters  Call these without a parameter to get the value and with a parameter to set the value.
msg(msg?: string)  Gets or sets the error message.
line(line?: number)  Gets or sets the line number of source code where the error happened.

Correlate Ajax Requests
Ajax requests can be correlated to virtual page views automatically or manually. When you create a vPageView tracker, startCorrelatingXhrs() is called automatically in the constructor, correlating any subsequent Ajax calls with that VPageView event. To set up manual correlation, call stopCorrelatingXhrs() to stop the automatic process and then call startCorrelatingXhrs() where you wish correlation to re-commence.

Sample Code

Report a custom Error event by passing properties via setters

```javascript
var errorT = new ADRUM.events.Error();
errorT.msg('I am a custom error at line 100');
errorT.line(100);
ADRUM.report(errorT);
```
Report a custom Error event by passing properties via the constructor

```javascript
var errorT = new ADRUM.events.Error({
  msg: 'I am a custom error at line 100',
  line: 100
});
ADRUM.report(errorT);
```

Report a custom Ajax event passing properties via setters

```javascript
var ajaxT = new ADRUM.events.Ajax();

// set url
ajaxT.url('your xhr Url');

// mark timings
ajaxT.markSendTime(100);
ajaxT.markFirstByteTime(200);
ajaxT.markRespAvailTime(300);
ajaxT.markRespProcTime(400);
ADRUM.report(ajaxT);
```
Set up backbone SPA monitoring

```javascript
var AppRouter = Backbone.Router.extend({
    routes: {
        "wines/:id": "wineDetails"
    },
    wineDetails: function (id) {
        var vpView = new ADRUM.events.VPageView();
        vpView.markVirtualPageStart();
        // vpView.markViewChangeStart();
        var wine = new Wine({id: id});
        wine.fetch({success: function(){
            vpView.markXhrRequestsCompleted();
            $('#content').html(new WineView({model: wine}).el);
            vpView.markViewDOMLoaded();
            vpView.markVirtualPageEnd();
            ADRUM.report(vpView);
            this.headerView.selectMenuItem();
        }));
    }
});
```
Correlate Ajax requests with VPageView Events

```javascript
var vPageView = new ADRUM.events.VPageView({
    url: 'http://localhost/#virtualpage1',
});

vPageView.start();

// SPA view routing and HTML partials fetching
vPageView.markViewChangeStart()
// AJAX requests for the HTML partials are automatically correlated with the VPageView
...
vPageView.markViewChangeEnd();

// HTML partials inserted into Browser DOM tree
...
vPageView.markViewDOMLoaded();

// SPA HTML AJAX data fetching
// Data AJAX requests are automatically correlated with the VPageView
...

vPageView.markXhrRequestsCompleted();

// call this when ending a new virtual page
vPageView.end();

ADRUM.report(vPageView);
```

You can exclude certain Ajax calls from being monitored by configuring ADRUM itself. Before you invoke the `adrum.js` script at the top of your page, add lines similar to the following:
Exclude Ajax from VPageView using ADRUM configuration

```javascript
window['adrum-config'] = {
  "spa": {
    "angular": {
      "vp": {
        "xhr": {
          "exclude": {
            "uris": [{
              "pattern": 'heartBeatAjax'
            }]
          }
        }
      }
    }
  }
}
```

Or you can exclude certain Ajax calls using the `vPageView.stopCorrelatingXhrs()` call, and then turn correlation back on with `vPageView.startCorrelatingXhrs()`, as in the following:

Exclude Ajax from VPageView event manually

```javascript
var vPageView = new ADRUM.events.VPageView();
vPageView.stopCorrelatingXhrs();

var xhr = new XMLHttpRequest();
xhr.open('GET', '/heartBeatAjax');
xhr.send();
vPageView.startCorrelatingXhrs();
```
Disable Monitoring of Fetch API Calls

By default, the JavaScript Agent reports Ajax requests made with XMLHttpRequest object (XHR) and the Fetch API.

You should only consider disabling monitoring support for the Fetch API for the following use cases:

- You just started using the JavaScript Agent v4.5.6 and start to see JavaScript errors caused by Ajax requests.
- Your browser app uses one of the following libraries:
  - Bluebird
  - shim.js
  - Zone.js

To disable monitoring for the Fetch API, you set `config.fetch` to `false` as shown below.

```javascript
<script type='text/javascript' charset='UTF-8'>
(function(config){
    config.fetch = false;
})(window['adrum-config'] || (window['adrum-config'] = {}));
</script>
<script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
```

To disable monitoring for the Fetch API for single-page applications:

```javascript
<script type='text/javascript' charset='UTF-8'>
(function(config){
    config.fetch = false;
    config.spa = {
        "spa2": true
    };
})(window['adrum-config'] || (window['adrum-config'] = {}));
</script>
<script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
```
Monitor Single-Page Applications

Browser RUM's single-page application (SPA) monitoring enables you to:

- Monitor the performance/throughput of user experiences of SPAs.
- Troubleshoot and resolve SPA problems within the context of the page load.
- Assist with business decisions by analyzing SPA data through analytics.
- Enable developers to quickly ship better web applications to the marketplace.

What Is SPA?

A single-page application (SPA) is a web technology and design paradigm that reduces browser-level page loads by using JavaScript to fetch resources and build pages. This creates a smoother, faster user experience more similar to a desktop or mobile application than a traditional web page. React and Angular are two popular JavaScript frameworks used to create SPAs.

Base Pages vs. Virtual Pages

When a user navigates to a SPA, the initial page download is considered the "base page." The base page includes the HTML skeleton, the core CSS, and the JavaScript framework for fetching and constructing new content. The ensuing pages are constructed from resources downloaded with the base page or fetched dynamically from the backend in response to user interaction. This new content (or views) constructed from different sources by the SPA are called "virtual pages."

For SPAs, you will have base pages, virtual pages, and Ajax requests. From Pages & AJAX Requests, you are able to view all three types and details for each as shown below.
SPA1 and SPA2 Monitoring

Until 4.4.3, Browser RUM only provided auto-instrumentation for AngularJS 1 SPA applications. For other frameworks, you had to manually report events with the JavaScript Agent API. This limited auto-instrumentation for SPA is known as SPA1 monitoring.

From 4.4.3 and later, Browser RUM supports auto-instrumentation for all SPA frameworks and has been certified to correctly auto-instrument Angular 1-5 and React SPAs. This support for the auto-instrumentation for all SPA frameworks is known as SPA2 monitoring.

The table below describes the support for SPA1 and SPA2 monitoring:

<table>
<thead>
<tr>
<th>SPA Monitoring Version</th>
<th>Auto-Instrumentation</th>
<th>Metrics</th>
<th>JavaScript API</th>
</tr>
</thead>
</table>
| SPA1                  | AngularJS 1 only     | • Base / Virtual Pages: Browser RUM page timing metrics  
                      |                      | • Ajax Requests: All available Ajax metrics     | Yes |
| SPA2                  | All SPA frameworks.  | • Base pages: Browser RUM page timing metrics, including End User Response Time (EURT), Visually Complete Time (VCT), and Page Complete Time (PCT)  
                      | Auto-instrumentation is certified for Angular 1-5 and React.  | • Virtual Pages: End User Response Time (EURT), Visually Complete Time (VCT)  
                      |                      | • Ajax Requests: All available Ajax metrics     | Yes |

**Which SPA Monitoring to Use?**

In general, you are highly recommended to use SPA2 monitoring. For most applications, the JavaScript Agent using SPA2 monitoring identifies the SPA transitions deterministically and captures the metrics and resources correctly without you having to write any special configuration or call methods from the JavaScript Agent API. If your SPA applications are built with AngularJS 1, you can simply add some configuration to enable auto-instrumentation.

**View SPA2 Monitoring Data**

You can view end user response time (EURT) for SPAs in many places in the Controller. The EURT measures the time from when the user navigates to a virtual page until all resources are downloaded for the page. See SPA2 Metrics for more information.

**Browser Snapshots**

The Browser Snapshots tab displays a list of different types of pages as shown below. You can view relevant metrics for all the pages and other metrics for base pages and Ajax requests.
Snapshot Details: Summary

To get to the Snapshot Details dialog shown below, you double-click one of the browser snapshots in the Browser Snapshots page. The Summary tab is the default tab for the dialog, and it gives some additional information such as the parent page URL.
Snapshot Details: Resource Details

The **Resource Details** tab shown below provides details about resources such as the number and type of resources, the domains where the resources were requested, and resource load times.
Pages & AJAX Requests

The Pages & AJAX Requests page as shown below enables you to view the number of requests for each type of page and the average metrics per page. You can also select which page types to view, such as virtual pages.

Page, Ajax, and Iframe Dashboards
From the **Pages & AJAX Requests** page, you can double-click a page to view the Page and IFrame Dashboards. Dashboards for each page type will have a **Summary** section displaying the average metrics as well as a **Key Performance Times** section showing the requests and the metrics on a timeline.

The tabs below show the differences in the dashboard for each page type.

**Base Pages**

Base pages contain timing metrics such as End User Response Time, Page Complete Time, and Visually Complete time.
Virtual Pages

Virtual pages provide the resources requested.

AJAX Requests

Ajax Requests include a more limited set of timing metrics as well as Ajax-specific section for the AJAX Response Download Time and the AJAX Callback Execution Time.
**Browser Analyze**

From Browser Analyze, you can filter by criteria, such as page types shown in the screenshot below, and configure the fields you want to view.
The following sections show you how to use SPA1 monitoring for AngularJS 1 applications and manually report events for Ember.js applications. This page also provides information about the AngularJS 1 support for SPA1 monitoring.

**Manual Injection for Angular 1.x Applications**

For non-Angular applications, adrum.js needs to be loaded before any other JavaScript on the page. When you are using AngularJS 1, however, you need to inject adrum.js after angular.js, but before angular.js is bootstrapped.

To learn which Angular versions have monitoring support, see Monitoring Support for AngularJS.

**Manual Injection for Ember.js Applications**

For Ember.js applications, you manually inject the JavaScript Agent much in the same way you would do for web pages loaded from an HTTP request. To monitor virtual pages, you manually start and end Virtual Page events based on events triggered when pages are dynamically created. You can also monitor virtual pages to report errors for pages.

The following sections will show you how to inject the JavaScript Agent and monitor virtual pages.

**Manual Injection of the JavaScript Agent**

The file app/index.html is the HTML skeleton for all dynamics pages in Ember.js. Thus, you can inject the JavaScript Agent in this file so it is included in every page.
Monitor Virtual Pages

When a user requests a new page, the route handler renders the associated template to form the new content of the virtual page. You can listen for events in the route handler indicating when the handler is started and completed, which in effect mark the lifetime of the virtual page. To monitor the virtual page, you start a Virtual Page Event when the activate event is triggered, close the Virtual Page when the deactivate event is triggered, and then report the completed virtual page.

The code snippet below listens to the activate and deactivate events and reports the created virtual page event.
import Ember from 'ember';
import config from '.././config/environment';

export default Ember.Route.extend({
    beforeEnterAbout: Ember.on('activate', function(){
        console.log('hello about');
        config.aboutVpView = new ADRUM.events.VPageView();
        config.aboutVpView.start();
    }),
    afterEnterRental: Ember.on('deactivate', function(){
        console.log('goodbye about');
        config.aboutVpView.end();
        ADRUM.report(config.aboutVpView);
    })
});

Use Virtual Pages to Capture Errors

You can also create actions in the routing handlers that can be monitored through virtual pages. For example, you might want to monitor Ajax calls. You can create an action that performs the Ajax call and then use a Virtual Page event to capture the results and errors as shown in the code snippet below.

actions: {
    makeXhrCall() {
        config.xhrVpView = new ADRUM.events.VPageView();
        config.xhrVpView.start();
        var xmlHttp = new XMLHttpRequest();
        xmlHttp.onreadystatechange = function () {
            if (xmlHttp.readyState === 4) {
                console.log(xmlHttp.responseText);
                config.xhrVpView.end();
                ADRUM.report(config.xhrVpView);
            }
        };
        xmlHttp.open("GET", "http://localhost:3000", true);
        xmlHttp.send(null);
    }
}

Monitoring Support for AngularJS 1

The JavaScript Agent supports monitoring by default for AngularJS versions 1.x.
Routing Engines

AngularJS 1 applications that have multiple Views use a route to move from one virtual page to another. You can use Browser RUM to instrument any virtual page that uses either of two routing engines, ngRoute or ui-router.

Metrics

Because virtual pages are constructed in the browser, normal page view metrics must be adjusted. In essence, what a metric for AngularJS 1 must do is correlate the time between various routing events, using their timestamps. Metrics are calculated as follows:

<table>
<thead>
<tr>
<th>Full Metric Name</th>
<th>Short Metric Name</th>
<th>How Calculated</th>
</tr>
</thead>
<tbody>
<tr>
<td>End User Response Time (not used for waterfall UI)</td>
<td>PLT</td>
<td>virtualPageStart to virtualPageEnd</td>
</tr>
<tr>
<td>HTML Download Time</td>
<td>DDT</td>
<td>viewChangeStart to viewChangeEnd</td>
</tr>
<tr>
<td>HTML Download and DOM Building Time</td>
<td>DRT</td>
<td>viewChangeStart to viewDOMLoaded</td>
</tr>
<tr>
<td>DOM Building Time</td>
<td>DPT</td>
<td>viewChangeEnd to viewDOMLoaded</td>
</tr>
<tr>
<td>DOM Ready Time (used instead of PLT for waterfall UI)</td>
<td>DOM</td>
<td>viewChangeStart to viewDOMLoaded</td>
</tr>
</tbody>
</table>

Because the two routing engines function in slightly different ways, what the AppDynamics event consists of differs slightly, based on the engine.

<table>
<thead>
<tr>
<th>AppDynamic Event</th>
<th>ngRoute Equivalent</th>
<th>ui-router Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtualPageStart</td>
<td>locationChangeStart</td>
<td>stateChangeStart</td>
</tr>
<tr>
<td>viewChangeStart</td>
<td>routeChangeStart</td>
<td>stateChangeStart</td>
</tr>
<tr>
<td>viewChangeEnd</td>
<td>routeChangeSuccess</td>
<td>stateChangeSuccess</td>
</tr>
<tr>
<td>viewDOMContentLoaded</td>
<td>viewContentLoaded</td>
<td>viewContentLoaded (may happen multiple times - timestamp overwritten each time)</td>
</tr>
<tr>
<td>viewFragmentsLoaded</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xhrRequestsCompleted</td>
<td></td>
<td></td>
</tr>
<tr>
<td>viewResourcesLoaded</td>
<td></td>
<td></td>
</tr>
<tr>
<td>virtualPageEnd</td>
<td>the latest one among viewContentLoaded, xhrRequestsCompleted, and viewResourcesLoaded</td>
<td></td>
</tr>
</tbody>
</table>

Page Load Process Visualized

Visualized the page load process looks something like this:
Compare these to the standard page metrics, which are shown in **Browser RUM Metrics**.

**Exclude Heartbeats or Background Requests from Timings**

You may wish to exclude certain events from your virtual page timings. To do this, you can customize the JavaScript Agent when you inject it.

Add the following snippet *before* you add the JavaScript Agent file, `adrum.js`, to the page:

```html
<br>
Option for excluding XHRs

```html
<script type="text/javascript">
(function(config) {
  (function(spa) {
    (function(angular) {
      (function(vp) {
        vp.xhr = {
          "exclude": {
            "urls": {
              pattern: '.*/dealActiveUsers'
            }
          }
        }
      }
    })(angular.vp || (angular.vp = {}));
  })(spa.angular || (spa.angular = {}));
})(config.spa || (config.spa = {}));
})(window["adrum-config"] || (window["adrum-config"] = {}));
</script>
```
Enable Resource Timing Collection for Virtual Pages

By default, Virtual Pages for AngularJS 1 do not include resource timing metrics. You need to set the Angular virtual page property `includeResTimingInEndUserResponseTiming` to `true`.

The JavaScript configuration below shows you how to enable resource timing collection for AngularJS 1 virtual pages. The configuration also sets limits for XHR calls per page, the buffer size for resource timings, and sets the flag for clearing resource timing metrics when a beacon is sent.

```javascript
window['adrum-config'] = {
  "xhr": {
    "maxPerPageView": 10000
  },
  "resTiming": {
    "bufSize": 300,
    "clearResTimingOnBeaconSend": true
  },
  "spa": {
    "angular": {
      "vp": {
        "metrics": {
          "includeResTimingInEndUserResponseTiming": true
        }
      }
    }
  }
}
```

View Correlated Server Times

Since there isn't a regular HTML page timing to which correlated server timings can be linked, to view server times you must drill down from the Dashboard or Snapshot virtual page view to the component XHR requests. The server times can be seen there.
For SPA2, you only need to set a flag to enable the JavaScript Agent to auto-instrument SPAs. When SPA2 auto-instrumentation is enabled, the JavaScript Agent will do the following:

- detect virtual pages
- measure the end-to-end time for virtual pages
- correlate resources, JavaScript errors, and Ajax requests
- name virtual pages

**SPA2 Requirements**

- JavaScript Agent 4.4.3+
- Controller / EUM Server 4.4.3+ (for on-premises deployments)
- Enable SPA2 monitoring

To manually report and name virtual pages with the JavaScript Agent API, you will need to use the JavaScript Agent 4.5+. The 4.4.3+ JavaScript Agent does not support the use of the JavaScript Agent API to manually report events.

**Enable/Disable SPA2 Monitoring**

In the JavaScript Agent configuration, set `spa2` to `true` and then inject the JavaScript Agent as shown below. You can use either manual or automatic injection. The default value for `spa2` is `false`, so to enable SPA2 monitoring, you must set `spa2` to `true`. 
You must set the configuration, including the setting to enable SPA2 monitoring, before you load the JavaScript Agent.

**Disable SPA2 Monitoring**

To disable SPA2 auto-instrumentation, just set \texttt{spa2} to \texttt{false}. Although the default value for \texttt{spa2} is \texttt{false}, you are recommended to set the configuration to \texttt{false} and not just remove it.

**Migrate from SPA1 to SPA2 Monitoring**

To migration from SPA1 to SPA2 monitoring:

1. Read [SPA1 and SPA2 Monitoring](#) to confirm that your use case is suitable for SPA2.
2. Meet the SPA2 requirements.
3. Enable SPA2 monitoring.

**SPA2 Monitoring Compatibility**

When you configure the JavaScript Agent to use SPA2 auto-instrumentation, the following happens:

- Metrics such as DOM ready, \# of digests, etc., are no longer measured and reported for virtual pages.

**Fetch API Support for SPA2 Monitoring**

The JavaScript Agent monitors Fetch API calls by default for all SPAs except for Angular applications.

For Angular.js and Angular 2-5 applications, follow these steps to ensure Fetch API calls are monitored.

1. The JavaScript Agent is configured to monitor Fetch API calls by default. Make sure that your JavaScript Agent configuration \texttt{does}n't have the following:
1. \[\text{config.fetch} = \text{false}\]

2. Load the JavaScript Agent before Angular (recommended, but not required).

\[
\text{<script src="adrum.js"></script>}
\text{<script src="angular.js"></script>}
\]

3. Add the following configuration to ensure the JavaScript Agent monitors the Angular application correctly.

\[
\text{function(config) \{ \}
\text{  \ldots}
\text{  config.angular = true;}
\text{  \ldots}
\text{\})(window['adrum-config'] || (window['adrum-config'] = {}));}
\]

**How SPA2 Monitoring Works**

The diagram below demonstrates how the JavaScript Agent defines the start and end points for the End User Response Time (EURT) metric and correlates with Ajax requests, resources, and JavaScript errors.
<table>
<thead>
<tr>
<th>Step</th>
<th>Browser Activity</th>
<th>JavaScript Agent Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The user navigates to the base page of a SPA. The HTML skeleton, core CSS, and JavaScript are loaded into the browser.</td>
<td>The JavaScript Agent sends a beacon for the base page.</td>
</tr>
<tr>
<td>2</td>
<td>From the base page, the user clicks a button to view products. The URL changes as the virtual page is loaded through a combination of previously downloaded content and from new content fetched through Ajax requests.</td>
<td>The JavaScript marks the user action as the start time of the virtual page.</td>
</tr>
<tr>
<td>3</td>
<td>User actions and the browser activity cease for five seconds. The browser activity includes requesting resources, making Ajax calls, and so on.</td>
<td>The JavaScript Agent or your code using the JavaScript API marks the end time of the virtual page as the last time browser activity was seen after the URL changed. The JavaScript Agent then sends a beacon for the virtual page; however, if the virtual-page load stalls on one slow activity, the JavaScript Agent waits eight seconds before marking the end of the virtual page. Any activities such as Ajax requests, resource loads, and JavaScript errors that happened during the virtual page load will be correlated to that virtual page. There are some kinds of browser activities that aren't accessible to JavaScript. For example, the time that the browser takes render the DOM to the screen after the DOM has been updated. Therefore, if your virtual pages don't request any resources, you will likely see a very fast EURT (&lt;10ms).</td>
</tr>
</tbody>
</table>

**SPA2 Metrics**

Browser RUM calculates metrics differently for base pages, Ajax requests, and virtual pages. The diagrams below demonstrate the differences between base page and virtual page metrics from the end-user perspective. See [Browser RUM Metrics](#) for metric definitions.

For base pages, Browser RUM calculates End User Response Time (EURT), Visually Complete Time (VCT), and Page Complete Time (PCT). For virtual pages, Browser RUM calculates EURT and VCT.

**End User Response Time**

End User Response Time (EURT) calculates the total time for all content (visual and non-visual) to be loaded on a page.

---

Base Page
All content (visual and non-visual) is loaded on the page.

New content is fetched through AJAX requests.
Visually Complete Time

Visually Complete Time (VCT) calculates the point in time when the browser has finished loading all visual content in the viewport.
Page Complete Time

Page Complete Time (PCT) is a SPA2 metric for base pages only. PCT calculates the point in time when the browser has finished loading all visual content on the page, regardless of whether the content is in or outside the viewport.
Troubleshoot SPA Monitoring

See the sections below to learn how to troubleshoot issues with monitoring SPAs.

- **Problems With Dynamically Loading the JavaScript Agent**

**Problems With Dynamically Loading the JavaScript Agent**

Because Angular allows you to dynamically load modules with AppdInitServlet, you may be tempted to dynamically load `adrum.js`. The problem with dynamically loading `adrum.js`, however, is that this could cause a race condition between the time `adrum.js` is evaluated and the time `angular.js` is bootstrapped. This race condition can result in JavaScript Agent missing virtual pages or XHR events.

You can avoid this race condition by doing the following:

- Loading Angular first and then immediately loading the JavaScript Agent. See Load Angular and the JavaScript Agent.
- **Bootstrap Angular** after you have loaded the JavaScript Agent.

**Load Angular and the JavaScript Agent**

The key here is simply to make sure that you load `angular.js` first and then immediately load `adrum.js` as shown below.

```
<script src="angular.js"></script>
<script src="adrum.js"></script>
```

**Bootstrap Angular**

*Manual (Recommended)*

After you have loaded `adrum.js`, you can then bootstrap Angular through manual initialization. You should load both `angular.js` and `adrum.js` as soon as possible, so that the JavaScript Agent can get more accurate metrics about the page load and the HTML DOM Ready /Build time. The examples below follow the recommended procedure of first loading `angular.js`, then `adrum.js`, and then bootstrapping `angular.js`
<!doctype html>
<html>
<head>
    <script src="http://code.angularjs.org/snapshot/angular.js"></script>
    <script charset='UTF-8'>
        window['adrum-start-time'] = new Date().getTime();
        (function(config){
            config.appKey = '<EUM_APP_KEY>';
            config.adrumExtUrlHttp = 'http://cdn.appdynamics.com';
            config.adrumExtUrlHttps = 'https://cdn.appdynamics.com';
            config.beaconUrlHttp = 'http://col.eum-appdynamics.com';
            config.beaconUrlHttps = 'https://col.eum-appdynamics.com';
            config.xd = {enable : false};
        })(window['adrum-config'] || (window['adrum-config'] = {}));
    </script>
    <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
</head>
<body>
    <div ng-controller="TestController">
        It's {{currentTime}}!
    </div>
    <script>
        angular.module('angularApp', [])
            .controller('TestController', ['$scope', function ($scope) {
                $scope.currentTime = new Date().toLocaleString();
            }]);
        angular.element(document).ready(function() {
            angular.bootstrap(document, ['angularApp']);
        });
    </script>
</body>
</html>
<!doctype html>
<html>
<head>
  <script src="http://code.angularjs.org/snapshot/angular.js"></script>
  <script charset='UTF-8'>
    window['adrum-start-time'] = new Date().getTime();
    (function(config){
      config.appKey = '<EUM_APP_KEY>';
      config.adrumExtUrlHttp = 'http://<your-cdn.com>';
      config.adrumExtUrlHttps = 'https://<your-cdn.com>';
      config.beaconUrlHttp = 'http://col.eum-appdynamics.com';
      config.beaconUrlHttps = 'https://col.eum-appdynamics.com';
      config.xd = {enable : false};
    })(window['adrum-config'] || (window['adrum-config'] = {}));
  </script>
  <script src='//<your-cdn.com>/adrum/adrum.js' type='text/javascript' charset='UTF-8'></script>
</head>
<body>
  <div ng-controller="TestController">
    It's {{currentTime}}!
  </div>
  <script>
    angular.module('angularApp', [])
      .controller('TestController', ['$scope', function($scope) {
        $scope.currentTime = new Date().toLocaleString();
      }]);
    angular.element(document).ready(function() {
      angular.bootstrap(document, ['angularApp']);
    });
  </script>
</body>
</html>
<!doctype html>
<html>
<head>
  <script src="http://code.angularjs.org/snapshot/angular.js"></script>
  <script>
    // Assumes that you downloaded the JS Agent and copied it to the 'js' directory of your app root.
    (function(config){
      config.appKey = '<EUM_APP_KEY>';
      config.adrumExtUrlHttp = 'http://cdn.appdynamics.com';
      config.adrumExtUrlHttps = 'https://cdn.appdynamics.com';
      config.beaconUrlHttp = 'http://col.eum-appdynamics.com';
      config.beaconUrlHttps = 'https://col.eum-appdynamics.com';
      config.xd = {enable : false};
    })(window['adrum-config'] || (window['adrum-config'] = {}));
  </script>
  <script src='//<your-cdn.com>/adrum/adrum.js' type='text/javascript' charset='UTF-8'/>
</head>
<body>
  <div ng-controller="TestController">
    It's {{currentTime}}!
  </div>
  <script>
    angular.module('angularApp', [])
      .controller('TestController', ['$scope', function ($scope) {
        $scope.currentTime = new Date().toLocaleString();
      }]);
    angular.element(document).ready(function() {
      angular.bootstrap(document, ['angularApp']);
    });
  </script>
</body>
</html>
Deferred (Optional)

If manual bootstrap is not possible, you can use deferred bootstrap. What this means is that you pause the bootstrap until `adrum.js` is successfully loaded, at which point, you resume the bootstrapping with `angular.resumeBootstrap()` as shown in the example below.
<html>
<head>
    <script src="http://code.angularjs.org/snapshot/angular.js"></script>
    <script>
        // This defers the bootstrap.
        window.name = 'NG_DEFER_BOOTSTRAP!';
        angular.module('angularApp', [])
            .controller('TestController', ['$scope', function ($scope) {
                $scope.currentTime = new Date().toLocaleString();
            }]);
        angular.element(document).ready(function() {
            angular.bootstrap(document, ['angularApp']);
        });
    </script>
    // For simplicity, we're just showing the injection snippet using the AppDynamics CDN.
    <script charset='UTF-8'>
        window['adrum-start-time'] = new Date().getTime();
        (function(config) {
            config.appKey = '<EUM_APP_KEY>';
            config.adrumExtUrlHttp = 'http://cdn.appdynamics.com';
            config.adrumExtUrlHttps = 'https://cdn.appdynamics.com';
            config.beaconUrlHttp = 'http://col.eum-appdynamics.com';
            config.beaconUrlHttps = 'https://col.eum-appdynamics.com';
            config.xd = {enable: false};
            })(window['adrum-config'] || (window['adrum-config'] = {}));
    </script>
    <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
</head>
<body>
    <div ng-controller="TestController">
        It's {{currentTime}}!
    </div>
    <script>
        // Resume the bootstrapping.
        angular.resumeBootstrap()
    </script>
</body>
</html>

Manually Initializing the JavaScript Agent (Optional)
If `adrum.js` loads successfully, but does not initialize, you can manually initialize it by calling
`ADRUM.ng.ngMonitor.init()` as shown below. This will force the JavaScript Agent to initialize even after the Angular bootstrap.

```html
<html>
<head>
    <script src="http://code.angularjs.org/snapshot/angular.js"></script>
    <script>
        // This defers the bootstrap.
        window.name = 'NG_DEFER_BOOTSTRAP!';
        angular.module('angularApp', [])
            .controller('TestController', ['$scope', function ($scope) {
                $scope.currentTime = new Date().toLocaleString();
            }]);
        angular.element(document).ready(function() {
            angular.bootstrap(document, ['angularApp']);
        });
    </script>
</head>
<body>
    <!-- For simplicity, we're just showing the injection snippet using the AppDynamics CDN. -->
    <script charset='UTF-8'>
        window['adrum-start-time'] = new Date().getTime();
        (function(config) {
            config.appKey = '<EUM_APP_KEY>';
            config.adrumExtUrlHttp = 'http://cdn.appdynamics.com';
            config.adrumExtUrlHttps = 'https://cdn.appdynamics.com';
            config.beaconUrlHttp = 'http://col.eum-appdynamics.com';
            config.beaconUrlHttps = 'https://col.eum-appdynamics.com';

            config.xd = {enable : false};
        })(window['adrum-config'] || (window['adrum-config'] = {}));
    </script>
    <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'></script>
</body>
</html>
```
<body>
    <div ng-controller="TestController">
        It's {{currentTime}}!
    </div>
    <script>
        // Resume the bootstrapping.
        angular.resumeBootstrap();
        ADRUM.ng.ngMonitor.init();
    </script>
</body>
In some cases, you may be able to place `ADRUM.ng.ngMonitor.init()` before `angular.resumeBootstrap()`.
Overview of the Controller UI for Browser RUM

On this page:

- Access a Browser RUM Application
- Browser App Dashboard
- Sessions
- Pages & Ajax Requests
- Analyze
- Enable Browser RUM

Browser RUM presents information in these views:

- An **Overview** dashboard, with sections for:
  - widgets for graphic display of common metrics
  - map-based performance display
  - snapshots of individual requests
  - usage statistics by browser and device/platform
- Unified data for entire sessions, following an individual user's path as they navigate through your site
- Detailed information on the performance of individual pages, as
  - aggregated lists of page, Ajax, iframe, and virtual page request types
  - multi-faceted data from a complete data store

Access a Browser RUM Application

To see the data for your app in the Controller UI, you must first open your application:

1. In the top tab bar, select **User Experience**. The list of instrumented Browser and Mobile Apps is displayed.
2. Make sure the **Browser App** tab is selected.
3. Double-click on the app you are interested in. The monitoring UI opens.

Browser App Dashboard

This view is good for getting a high-level understanding of how your app is performing overall.

**Overview**

The **Overview** tab is made up of a set of widgets showing common usage metrics. You can add, delete, move, and resize widgets as you wish. If you see a metric that interests you, click through to the main view. If you are using Synthetic, data from both RUM and Synthetic (separated by color) show up in some of the widgets.
The **Geo Dashboard** view provides high-level insight into how your application is performing across the world for your users. The size of a dot indicates the number of page views from that region, and the color indicates the average page load End User Response time in that region (red is slow). You can click to drill down to areas of specific interest. You can also see the same information presented in tabular form by clicking the grid icon in the upper left of the panel.

If you are using Synthetic, you can also select to see either RUM or Synthetic data displayed here. By default, the **Geo Dashboard** displays key performance metrics by country, but you can view performance metrics by region by checking the **Show regions in Global View**.
For more information, see Browser App Dashboard.

The option View Dark Mode has been removed from the Geo Dashboard.

**Browser Snapshots**

The Browser Snapshot view provides access to detailed information for individual requests. The list includes both periodic snapshots of requests operating within normal boundaries and problem snapshots of requests that have exceeded one or more configurable performance criteria. Double-clicking a specific item takes you to a detailed graphical representation of the execution flow of that request and other data associated with it.
Usage Stats

The **Usage Stats** view presents aggregated page load usage data based on the browser type and device/platform employed by your users. The view also breaks down performance by type and usage by country.

For more information, see **Browser Snapsots**.

For more information, see **Browser App Dashboard**.
Sessions

In most cases, users interact with your web application over a series of real (or virtual) pages. Sessions allow you to track your users’ interactions across time, as they navigate through an entire engagement period with your app. Using Browser RUM Sessions, you can analyze session results from all requests, as stored in the AppDynamics Events Service. Use the Records tab to focus on pertinent data and the Charts tab to view a dashboard of useful graphics widgets. Or follow a session from page to page using the Session Detail dialog.

For more information, see Browser RUM Sessions.

Pages & Ajax Requests

This view is good for tracking the performance of individual pages and components as well as understanding any issues that may be emerging. The Pages & Ajax Requests view shows you an aggregated view of how each of your page, Ajax request, iframe, and virtual page types are performing over time. You can look at All Pages for an overall sense or select a Top Pages view to see the worst performing pages sorted by common metrics like Page views with JavaScript Errors and First Byte Time. And you can drill down to a graphical dashboard showing a wide range of charted performances characteristics for any specific request type.
The Pages & Ajax Requests View

Analyze

The Analyze view allows you to query a complete repository of all traffic data that Browser RUM has collected in the past two weeks and display that data in multiple ways.

For more information, see Browser RUM Analyze
Enable Browser RUM

Browser RUM requires a separate license and must be enabled before it is available for use.

For information about licensing, see Browser RUM Licenses.

For information on enabling or disabling Web User Experience, see Set Up and Access Browser RUM.
Browser RUM Sessions

On this page:
- What is a Browser RUM Session?
- Why RUM Sessions Are Important
- Use Cases
- How EUM Captures RUM Sessions
- Session Persistence
- Privacy/Security of Session Data
- Browser RUM Support for Sessions
- Session Limits
- Sessions UI

This section provides both an overview as well as pertinent details of Browser RUM sessions. We also describe the Session UI that you’ll use to view session data.

**What is a Browser RUM Session?**

A Browser RUM session is a collection of chronological events for a particular end user from the start of a session to a configurable period of inactivity. An event represents an activity occurring at the moment in time that can be recorded, aggregated, and correlated at certain levels. In Browser RUM, events are limited to page loads, virtual page loads, iFrames, and Ajax requests.

Only Ajax requests configured to be published to the Events Service are included in sessions.

For example, when a user purchases product from an e-commerce site, the session might capture the page/virtual page loads, iFrames, or Ajax requests for the user logging in (start), selecting and paying for products, and logging out (end).

The figure below describes how the user’s actions cause events that are tracked in the session.
Why RUM Sessions Are Important

You can think of sessions as a time-based context to analyze a user's experience interacting with an application. By examining RUM sessions, you can understand how your applications are performing and how users are interacting with them. This enables you to better manage and improve your application, whether that means modifying the UI or optimizing performance.

Use Cases

In some use cases, you can use RUM sessions to analyze a user's or a group's behavior. You analyze a user's behavior based on the user's unique ID across multiple sessions in what is called sniping. In addition, you can query for a segment of users with similar behavior, such as visiting a certain specific page or using a particular device. This technique of grouping users with similar behavior is called bucketing. Browser RUM sessions allow you to query users on a wide array of criteria such as the type of browser, device, location, landing page, IP address, and more.

How EUM Captures RUM Sessions

The JavaScript Agent loads a temporary GUID into the browser local storage and then transmits beacons to the EUM Cloud or EUM Server that include this GUID. We can then use the GUID to track the user’s device over the duration of one or more sessions. The EUM Cloud then publishes these events into a session record stored in the Events Service.

Session Persistence

Sessions persist until a period of user inactivity that you configure in the controller. Sessions can also end when the GUID is removed from local storage, which can happen for a number of reasons. For example, the user could clear local storage or the browser is configured to clear the local storage when the browser is closed.

Maximum Session Time

We don’t technically limit the time of a session. Instead, we restrict the maximum number of events to 1000. You can, however, configure the maximum time for each event by configuring session monitoring timeouts and then derive the maximum time for a session.
by multiplying the configured session inactivity timeout by 1000. For example, if you configure the session inactivity timeout for each event to be five minutes, the maximum session time would be 5000 minutes. Of course, in this case, most sessions would not last 5000 minutes because end users would trigger events well before the maximum time of five minutes.

**Single Page / Multiple Page Sessions**

When the browser cannot write the GUID to local storage, Browser RUM cannot track the user's device over one or more sessions. Instead, Browser RUM uses the temporary GUID in memory to create one-page or multiple virtual page sessions. Virtual pages are simply those sections of the DOM that are dynamically created and loaded into the current page.

**Privacy/Security of Session Data**

We do not perform browser or device fingerprinting. By default, we do not store user IP addresses. If you configure the Controller to store IP addresses, then the IP addresses are stored for up to 14 days.

**Browser RUM Support for Sessions**

**JavaScript Agent Version Requirement for Sessions**

You are required to use the JavaScript Agent 4.2+. Older versions of the JavaScript Agent do not support sessions.

**Browser Requirements for Sessions**

To use Browser RUM sessions, your browser is required to support the following:

- cross-origin resource sharing (CORS) for beacons
- local storage for multiple-page sessions (single-page / multiple virtual page sessions don't need local storage)

Browser RUM sessions do not support beacons implemented with GIFs.

**Session Limits**

The table below lists the limits for session operations per account.

<table>
<thead>
<tr>
<th>Session Operations</th>
<th>Maximum Per Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stored session resource snapshots</td>
<td>10k per minute</td>
</tr>
<tr>
<td>Pending uploads of resource timing</td>
<td>100k</td>
</tr>
<tr>
<td>Published session beacons</td>
<td>10k per minute</td>
</tr>
<tr>
<td>Tracked sessions</td>
<td>5k per minute</td>
</tr>
</tbody>
</table>

After the pending uploads of resource timing snapshots reach 100k, no further resource-timing snapshots are queued.

**Sessions UI**

As in Browser RUM Analyze, the main Sessions page is made up of two tabs:

- Records
- Charts

**Records**

The Records tab lets you scan individual sessions and allows you to filter and sort to get exactly the data in which you are interested.
Click **View Details** or double-click an item to see the information for a specific session. The sequence of page views for the session is shown on the left side of the screen. Select a specific page view to see detailed information, including a page load waterfall, tabular details for resources, and, if your backend is instrumented with AppDynamics, correlated server-side business transactions.

**Charts**

The **Charts** tab provides you with a set of predefined widgets of the data set you have created plus a custom widget builder. As with the **Charts** tab of the Analyze UI, you can delete, re-add, resize, and drag-and-drop to move the widgets.
Pages & Ajax Requests

On this page:
- Access Pages & Ajax Requests
- What is a Page?
- Types of Pages
- Pages & Ajax Requests
- How the All Pages List is Organized
- Access Top Pages
- Page Limits

Related pages:
- Page, Ajax, and Iframe Dashboards
- Configure Page Identification and Naming

The Pages & Ajax Requests page gives you detailed information on how your pages, Ajax requests, iframes, and virtual pages perform over time.

Access Pages & Ajax Requests

1. Open the browser application in which you are interested.
2. On the left navigation bar, select Pages & Ajax Requests.

What is a Page?

In Browser RUM, a page represents an individual source for the information that an end-user sees in one browser window.

Types of Pages

A base page is the core HTML page.

A base page may also include one or more iframes, which can be nested.

A base page or an iframe can also make one or more Ajax requests (using XMLHttpRequest or the Fetch API) to fetch data to display on the Web page.

You can collect Browser RUM metrics for base pages, iframes, Ajax requests, and virtual pages.
Each base page, iframe, Ajax request, and virtual page type is assigned a unique name, which you can configure. See Configure Page Identification and Naming for information on how to do that.

**Pages & Ajax Requests**

The Pages & Ajax Requests page has two options: All Pages and Top Pages, selected using the dropdown in the upper right. All Pages displays a list showing a high-level summary of all the monitored types, along with their key performance indicators. Top Pages displays the ten worst performing items grouped by common metrics: Requests per Minute, Page Render Time, and so forth. Use this option for a quick start to troubleshooting.

**How the All Pages List is Organized**

The All Pages list displays each monitored base page, iframe, Ajax request, and virtual page type. Click a column header to sort the pages based on the column's metric. To view a dashboard for a specific page item in the list, select it and click View Dashboard or just double-click the page. See Page, Ajax, and Iframe Dashboards for more information.

To filter the types of pages displayed in the list, select the type at the top of the list. Check With Load to see only pages currently reporting load.

**More Actions Menu**

Using the More Actions menu, you can select a page in the list and make changes to it, including directing the agent to ignore the page and stop reporting metrics for it.

**Access Top Pages**

Click the View dropdown on the upper-right side of the page and select Top Pages as a shortcut to troubleshooting the ten worst performing pages regarding various metrics.

**Page Limits**

There is a limit of 500 total base pages (including virtual pages) + iframes and 500 Ajax calls that can be individually tracked per application. If your usage exceeds these limits, the Controller begins to drop metrics. If your installation is approaching these limits, you can modify how your metrics are collected by:

- Limiting the number of pages you instrument. If you are using manual injection, remove the JavaScript Agent from pages that are less important. See Inject the JavaScript Agent for more on injection types. If you are using automatic injection, create request match rules and request exclude rules to restrict injection to pages that meet certain criteria. See To Create Match Rules for Automatic Injection.
- Using custom naming rules to group similar pages together. See Configure Page Identification and Naming.

If you have hit your page limit, you need to delete the affected old page names before any new page names can be added to the database. Doing so also deletes any accumulated metrics for those page names.
Page, Ajax, and Iframe Dashboards

On this page:
- Access These Dashboards

Related pages:
- Page and IFrame Dashboards
- Ajax Dashboard
- The Pages & Ajax Requests View

Dashboards provide quick access to graphic representations of metrics for pages (including virtual pages), iframes, and Ajax request types. Each page, iframe, and Ajax request type has a dashboard.

Access These Dashboards

To view a dashboard for a page, iframe or Ajax request type:

1. Open the browser application in which you are interested.
2. On the left navigation bar of your application, select Pages & AJAX Requests.
3. From the list, select the page, iframe or Ajax request type in which you are interested.
4. Either double-click the item or select the item and click Details.
Page and IFrame Dashboards

On this page:

- Page Dashboard Summary Section
- Timing Breakdown
- Overall Performance
- Key Performance Times / Load
- Server Connect Time Breakdown / Response Available Time
- Server Time / HTML Download Time / Related Business Transactions
- DOM Building Time / Resource Fetch Time / Page Resources Requested

Related pages:

- Ajax Dashboard

Page and iframe dashboards are divided into six areas:

- A summary of key performance indicators. To see browser snapshots associated with these instances of this page type, open the Browser Snapshots tab. To move to the Analyze view, click View Analyze.
- A waterfall graph of the entire load sequence. To see details for each set of data, use the Trends/Details checkboxes.
- Four sections of Trends/Details across time for the main performance categories:
  - Overall performance
  - Time between the request and the first byte of the response
  - Time taken by the server to process the request through the completion of the HTML download for the item
  - Time taken to process and render the item, including any external resources, in the browser
- Detailed information on the performance of Ajax requests and iframes for this item.

Page Dashboard Summary Section

This section gives you a quick overview of the item's performance over time.

Key performance indicators—End User Response Time, Load, Cache Hits, and Page Views with JS errors—across the period selected in the timeframe dropdown from the upper-right side of the Controller UI are displayed across the top of the summary area. Cache Hits indicates a resource fetched from a cache, such as a CDN, rather than from the source. This metric is only reported when there is a correlated AppDynamic agent on the server-side source.
Timing Breakdown

The waterfall graph shown below displays the average times needed for each aspect of the page load process. For more information on what each of the metrics measures, hover over its name on the left. A popup appears with a definition. For more detailed information, see Browser RUM Metrics.
To see detailed breakouts of the data behind the graph, check the Trend/Detail checkbox by the data group in which you are interested. To turn the details off, uncheck the checkbox.

Some metrics—for example, TCP Connection Time—only appear if they have a non-zero value.

**Overall Performance**

This section displays detailed trend graphs of key performance metrics measured across the selected timeframe. To see detailed information for a particular moment, hover over the graph and a popup with that information appears. To see any of the listed total metrics in the context of the metric browser, click the desired value (shown in link blue) on the left side of the panel. The Metric Browser appears, with that metric displayed. You can then use the Metric Browser to compare other related values in one display.

**Key Performance Times / Load**

This section displays detailed trend graphs of key performance indicators and the request load across the selected timeframe. They measure:

- the time between the user's request and the completion of the page load of the response.
- the time between the user's request and the DOM being loaded.
- the time between the user's request and the browser receiving the first response byte.
- the number of requests per minute.

**Server Connect Time Breakdown / Response Available Time**

This section displays detailed trend graphs of initial server connection metrics measured across the selected timeframe. They measure:

- the time the user's request takes in negotiating its initial connection with the server, which may include the DNS, TCP Connect, and SSL/TLS time. The **Total Server Connect** value is always displayed.
- the time between that initial connection and the time the first byte of information begins to arrive at the browser.

**Server Time / HTML Download Time / Related Business Transactions**

This section displays detailed graphs for the time spent acquiring the page data:

- the time the request takes on the server.
- the time it takes to complete the download of the HTML to the page.
- If the request is correlated with a server-side app agent, related business transactions on the server are displayed. Cross-app correlation is supported.

**DOM Building Time / Resource Fetch Time / Page Resources Requested**

This section displays detailed graphs for the time spent creating the page and getting resources. They measure:

- the time required by the browser to create the DOM from the end of the HTML download.
- the time required to fetch any external resources. For example, the results of a third-party Ajax request.

The **Page Resources Requested** section displays detailed graphs of when in the page load cycle individual external—first and third-party—resources are fetched, and how much time is taken to fetch them, based on the selected timeframe. They measure:
• average time and load associated with that resource
• whether the request is blocking or non-blocking
• request and response time per resource request
• type—iframe or Ajax—of the resource

To see the dashboard for any of the listed resources, click the name.
Ajax Dashboard

On this page:
- Ajax Dashboard Summary Section
- Timing Breakdown
- Overall Performance
- Ajax Response Download and Ajax Callback Execution Time

Related pages:
- Page and IFrame Dashboards

The Ajax Dashboard is divided into three areas:

- A summary, with a waterfall graph of the entire load sequence. To see details for each set of data, use the Trends/Details checkboxes.
- Two sections of Trends/Details across time for the main performance categories:
  - Overall performance
  - Time is taken by the server to process the request through the browser’s incorporation of the data into the HTML document

Ajax Dashboard Summary Section

This section gives you a quick overview of the item’s performance over time.

Key performance indicators—End User Response Time, Load, Cache Hits and Views with Errors—across the period selected in the timeframe dropdown from the upper-right side of the GUI—are displayed across the top of the summary area.

A waterfall graph displays the average times needed for each aspect of the Ajax request load process.

For more information on what each of the metrics measures, hover over its name on the left side of the graph. A popup appears with a definition. See the graphic above for an illustration. For more detailed information, see Browser RUM Metrics.

To see detailed breakouts of the data behind the graph, check the Trend/Detail checkbox by the data group in which you are interested.

Timing Breakdown

A waterfall graph shown below displays the average times needed for each aspect of the Ajax request. For more information on what each of the metrics measures, hover over its name on the left. A popup appears with a definition. For more detailed information, see Browser RUM Metrics.

To see detailed breakouts of the data behind the graph, check the Trend/Detail checkbox by the data group in which you are interested. To turn off details, uncheck the checkbox.

Overall Performance
This section displays detailed trend graphs of key performance metrics, load requests, and Ajax request errors measured across the selected timeframe. To see detailed information for a particular moment, hover over the graph and a popup with that information appears. To see any of the listed total metrics in the context of the metric browser, click the desired value (shown in link-blue) on the left side of the panel. The metric browser appears, with that metric displayed. You can then use the metric browser to compare other related values in one display.

### Ajax Response Download and Ajax Callback Execution Time

This section displays detailed trend graphs of the Ajax response download time and the Ajax callback execution time. To see detailed information for a particular moment, hover over the graph to view a popup.

These metrics measure the total time for processing all server-side business transactions for this item.

- the total time for the browser to completely download all of the Ajax responses
- the time for the browser to complete any Ajax callbacks
Browser RUM Analyze

On this page:

- Access the Browser Analyze View
- Records
- Charts
- Request Number Differences in Browser Analyze and Pages & AJAX Requests
- Browser Analyze versus Browser Request Analytics

Related pages:

- Analytics Browser Requests Data
- Analytics Browser Sessions Data

Browser RUM collects data on every load event that you have instrumented, and it also takes detailed snapshots periodically and when performance issues are detected. Over time, the load event metrics are rolled up based on averages and percentiles. Sometimes you want to see results based on all of the data.

With Browser Analyze, every event is collected and stored for a license specified period in the AppDynamics Platform Events Service. Using the Analyze tab you can see results based on this cumulative data.

The Analyze view is made up of two tabs:

- Records
- Charts

**Access the Browser Analyze View**

1. Open the application in which you are interested.
2. On the left navigation bar, select Analyze.

**Records**

The Records tab lets you scan individual events and filter and sort to get exactly the data set in which you are interested.
After you click View Details, you see the details of a particular request as a page with a browser session in the Session Details dialog.

The Charts tab provides you with a set of predefined widgets that offer visualizations of the data set you have created.

You can quickly see how pages, browsers, and geographic location (i.e., dimensions) affect the end user response time (EURT). You can also delete, re-add, resize, and drag-and-drop to move all of the widgets.

The time series widgets such as **Response Time** and **End User Response Time** use median
Request Number Differences in Browser Analyze and Pages & AJAX Requests

The number of requests for a page over a period of time should be the same in Pages & AJAX Requests and Browser Analyze, but they may vary for the following reasons:

- If there are fewer requests given in Browser Analyze, the Events Service, which is responsible for the data in Browser Analyze and Pages & AJAX Requests, has probably not received all of the records yet. To verify this, compare the time of the last request for the page in Browser Analyze and Pages & AJAX Requests. The latter will most likely have a later timestamp.
- If there are more requests given in Browser Analyze, it's possible that the reason is because error records are included in the total number of requests in Pages & AJAX Requests, but listed separately in Browser Analyze. The errors are listed separately in Browser Analyze to enable you to investigate and understand the cause of the error.

Browser Analyze versus Browser Request Analytics

The data shown on the Analyze page is processed and stored by the AppDynamics Platform Events Service. The Charts tab displays a set number of widely used chart types to let you explore your application's performance. A separate product, AppDynamics Application Analytics, has a component called Browser Analytics. This component is based on the same Events Service and uses the same data, but it offers additional capabilities, including:

- additional predefined widgets, such as the funnel widget
- ADQL for searching the data
- creating custom widgets
- manipulating multiple dashboard types
- longer retention time for data storage

Browser Analytics requires a license separate from the Browser RUM license.

To reduce noise in data stored in the Event Service, Ajax calls from the following tracker domains are no longer published to the Events Service:

- .mixpanel.com
- .google-analytics.com
- .altocloud.com
- .optimizely.com
- inspectlet.com
Configure the Controller UI for Browser RUM

You can enable or disable Browser RUM with the Browser Monitoring toggle.

You can manage how Browser RUM information is displayed in the Controller UI, including:

- The display names for your pages, Ajax requests, and iframes
- The display names for your virtual pages
- The errors that should be shown in the UI, and the ones that should not be shown

You can also configure:

- The thresholds for slow, very slow, and stalled transactions
- When browser snapshots should be taken
- Percentile levels you would like to display, if any
- Which Ajax requests should be sent to the Event Service
- The session timeout period
- Whether to store request IP addresses or not

In addition, you can do the following:

- Choose a hosting option for the JavaScript Agent

To configure Browser RUM from the Controller UI, your user account must belong to a role that has the Configure EUM permission. See End User Monitoring Permissions for more information.

Access the Browser RUM Instrumentation Configuration

To access the instrumentation configuration for a browser app:

1. Open the browser application in which you are interested.
2. From the left-hand navigation menu, click Configuration.
3. From the Configuration page, click Instrumentation >.
Configure the Primary Metric

On this page:
- Available Metrics
- Primary Metric Visibility

Related pages:
- Browser RUM Metrics

The primary metric is only available for Controller version 4.5.13 or later and JS Agent version 4.5.13 or later.

The primary metric is a way to customize your browser application widgets. The default primary metric is set to End User Response Time (EURT), but you can choose between EURT and Visually Complete Time (VCT). Once the primary metric is configured, the various dashboards, summaries, and charts will automatically update to show trending data for that configured metric. This allows you to view the EUM event timing that is most relevant to your business needs.

Configure Primary Metric

<table>
<thead>
<tr>
<th>Setup</th>
<th>End User Response Time</th>
<th>Visually Complete Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The primary metric is only available for Controller version 4.5.13 or later and JS Agent version 4.5.13 or later.

To configure the primary metric, go to Configuration > Instrumentation > Settings.

Available Metrics

You can set the primary metric to EURT or VCT. The default primary metric is EURT. EURT tracks the total amount of time for all content on a web page to be loaded. VCT tracks the total amount of time for all visual content to be loaded in the viewport. To see how EURT and VCT are calculated for SPA pages, see SPA2 Metrics.
Primary Metric Visibility

The primary metric is visible across several dashboards, charts, and summaries. The following screenshots are examples of primary metric visibility when configured to VCT.

Browser App Overview

Geo Dashboard
Browser Snapshots

Pages & AJAX Requests
Configure Page Identification and Naming

**On this page:**
- Access Exclude/Include Rules
- Evaluation Logic for Exclude/Include Rules
- Default Page Naming Rules
- Custom Include Rules
- Custom Page Exclude Rules

**Related pages:**
- The Pages & Ajax Requests View
- Configure Virtual Page Naming
- Page, Ajax, and Iframe Dashboards
- Set Custom Page Names

You can configure the display names by which various pages and iframes are referred to and sorted in Controller lists and dashboards.

You can:
- Use the AppDynamics default naming rule, which you can leave as is or modify.
- Create custom include rules to override the default convention.
- Create custom exclude rules to exclude from monitoring pages that meet certain criteria.
- Disable the default naming configuration and use only your custom include rule(s).

In this topic, the term "pages" includes iframes and base pages.

No matter how the page is named, AppDynamics always reports the page name in lowercase.

**Access Exclude/Include Rules**

1. Open the browser application in which you are interested.
2. On the left navigation bar, click **Configuration**.
3. Click **Instrumentation** >
4. From the **Base Page and iFrames** tab, you can define include/exclude rules or modify the **Default Naming Configuration**.

To save any configuration changes, click **Save**.

**Evaluation Logic for Exclude/Include Rules**

This is the order in which AppDynamics evaluates the page naming rules.
Default Page Naming Rules

If you enable the default naming configuration and do not modify it, AppDynamics identifies and names your pages using the first two segments of the page URL.

You can modify the default configuration by double-clicking Default Naming Configuration in the Include Rules section. This opens the Include Rule popup, where you can select the dropdown list using parts of the URL to include the protocol or domain in the name, use different segments of the URL, or include query parameters or anchors in the name.
You can also choose the dropdown using a Regex match on the URL to match page names by running a regular expression.

If you do not want to use the default convention at all, disable it by clearing the Enabled checkbox. In this case, you must configure at least one custom page naming rule so that AppDynamics can identify and name pages.

Custom Include Rules

You can create custom include rules for identifying and naming pages.

To create a custom include naming rule, from the Include Rules section, click Add. Then you use the same Include Rule popup to configure the custom include rule to identify and name the page.

The order of the include rules in the list determines the priority of the rules: The rules at the top are evaluated first, and the rules at the bottom are evaluated last. You cannot change the priority of Default Naming Configuration, but you can modify or disable it.

**Guidelines for Using Regular Expressions in Include Rules**

When using regular expressions to match URLs in your include rules, you should note the following:

- URL strings are case-sensitive. So, although the page names displayed in Pages & Ajax Requests are converted to lowercase, your regular expressions still need to match the case used in URLs that your include rule are trying to capture.
- Your regular expression should match the entire URL, from beginning to end, not just a section; otherwise, the rule will not match. This differs from using regular expressions in custom match rules for naming transactions, which only need to match sections of the URL.
- Beacons for older browsers (primarily Internet Explorer 6-8) are sent using image requests, and therefore, have an inherent length limitation. To manage this limitation, Browser RUM does not support URLs longer than 180 characters, page names longer than 50 characters, and user data longer than 128 characters for these browsers.

**Example Include Rule**
Suppose you have multiple pages that include the string `search/r/region` in their URLs, such as `search/r/region01`, `search/r/region23`, etc., and you want to name all the pages from that set as a single-page named `search/r/region`. By selecting the dropdown using a Regex match on the URL, you can enter a regular expression to remove the domain name and the number at the end of the URL, grouping all your `/search/r/region` URLs into one set. Because all the URLs contain `search/r/region`, AppDynamics now collects information for them all under the single-page name `search/r/region`. Otherwise, AppDynamics uses the default page naming rule or a rule with higher priority.

### Include Rule

- **Enabled**
- **Rule Name**: `search/r/region`
- **Criteria**: This Include Rule applies to any URL that contains `search/r/region`
- **Name Pages**: Using a Regex match on the URL
- **Use regex on the URL**: `(.*)search/r/region([^/]*).*` and use groups 2

### Custom Page Exclude Rules

You can configure custom exclude rules for pages in the same way you configure include rules. Any page with a URL matching the configuration is excluded from monitoring.

### Exclude Rule

- **Enabled**
- **Rule Name**: `Exclude Rule 1`
- **Criteria**: This Exclude Rule applies to any URL that contains `appdynamics`
Configure Page Naming for Ajax Requests

You can configure the display names by which Ajax requests are referred to and sorted in controller lists and dashboards.

Access Page Naming Rules for Ajax Requests

1. Open the browser application in which you are interested.
2. On the left navigation bar, click Configuration.
3. Click Instrumentation >.
4. Select the AJAX > Monitor tab.

Naming Rules for Ajax Requests

The logic for naming Ajax requests is nearly identical to that for naming virtual pages in that they can use anchors (the part of the URL after the #) to distinguish among Ajax requests. In these cases, using the What part of anchor should be used in page name section allows you to specify which Ajax request is being accessed correctly. The difference in naming rules for Ajax requests is that you can also specify the HTTP method(s) to filter in the rule.

For example, the Ajax naming include rule below specifies the HTTP methods POST and PUT and is configured to show the HTTP method in the results displayed in the Controller UI:
You can also create exclude rules that specify which Ajax requests to filter based on the specified HTTP method(s) as shown here:
Configure Which Ajax Requests Are Sent to the Events Service

Modern web pages often include a large number of Ajax requests. Not all of these Ajax requests may be equally important to monitor, but they all equally affect license entitlements and restrictions. Thus, you may want to configure which Ajax requests are sent to the Events Service to manage the impact on your overall Events Service usage.

To do this, you can create rules to specify the requests to be sent to the Events Service, either by excluding a request entirely, including a particular request or a sample of that request types by percentage or by just allowing the request to be sent. In general, the behavior follows this pattern:

- If no rules are specified, no data on Ajax requests is sent.
- If exclude rules are specified, and an Ajax request satisfies a rule, that data is not sent.
- If include rules are specified, any Ajax request that satisfies a rule is sent, based on sampling defined by the percentage indicated in the rule.
- If both include and exclude rules are specified, an Ajax request that satisfies an include rule but does not satisfy an exclude rule is sent.

See Configure Page Identification and Naming for more information about default naming rules and how rules are evaluated.

Access Ajax Requests Rules

1. Open the browser application in which you are interested.
2. On the left navigation bar, select Configuration.
3. Click Instrumentation >.
4. Select the Ajax > Events Service tab.

Configure Exclude Rules

1. Click Add to create a new rule. The Exclude Rule popup appears.
2. Specify a display name for the rule.
3. Check Enabled to enable the rule.
4. Enter a string and select one of the options in the dropdown to match the URLs that you want to exclude from the Ajax requests.
5. Click OK to save the rule.

Configure Include Rules

1. From the Events Service Include Rules section, click Add to create a new rule. The Include Rule popup appears.
2. Give your rule a display name.
3. Check Enabled to enable the rule.
4. For Criteria, enter a string and select one of the options in the dropdown to match URLs for including Ajax requests.
5. Configure how to name pages:
   - Select one of the options from the dropdown to determine the part of the URL.
   - Select the path segment.
   - Enter the query string to use in the page name.
   - Select the part of the anchor (or none) to use in the page name.
6. Click Save.
Configure Virtual Page Naming

On this page:
- Access Virtual Page Naming Rules
- Virtual Page Naming Rules

Related pages:
- The Pages & Ajax Requests View
- Configure Page Identification and Naming
- Page, Ajax, and Iframe Dashboards
- Set Custom Page Names

Web applications built using Single Page Application (SPA) principles minimize network traffic by transferring to the browser itself much of the computing work for creating what the user sees. The initial page request downloads everything that is necessary for constructing all aspects of the application, with the possible exception of some data, including HTML partials and fragments, that may be fetched dynamically from the backend in response to user interaction.

The individual views that the user sees are known as virtual pages. Browser RUM supports virtual pages created using the AngularJS framework, sometimes known simply as Angular.

In AngularJS, a virtual page is an individual view, comprised of the rendered template of the current route in the context of the main layout file. You can configure how AngularJS virtual pages are referred to and sorted in UI lists and dashboards. For more information on using Browser RUM with virtual pages, see AngularJS Support.

Access Virtual Page Naming Rules

1. Open the browser application in which you are interested.
2. On the left navigation bar, click Configuration.
3. Click Instrumentation >.
4. Select the Virtual Pages tab.

Virtual Page Naming Rules

The logic for naming virtual pages is identical to that for naming regular pages, with one exception. Because AngularJS pages can use anchors (the part of the URL after the #) to distinguish among virtual pages, in these cases, using the What part of anchor should be used in page name section allows you to specify which virtual page is being accessed correctly.
Include Rule

Enabled

Rule Name
Example Include Rule for Virtual Pages

Criteria
This Include Rule applies to any URL that contains contact

Name Pages
using parts of the URL

Show Protocol (Ex: http, https, etc)
Show Domain (Ex: mywebsite.com)
  - Show Full Domain
  - Show Sub-domain

Path Segments
  - Don't use path segments
  - Use first 1 segments
  - Use last 1 segments
  - Use segment numbers

Query String Parameters to use in Page Name (Optional)

What part of anchor should be used in Page Name
  - Don't use the anchor
  - Use first 1 segments
  - Use last 1 segments
  - Use segment numbers

Cancel  OK
Configure JavaScript and Ajax Error Detection

On this page:
- Access Error Detection Rules
- Enable and Disable Browser RUM Error Detection
- Configure Rules to Ignore Errors based on Script or Error Message
- Configure Rules to Ignore Errors by Page
- Configure Rules to Ignore Errors by URL

Related pages:
- Set Up and Access Browser RUM
- Browser Snapshots

You can enable and disable reporting of JavaScript and Ajax request errors. When enabled, the Browser Monitoring UI reports Ajax request errors in the following places: in the Geo Dashboard, in the User Stats page and device dashboards, in the page list, and in browser snapshots.

Also, you can configure which errors are included in the error count by specifying the errors to "ignore".

You can specify errors to ignore:
- by the script and/or error message
- by page
- by URL

"Ignored errors" are not actually ignored. They are still tracked, but the error count in the places where error totals are reported on the user interface is not incremented.

Access Error Detection Rules

1. From the browser application you are interested in, click Configuration.
2. Click Instrumentation >.
3. Select the Errors tab.

Enable and Disable Browser RUM Error Detection

In the Errors tab:
- Check/clear the Enable JavaScript Error Capture checkbox to enable/disable JavaScript error display.
- Check/clear the Enable AJAX Request Error Capture check box to enable/disable Ajax error display.

When both checkboxes are clear, no JavaScript or Ajax request errors are displayed.

Even if capture is enabled globally, you can configure certain errors to be ignored so that they are not counted in the error totals.

Configure Rules to Ignore Errors based on Script or Error Message

You can configure the agent to ignore specific JavaScript errors that are identified by:
- a matching string pattern in the name of the script that generated the error
- the line number in the script
- a matching string pattern in the error message

You can specify one, two, or all three of these criteria. Configure more criteria to increase the granularity of which errors you ignore.

Adding Rules for Ignoring Errors

From the Ignore JavaScript Error Rules section, click Add to open the Ignore JavaScript Errors popup. From there, you can enable the rule, specify the script name, line number, and optionally, the error message.

For example, the following configuration in the Ignore JavaScript errors section where all three fields are specified means "Ignore all errors generated by line 27 of a script whose name starts with "Nightly" and whose error message contains the string "WARNING::"."
If the line number were not specified (e.g., set to 0), the configuration would mean "Ignore all errors generated by any line of a script whose name starts with "Nightly" and whose error message contains the string "WARNING::".

If neither the line number nor the error message field was specified, the configuration would mean "Ignore all errors generated by any line of a script whose name starts with "Nightly".

If the error message were the only field specified, the configuration would mean "Ignore all errors generated by any script when the error message contains the string "WARNING::".

**Modifying Rules**

To modify an existing ignore rule, you can either double-click the rule in the list or select the rule and then click **Edit** from the dropdown. To remove an ignore rule, select the rule in the list and then click **Delete** from the dropdown.

**Configure Rules to Ignore Errors by Page**

You can also ignore all errors generated by a specific page, iframe, or Ajax request.

From the **Ignore Errors from these Pages** section, click **Add** to open the **Ignore errors from specific Pages** popup, where you can create a rule for every page for which you want to ignore all errors.

To remove an ignore rule, select it in the list and click **Delete** from the dropdown list.

**Configure Rules to Ignore Errors by URL**

You can ignore all errors generated by a specific URL.

From the **Ignore Errors from these URLs** section, click **Add** to open the **Ignore errors from specific URLs** popup, where you can create a rule for every URL for which you want to ignore all errors.

To remove an ignore rule, select it in the list and click **Delete** from the dropdown.
Configure Browser RUM Performance Thresholds

You can configure the thresholds that define slow, very slow, and stalled end-user requests. These thresholds are used to trigger browser snapshots. See Transaction Thresholds for a detailed discussion about static, dynamic, percentage, and standard deviation thresholds.

You can define Browser Monitoring thresholds as one of the following:

- a multiple of the standard deviation. The default standard deviation threshold duration is two hours. For example: “Experience is slow if end-user response time is slower than 3 X the standard deviation.”
- a static value; For example: “Experience is stalled if end-user response time is slower than 30000 ms.”

The default thresholds are:

- Slow = 3 x standard deviation
- Very Slow = 4 x standard deviation
- Stalled = 45000 ms

Access Browser RUM Threshold Rules

1. Open the browser application in which you are interested.
2. On the left navigation bar, click Configuration.
3. Click Instrumentation >.
4. From the Settings tab, find the Thresholds for Slow End User Experience section.

Configure Browser RUM Threshold Rules

1. Select the relevant radio button to indicate whether the threshold is based on standard deviations or static values.
2. Type the values in the fields or select them using the scrollbars for one or more of the following:
   a. Slow greater than
   b. Very slow greater than
   c. Stall greater than
3. Click Save.
Configure Browser Snapshot Collection

Every page view and AJAX request sends a beacon to the EUM server. Every minute, the EUM server evaluates the collection of beacon samples and sends what are called "snapshots" to the Controller.

With browser snapshots, you can:

- Enable/disable slow snapshot collection, that is snapshots of requests where the End User Response Time is higher than the configured threshold.
- Enable/disable periodic snapshot collection.
- Enable/disable error snapshots, that is snapshots of requests for which a JavaScript error is reported or an Ajax request receives an HTTP error response. An error response is any HTTP code equal to or greater than 400.

If all three kinds of browser snapshot types—periodic, error, and slow response time—are disabled, the agent does not collect any browser snapshots.

Access Snapshot Collection Rules

1. Open the browser application in which you are interested
2. On the left navigation bar, select Configuration.
3. Click Instrumentation >.
4. Select the Settings tab.
5. Find the Event Policy Configuration section.

Configure Snapshot Collection Rules

1. Do one or more of the following:
   - Check or clear the check box Enable Slow Snapshot Collection to enable/disable slow response time snapshot collection.
   - Check or clear the check box Enable Periodic Snapshot Collection to enable/disable periodic snapshot collection.
   - Check or clear the check box Enable Error Snapshot Collection to enable/disable error snapshot collection.

2. Click Save.
Configure Web Percentile Metrics

Parts of the Controller UI for Browser RUM rely on the processing done by the Events Service, including some of the widgets in the Browser App Dashboard and the Analyze page. For these metrics, you can choose to display either averages or percentiles.

A percentile is a measure that indicates a value below which a given percentage of values in a set falls: for example, the 99th percentile means that 99% of all values are below this level. Using percentiles can be a good way to reduce the impact of extreme outliers in performance metrics, which can be useful in the often noisy environments of end-user monitoring. The Metric Browser also displays the percentiles.

You can:

- Enable/disable percentile metric display
- Set up to four different percentile levels to be applied to metrics

Access Configure Percentile Metrics

1. Open the browser application in which you are interested.
2. On the left navigation bar, select Configuration.
3. Click Instrumentation >.
4. Select the Settings tab.
5. Find the Configure Percentile Metrics section.

Configure Percentile Metrics

1. Check the Enable Percentile Metrics checkbox.
2. Enter up to four percentile levels to collect in the Percentiles to Collect fields. Each value must be a whole number between 1 and 99.
Configure Session Monitoring Timeouts

On this page:

- Access Session Monitoring
- Set Inactivity Timeout

Related pages:

- Browser RUM Sessions

You can control the timeframe for browser sessions by setting an inactivity limit. No activity beyond this limit is connected to previous activity as part of an individual session. See Session Persistence to learn more about how the duration of a session is determined.

Access Session Monitoring

1. Open the browser application in which you are interested.
2. On the left navigation bar, select Configuration.
3. Click Instrumentation >.
4. Select the Settings tab.
5. Find the Configure Session Monitoring section.

Set Inactivity Timeout

From Configure Session Monitoring, set the number of minutes for the Session Inactivity Timeout field.
Configure Request IP Address Storage - Browser

For security and privacy reasons, Browser RUM does not store source IP addresses associated with requests. To have Browser RUM store IP addresses associated with requests, follow the instructions below.

1. Open the browser application in which you are interested.
2. On the left navigation bar, select Configuration.
3. Click Instrumentation >.
4. Select the Settings tab.
5. At the bottom of the Settings tab, check the Store IP Address checkbox to enable IP address storage. (By default, the box is not checked.)
6. Click Save.
Correlate Business Transactions for Browser RUM

You can correlate page and Ajax requests with business transactions. In actuality, the correlation is made between instances of page/Ajax requests (browser snapshots) and instances of business transactions (transaction snapshots).

The correlation enables you to map end-user requests with an underlying backend application. Business-transaction correlation is disabled by default.

Benefits of Correlating Business Transactions

By correlating business transactions with browser snapshots, you can identify potential issues with the backend application that are causing bad user experiences. For example, you might find that server errors or a database query is causing a slow or very slow user experience.

Requirements

To correlate business transactions, you are required to do the following:

- purchase licenses for Application Performance Monitoring (APM) and End-User Monitoring (EUM)
- instrument an application with an app server agent that supports business transaction correlation
- inject a browser application with the JavaScript Agent
- enable business transaction correlation in the Controller UI

App Server Agents Supporting Business Transaction Correlation

To correlate business transactions, your business application must have one of the following app server agents installed:

- Java Agent
- .NET Agent
- Node.js Agent
- PHP Agent
- Python Agent

Avoid Tagging Cookies with the HttpOnly Flag

HttpOnly is a flag that servers can set on cookies to prevent their contents from being accessed by JavaScript. This is often done for session cookies to hide the session identifier as a security measure. The JavaScript Agent, however, needs to be able to read special cookies set by the AppDynamics server-side agent (all prefixed with ADRUM) to collect correlation information. If HttpOnly is set on these cookies, no server-side correlation information can be transmitted. Thus, make sure that your server does not set the HttpOnly flag on any cookies prefixed with ADRUM.

If you want to securely transmit cookies, use HTTPS. The app agent sets the secure flag if the originating base page is loaded over HTTPS.

These can be turned off, but that'd prevent correlation from working for base pages unless the customer changes the page source code to do footer injection.
How It Works

When an end user requests a page from your browser app:

1. The app agent does the following:
   - sends HTTP headers identifying the business transaction and the HTML with the injected JavaScript Agent to the end user's browser.
   - aggregates backend metrics and sends them along with the business transaction identifiers to the Controller. This serves as the content for the transaction snapshot.
2. The JavaScript Agent sends browser metrics and the business transaction identifiers (from the HTTP header) to the EUM Server. This serves as the content for the browser snapshot.
3. The Controller fetches the metrics and business transaction identifiers from the EUM server and then uses the business transaction identifiers to correlate the browser snapshot with the transaction snapshots.

Enable Business Transaction Correlation

You need to configure the Controller to correlate business transactions. The Controller will map the business transactions with the browser snapshots based on the process described above in How It Works.

To enable business transaction correlation:

1. From the Application Dashboard, click Configuration.
2. Click User Experience App Integration >.
3. From the Business Transaction Correlation tab, check the Enable Business Transaction Correlation check box.

Specify Business Transactions to Include Correlation Headers

You can also specify which business transactions will include or exclude correlation headers. If you do not add request match rules or request exclude rules, correlation headers will be added to all requests.

To add a request rule:

1. Click + to open the Create HTTP Request Match Rule dialog.
2. From the Create HTTP Request Match Rule dialog:
   a. Check the Method checkbox and select an HTTP method that you want to match. If you do not select an HTTP method, the rule will be applied to all HTTP requests.
   b. Check the URI checkbox and enter your criteria.
   c. Click Save to save the match rule.
3. From the Business Transaction Correlation tab, click Save.

View Business Transaction Correlation

There are several ways to navigate from a browser snapshot to its correlated business transaction. The following steps show you one possible way:

1. From the Browser App Dashboard, click Browser Snapshots.
2. Click Filters and check the Server Snapshot Exists checkbox.
3. You should now only see browser snapshots that have transaction snapshots as seen below:
3. Double-click one of the browser snapshots to open the Browser Snapshot Details dialog containing a transaction snapshot.

4. You can then click links in the transaction snapshot or transaction snapshot itself to view corresponding pages in APM.

Get Complete Timing Data for Correlated Business Transactions

To get the full real execution time for correlated business transactions, your injection method may need to write the `JS_FOOTER` variable to your page. Manual injection gives the server-side agent the ability to write data only to the header of the page as it is being constructed by your web application. It is possible that complete business-transaction timing information is not available at the moment that the header data is written. Using the footer allows the server-side agent to write timing data at the footer of the page, by which time a fuller picture of business transaction timing may be available.

You can write the `JS_FOOTER` data variable into the footer of a web page using the following techniques:
• If you use automatic injection for the injecting into the head section, you automatically get an injection into the footer as well.
• If you use manual injection for the head section, for applications built on Java platforms you can use assisted injection to inject into the footer. Or for applications built on Java servlet or ASP.NET platforms, you can use assisted injection using attribute injection.

If you cannot add the JS_FOOTER variable to your page, the timing shown for correlated business transactions may be the average response time for that transaction rather than the real execution time for that specific page.
Enable the Content Security Policy (CSP)

On this page:
- Directives Required for CSP
- `script-src`
- `connect-src`
- `child-src`
- `frame-ancestors`
- `img-src`
- Example Content-Security-Policy Header

This page shows you how to make configurations to enable CSP, so your application is compatible with Browser RUM.

Directives Required for CSP

To enable CSP for instrumented applications, you add the following required directives in the `Content-Security-Policy` header:

- `script-src`
- `connect-src`

In certain cases, you are also required to use the following directives:

- `child-src`
- `frame-ancestors`
- `img-src`

**script-src**

The `script-src` directive specifies the location of `adrum-ext.js`. By default, `adrum-ext.js` is loaded from our content delivery network (CDN) at `cdn.appdynamics.com`. The example below shows how you might use the `script-src` directive.

```
script-src cdn.appdynamics.com;
```

To measure first-byte time accurately, include the following line at the top of pages:

```
window["adrum-start-time"] = new Date().getTime();
```

For this line to be read, you also need to set the `script-src` directive to 'unsafe-inline' as shown here:
connect-src
The `connect-src` directive specifies the location where beacons are sent. If you are using the SaaS-based EUM, you might use something like the following:

```
connect-src col.eum-appdynamics.com;
```

If you are using on-prem EUM, you would have `connect-src` point to your EUM Server.

child-src
For cross-domain sessions, we load `adrum-xd.html` into an iframe. By default, this is loaded from our CDN, so you need to have `child-src` specify a CDN as shown below.

```
child-src cdn.appdynamics.com;
```

frame-ancestors
If `adrum-xd.html` is hosted locally, you would use the `frame-ancestors` directives in the following way:

```
frame-ancestors /path/to/adrum-xd.html;
```

img-src
In older browsers, we send our beacons as image beacons. Although older browsers don't support CSP, you can configure the JavaScript Agent to always send image beacons. You do this using `img-src` directive to specify the beacon location as shown in the example below.
Example Content-Security-Policy Header

The following Content-Security-Policy header loads the adrum files from our CDN and then sends beacons to our SaaS-based EUM.

```
Content-Security-Policy: connect-src 'self' col.eum-appdynamics.com; script-src 'unsafe-inline' cdn.appdynamics.com; img-src cdn.appdynamics.com; child-src cdn.appdynamics.com
```
Host a Geo Server

AppDynamics hosts a Geo Server that resolves the user's geographic location based on the request's reported IP address. You may prefer to host your own Geo Server if:

- You have intranet applications where the public IP address does not provide meaningful location information but the user's private IP does.
- You have a hybrid application where some users access the application from a private location and some access it from a public one. If a user doesn't come from a specific private IP range mapped by the custom geo server, the system can be set to default to the public Geo Server.

The custom Geo Server supports an HTTP header, AD-X-Forwarded-For. You can use this to declare an IP address specifically for the purpose of geo resolution. As of version 4.2, the custom Geo Server also supports modifying the JavaScript Agent itself to specify a particular IP or location. Custom Geo Server also supports the HTTP header X-Real-IP.

See Install and Host a Custom Geo Server for Browser RUM for more information on setting up your own Geo Server and private IP mapping file.

The AppDynamics Geo Server requires JDK 7 or higher.
Browser RUM Metrics

On this page:

- Browser RUM Timing Metrics Overview
- Browser RUM Metrics Defined

Related pages:

- Metric Browser
- AppDynamics APIs
- The Pages & Ajax Requests View
- WC3 Navigation Timing API

Key Browser RUM metrics are displayed on the Geo, Pages & AJAX Requests, and Usage Stats dashboards. They can also be seen on the All Pages and Top Pages lists and in the Metric Browser.

In addition, the Metric Browser enables you to view these metrics in the context of Ajax requests, iframes, applications, base pages, browsers, devices, and geographic locations.

You can build custom health rules based on Browser RUM Page, Ajax, and iframe metrics in the health rule builder. Use these rules to automatically monitor key metrics in your installation. For more information, see Health Rules.

Browser RUM Timing Metrics Overview

The following provides an overview of the first Browser RUM page and iframe timing metrics. For key Ajax metrics, see Ajax Metrics Availability. Metrics in blue are available only from NavTime browsers. You may see a value of unknown for some metrics taken using older browsers.

When users begin to load the first instrumented page, the JavaScript Agent starts a timer. The timer or the NavTime responseStart starts the first-page timing.
Subsequent Pages

End User Response Time (EURT)

Visually Complete Time (VCT)

DOM Ready Time

First Byte Time

Front End Time

HTML Download & DOM Building Time

Connection Details

navigationStart or start time cookie

responseStart or script timer starts

responseEnd

domContentLoadedEventStart or unready

loadEventEnd or unload

Connection Details
NavTiming-capable browsers also provide highly granular information on connection details.

**Browser RUM Metrics Defined**

Timing metrics are the average times, in milliseconds, over the time range selected in the Controller UI or REST API call. The three-letter abbreviation is the short name as it is recorded in the web beacon.

<table>
<thead>
<tr>
<th>Name (Short Name)</th>
<th>NavTiming Capable Browsers</th>
<th>Browsers without NavTiming Support</th>
<th>Available For . . .</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ajax Callback Execution Time (DPT)</td>
<td>responseEnd to</td>
<td>responseEnd to domContentLoadedEventStart (context of the Ajax object)</td>
<td>Ajax Requests (XHR, Fetch API)</td>
<td>Time for the browser to process the Ajax response. This typically includes the time to apply the response data to the DOM.</td>
</tr>
<tr>
<td></td>
<td>domContentLoadedEventStart (context of the Ajax object)</td>
<td>domContentLoadedEventStart (context of the Ajax object)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ajax Response Download Time (DDT)</td>
<td>responseStart to</td>
<td>responseStart to</td>
<td>Ajax Requests (XHR, Fetch API)</td>
<td>Time for the browser to download the complete Ajax response.</td>
</tr>
<tr>
<td></td>
<td>responseEnd</td>
<td>responseEnd (context of the Ajax object)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ajax Request Errors per Minute</td>
<td></td>
<td></td>
<td></td>
<td>• Ajax Requests (XHR, Fetch API)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• App</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Browser</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Device</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• Geo</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Total number of Ajax requests that generate an error per minute.</td>
</tr>
</tbody>
</table>

Because a NavTime browser always sets navigationStart, you can retrieve this information even on the first page a user loads from your site.
| **Ajax Requests per Minute**  
(Short name based on PLC entries) | • App  
• Browser  
• Device  
• Geo | Total number of Ajax requests per minute. |
|---|---|---|
| **Application Server Calls per Minute** | • Ajax Requests (XHR, Fetch API)  
• App  
• Base Pages  
• Browser  
• Device  
• Geo  
• iframes | Number of requests that actually hit the application server, rather than a cache. |
| **Application Server Time**  
(also called Server Time in the UI) | • Ajax Requests (XHR, Fetch API)  
• App  
• Base Pages  
• Browser  
• Device  
• Geo  
• iframes | Processing time for requests on the application server. |
| **Browser Queue Time** | • Ajax (XHR, Fetch API)  
• App  
• Base Pages  
• Browser  
• Device  
• Geo  
• iframes | Time for the browser to build the Document Object Model (DOM) and make it available for JavaScript to apply rendering logic. |
| **DOM Building Time (DPT)** | responseEnd to  
domContentLoadedEventStart | N/A | N/A  
• App  
• Base Pages  
• Browser  
• Device  
• Geo  
• iframes | Time for the browser to build the Document Object Model (DOM) and make it available for JavaScript to apply rendering logic. |
| **DOM Ready Time (DOM)** | navigationStart to  
domContentLoadedEventStart | Time between the writing of the start time cookie on the previous page and an internal handler's onready event, similar to jquery.onload() | N/A  
• App  
• Base Pages  
• Browser  
• Device  
• Geo  
• iframes | Interval between the time that a user initiates a request and the time that the DOMContentLoaded event (or the internal handler's onready event) occurs. |
| **Domain Lookup Time (DNS)** | domainLookupStart to  
domainLookupEnd | N/A | N/A  
• App  
• Base Pages  
• Browser  
• Device  
• Geo  
• iframes | Time to complete the domain lookup portion of the server connection time. |
<table>
<thead>
<tr>
<th>Metric</th>
<th>Start Event</th>
<th>End Event</th>
<th>Description</th>
<th>Example Events</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>End User Response Time (PLT)²</td>
<td>navigationStart</td>
<td>loadEventEnd</td>
<td>Time between the writing of the startTime cookie on the previous page and the onload event.</td>
<td>Ajax Requests (XHR, Fetch API), App, Base Pages, Browser, Device, Geo, iframes</td>
<td>Average interval between the time that a user initiates a request and the completion of the page load of the response in the user's browser. In the context of an Ajax request, it ends when the response has been completely processed. For information about End User Response Time for SPA2 pages, see SPA2 Metrics.</td>
</tr>
<tr>
<td>First Byte Time (FBT)²</td>
<td>navigationStart</td>
<td>responseStart</td>
<td>Time between the writing of the startTime cookie on the previous page and when the page's JavaScript Agent begins executing.</td>
<td>Ajax Requests (XHR, Fetch API), App, Base Pages, Browser, Device, Geo, iframes</td>
<td>Interval between the time that a user initiates a request and the time that the browser receives the first response byte. In the context of an Ajax request, First Byte Time is the interval between the Ajax request dispatch and the time that the browser receives the first response byte.</td>
</tr>
<tr>
<td>Front End Time (FET)</td>
<td>responseStart</td>
<td>loadEventEnd</td>
<td>Time between when the page's JavaScript Agent begins executing and the onload event.</td>
<td>App, Base Pages, Browser, Device, Geo, iframes</td>
<td>Interval between the arrival of the first byte of text response and the completion of the response page rendering by the browser. Includes HTML Download, DOM Building Time, and Resource Fetch Time.</td>
</tr>
<tr>
<td>HTML Download and DOM Building Time (DRT)</td>
<td>responseStart</td>
<td>domContentLoadedEventStart</td>
<td>Time between when the page's JavaScript Agent begins executing and an internal handler's onready event, similar to jQuery.onready().</td>
<td>App, Base Pages, Browser, Device, Geo, iframes</td>
<td>Time to make the complete HTML document (DOM) available for JavaScript to apply rendering logic. Includes the HTML Download and the DOM Building Time.</td>
</tr>
<tr>
<td>HTML Download Time (DDT)</td>
<td>responseStart</td>
<td>responseEnd</td>
<td>N/A</td>
<td>App, Base Pages, Browser, Device, Geo, iframes</td>
<td>Time for the browser to download the complete HTML document content.</td>
</tr>
<tr>
<td>Iframe Requests per Minute (Short name based on PLC entries)</td>
<td></td>
<td></td>
<td>Total number of iframe requests per minute.</td>
<td>App, Browser, Device, Geo</td>
<td></td>
</tr>
<tr>
<td>Images Loaded per Pageview</td>
<td></td>
<td></td>
<td>Total number of images in a pageview. Calculated after page has completely loaded.</td>
<td>Ajax (XHR, Fetch API), App, Base Pages, Browser, Device, Geo, iframes</td>
<td></td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
<td>Short Name</td>
<td>Full Screen Name</td>
<td>Availability</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
<td>------------------</td>
<td>-------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Page Complete Time (PCT)</strong></td>
<td>Time for the browser to render all visual page elements (both in and out of the viewport). Available for SPA base pages only. For information about Page Complete Time for SPA2 pages, see SPA2 Metrics.</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Page Requests per Minute</strong></td>
<td>Total number of Page requests per minute. This is the metric displayed across most of the UI.</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Page views with JavaScript Errors per minute</strong></td>
<td>Total number of page views that contain JavaScript errors per minute.</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Resource Fetch Time</strong></td>
<td>Time for the browser to complete the download of remaining resources, including images, and finish rendering the page.</td>
<td>domContentLoadedEventStart</td>
<td>loadEventEnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Resources Loaded per Pageview</strong></td>
<td>Total number of resources in a page. Calculated after the page has completely loaded.</td>
<td>Ajax</td>
<td>(XHR, Fetch API)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Requests per Minute</strong></td>
<td>Total number of requests (Page + Ajax + iframe) per minute.</td>
<td>Ajax</td>
<td>Requests (XHR, Fetch API)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Scripts Loaded per Pageview</strong></td>
<td>Total number of scripts in a pageview. Calculated after the page has completely loaded.</td>
<td>Ajax</td>
<td>(XHR, Fetch API)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metric</td>
<td>Start Event</td>
<td>End Event</td>
<td>Time Type</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>------------------------------</td>
<td>------------------------------</td>
<td>-----------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Server Connection Time (SCT)</strong></td>
<td>navigationStart</td>
<td>requestStart</td>
<td>N/A</td>
<td>Interval between the time that a user initiates a request and the start of fetching the response document from the server or application task. Includes the time spent on redirects, domain lookups, TCP connects and SSL handshakes.</td>
<td></td>
</tr>
<tr>
<td><strong>SSL Handshake Time (SSL)</strong></td>
<td>secureConnectionStart</td>
<td>connectEnd</td>
<td>N/A</td>
<td>Time taken to complete the SSL handshake.</td>
<td></td>
</tr>
<tr>
<td><strong>TCP Connect Time (TCP)</strong></td>
<td>connectStart</td>
<td>connectEnd</td>
<td>N/A</td>
<td>Time to complete the TCP connect portion of the server connection time, the equivalent of one network roundtrip of latency.</td>
<td></td>
</tr>
<tr>
<td><strong>Total Resource Redirect Time</strong></td>
<td></td>
<td></td>
<td>N/A</td>
<td>Time for all redirects associated with fetching resources. Calculated after the page has completely loaded.</td>
<td></td>
</tr>
<tr>
<td><strong>Visually Complete Time (VCT)</strong></td>
<td></td>
<td></td>
<td>N/A</td>
<td>Total time for all visual elements within the first screen (above-the-fold content) to load in an end user’s browser. Calculated using the last visual change to the page in a browser window. For information about Visually Complete Time for SPA2 pages, see SPA2 Metrics.</td>
<td></td>
</tr>
</tbody>
</table>
Browser RUM Licenses

On this page:
- License Entitlements and Restrictions
- License Editions
- Usage Period
- License Key
- Browser Real-User Monitoring Agents Licensed
- Pageviews Allocated
- Pageviews Consumed
- Overages

Related pages:
- The Pages & Ajax Requests View
- Set Up and Access Browser RUM

This topic describes how to interpret the details of your Browser RUM licensing information. It assumes that you have acquired your Web EUM license from your AppDynamics sales representative.

License Entitlements and Restrictions

For AppDynamics official license details, entitlements, and restrictions, see the following:

- Browser Real User Monitoring (SaaS)
- Browser Real User Monitoring (on-prem)

License Editions

The two supported license editions for Browser RUM are described in the table below.

<table>
<thead>
<tr>
<th>Available Editions</th>
<th>License Unit</th>
<th>Description/Notes</th>
</tr>
</thead>
</table>
| Pro                | Pageviews per year, per agent | • A Pageview is an instance of a base page loaded by a Web browser. Repeated views of one page are counted as separate Pageviews.  
• AppDynamics counts a Pageview each time the JavaScript Agent for Browser RUM is executed on a base page. Only Pageviews of pages that are instrumented by the agent are counted as Pageviews consumed. For example, pages excluded from monitoring through configuration are not counted.  
• Five Ajax requests are counted as one Pageview.  
• One iFrame is counted as one Pageview. |
| Lite               | N/A          | A Browser RUM Lite license includes the following features:  
• Geo Dashboard for the previous 24-hour period for up to 500 million Pageviews  
• Page/iframe List for the last 24-hour period for up to 500 million Pageviews  
• Usage Stats for the last 24-hour period for up to 500 million Pageviews |

Usage Period

The usage period is per year, even if you have a multi-year license.

The Browser RUM meter resets every year based on the expiration of your license agreement. For example, if your license expires on 5/15/2020, your usage period resets every year on 5/15.

License Key
This is the license key associated with your EUM license.

**Browser Real-User Monitoring Agents Licensed**

This is the number of agents you have currently licensed. Each agent provides a certain number of annual Pageviews.

**Pageviews Allocated**

This is the number of Pageviews allocated for the current usage period by your license agreement.

**Pageviews Consumed**

This is the number of Pageviews consumed during the current usage period.

If this value is greater than **Page Views Allocated** and your license allows overages, you are incurring overage charges.

If you need to stop incurring overage charges, you can disable Browser RUM to stop end-user monitoring and Pageviews from being charged after a delay of approximately one minute.

**Overages**

Your Browser RUM license agreement determines how overages are handled. When you reach a limit, you are notified at the top of the **All Pages** section of the **Pages & Ajax Requests** tab.

If your license does not allow overages, AppDynamics stops reporting metrics after your Browser RUM limit has been reached.

If your license does allow overages and your usage exceeds the limit, AppDynamics continues reporting Browser RUM metrics and bills you for the overages at the unit rate stipulated by your license agreement prorated over the number of Pageviews that exceed the limit.
Browser RUM Supported Environments

The following tables describe the supported environments and versions for Browser RUM.

Browser RUM Support

**Browser Compatibility**
- IE6/7/8/9/10/11/Edge
- Chrome, including Mobile
- Firefox, including Mobile
- Safari, including Mobile
- Opera

Browsers are rapidly evolving, and not all versions have been individually tested with Browser RUM. You can view the browser versions likely to support the Resource Timing API.

**Browser Requirements for Sessions**
To use Browser RUM sessions, your browser is required to support for the following:
- cross-origin resource sharing (CORS) for beacons
- local storage for multiple-page sessions (single-page / multiple virtual page sessions don’t need local storage)

Browser RUM sessions do not support beacons implemented with GIFs.

**JavaScript Agent Version Requirement for Sessions**
You are required to use the JavaScript Agent 4.2+. Older versions of the JavaScript Agent do not support sessions.

**Injection Types for Browser RUM in Java Environments**
Listed below are the injection types with their supported frameworks/technologies:
- **Automatic Injection**: JavaServer Pages (JSP) compiled using the Jasper compiler. Jasper is the default JSP compiler in Tomcat, Glassfish, and JBoss.
- **Assisted Injection**: All Servlet frameworks.
- **Manual Injection**: All technologies that generate HTML pages.

See [Inject the JavaScript Agent](#) for details.

**Injection Types for Browser RUM in .NET Environments**
AppDynamics certifies Browser RUM instrumentation for the following .NET frameworks:
- All listed frameworks support manual injection of the JavaScript Agent for Browser RUM.
- The Script Injection column list additionally supported script injection strategies. See [Inject the JavaScript Agent](#) for details.
<table>
<thead>
<tr>
<th>Web Application/ AJAX Frameworks</th>
<th>Versions</th>
<th>Additional Supported Script Injection Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP.NET Web Forms (.aspx)</td>
<td>3, 4</td>
<td>Automatic, Using Attribute Injection</td>
</tr>
<tr>
<td>ASP.NET MVC Web Forms (.aspx)</td>
<td>3, 4, 5</td>
<td>Automatic, Using Attribute Injection</td>
</tr>
<tr>
<td>ASP.NET MVC Razor</td>
<td>3, 4, 5</td>
<td>Automatic, Using Attribute Injection</td>
</tr>
<tr>
<td>ASP.NET Core on the full .NET Framework</td>
<td>4.0+</td>
<td>Using Attribute Injection</td>
</tr>
<tr>
<td>Microsoft SharePoint</td>
<td>2007, 2010</td>
<td>Automatic</td>
</tr>
</tbody>
</table>

AppDynamics does not support Browser RUM instrumentation of legacy ASP (.asp) pages.

**Supported Runtime Environments for .NET Browser RUM**

- Microsoft IIS versions 6.0, 7.0, 7.5, 8.0, 8.5
Browser RUM Countries and Regions by Geo Dashboard

On this page:

- Countries
- Regions

The countries and regions in the sections below can be displayed in the Geo Dashboard. Each of these countries and regions also displays their own aggregate Browser RUM data. Data can be collected from areas not in the following list, but it will not be displayed in these parts of the interface. Some countries and regions are only displayed in the grid view.

Countries

**Grid View**

The following countries are displayed in the grid view of the Geo Dashboard.

**A – D**

Afghanistan
Aland Islands
Albania
Algeria
American Samoa
Andorra
Angola
Anguilla
Antarctica
Antigua and Barbuda
Argentina
Armenia
Aruba
Asia/Pacific Region
Australia
Austria
Azerbaijan
Bahamas
Bahrain
Bangladesh
Barbados
Belarus
Belgium
Belize
Benin
Bermuda
Bhutan
Bolivia
Bonaire, Saint Eustatius, and Saba
Bosnia and Herzegovina
Botswana
Bouvet Island
Brazil
British Indian Ocean Territory
Brunei Darussalam
Bulgaria
Burkina Faso
Burundi
Cambodia
Cameroon
Canada
Cape Verde
Cayman Islands
Central African Republic
Chad
Chile
China
Christmas Island
Cocos (Keeling) Islands
Colombia
Comoros
Congo
Congo, The Democratic Republic of the
Cook Islands
Costa Rica
Cote d'Ivoire
Croatia
Cuba
Curacao
Cyprus
Czech Republic
Denmark
Djibouti
Dominica
Dominican Republic

E – K
Ecuador
Egypt
El Salvador
Equatorial Guinea
Eritrea
Estonia
Ethiopia
Europe
Falkland Islands (Malvinas)
Faroe Islands
Fiji
Finland
France
French Guiana
French Polynesia
French Southern Territories
Gabon
Gambia
Georgia
Germany
Ghana
Gibraltar
Greece
Greenland
Grenada
Guadeloupe
Guam
Guatemala
Guernsey
Guinea
Guinea-Bissau
Guyana
Haiti
Heard Island and...
McDonald Islands
Holy See (Vatican City State)
Honduras
Hong Kong
Hungary
Iceland
India
Indonesia
Iran, Islamic Republic of
Iraq
Ireland
Isle of Man
Israel
Italy
Jamaica
Japan
Jersey
Jordan
Kazakhstan
Kenya
Kiribati
Korea, Democratic People's Republic of
Korea, Republic of
Kuwait
Kyrgyzstan

L – Q

Lao People's Democratic Republic
Latvia
Lebanon
Lesotho
Liberia
Libyan Arab Jamahiriya
Liechtenstein
Lithuania
Luxembourg
Macao
Macedonia
Madagascar
Malawi
Malaysia
Maldives
Mali
Malta
Marshall Islands
Martinique
Mauritania
Mauritius
Mayotte
Mexico
Micronesia, Federated States of
Moldova, Republic of
Monaco
Mongolia
Montenegro
Montserrat
Morocco
Mozambique
Myanmar
Namibia
Nauru
Nepal
Netherlands
New Caledonia
New Zealand
Nicaragua
Niger
Nigeria
Niue
Norfolk Island
Northern Mariana Islands
Norway
Oman
Pakistan
Palau
Palestinian Territory
Panama
Papua New Guinea
Paraguay
Peru
Philippines
Pitcairn
Poland
Portugal
Puerto Rico
Qatar

R – Z

Reunion
Romania
Russian Federation
Rwanda
Saint Barthelemy
Saint Helena
Saint Kitts and Nevis
Saint Lucia
Saint Martin
Saint Pierre and Miquelon
Saint Vincent and the Grenadines
Samoa
San Marino
Sao Tome and Principe
Saudi Arabia
Senegal
Serbia
Seychelles
Sierra Leone
Singapore
Sint Maarten
Slovakia
Slovenia
Solomon Islands
Somalia
South Africa
South Georgia and the South Sandwich Islands
Spain
Sri Lanka
Sudan
Suriname
Svalbard and Jan Mayen
Swaziland
Sweden
Switzerland
Syrian Arab Republic
Taiwan
Tajikistan
Tanzania, United Republic of
Thailand
Timor-Leste
Togo
Tokelau
Tonga
Trinidad and Tobago
Tunisia
Turkey
Turkmenistan
Turks and Caicos Islands
Tuvalu
Uganda
Ukraine
United Arab Emirates
United Kingdom
United States
United States Minor Outlying Islands
Uruguay
Uzbekistan
Vanuatu
Venezuela
Vietnam
Virgin Islands, British
Virgin Islands, U.S.
Wallis and Futuna
Western Sahara
Yemen
Zambia
Zimbabwe

Map View

The following countries are displayed in the map view of the Geo Dashboard.

A – D

Afghanistan
Aland Islands
Albania
Algeria
American Samoa
Andorra
Angola
Anguilla
Antigua and Barbuda
Argentina
Armenia
Aruba
Australia
Austria
Azerbaijan
Bahamas
Bahrain
Baker Island
Bangladesh
Barbados
Belarus
Belgium
Belize
Benin
Bermuda
Bhutan
Bolivia
Bosnia and Herzegovina
Botswana
Bouvet Island
Brazil
British Indian Ocean Territory
Brunei Darussalam
Bulgaria
Burkina Faso
Burundi
Cambodia
Cameroon
Canada
Cape Verde
Cayman Islands
Central African Republic
Chad
Chile
China
Christmas Island
Cocos (Keeling) Island
Colombia
Comoros
Republic of Congo
the Democratic Republic of the Congo
Cook Islands
Costa Rica
Cote d'Ivoire
Croatia
Cuba
Cyprus
Czech Republic
Denmark
Djibouti
Dominica
Dominican Republic

E – K

Ecuador
Egypt
El Salvador
Equatorial Guinea
Eritrea
Estonia
Ethiopia
Faeroe Islands
Falkland Islands
Fiji
Finland
France
French Guiana
French Polynesia
French Southern and Antarctic Territories
Gabon
Gambia, The
Georgia
Germany
Ghana
Gibraltar
Glorioso Islands
Greece
Greenland
Grenada
Guadeloupe
Guatemala
Guam
Guinea
Guinea-Bissau
Guernsey
Guyana
Haiti
Heard and Mcdonald Islands
Honduras
Hong Kong
Howland Island
Hungary
Iceland
India
Indonesia
Iran
Iraq
Ireland
Isle of Man
Israel
Italy
Jamaica
Japan
Jarvis Island
Jersey
Johnston Atoll
Juan De Nova Island
Jordan
Kazakhstan
Kenya
Kiribati

L - Q

Lao People's Democratic Republic
Latvia
Lebanon
Lesotho
Liberia
Libya
Liechtenstein
Lithuania
Luxembourg
Macau
Macedonia
Madagascar
Malawi
Malaysia
Maldives
Mali
Malta
Marshall Islands
Martinique
Mauritania
Mauritius
Mayotte
Mexico
Federated States of Micronesia
Midway Islands
Moldova
Monaco
Mongolia
Montenegro
Montserrat
Morocco
Mozambique
Myanmar
Namibia
Nauru
Nepal
Netherlands
New Caledonia
New Zealand
Nicaragua
Niger
Nigeria
Niue
Northern Mariana Islands
Norway
Oman
Pakistan
Palau
Palestinian Territories
Panama
Papua New Guinea
Paraguay
Peru
Philippines
Pitcairn Islands
Poland
Portugal
Puerto Rico
Qatar

R – Z

Reunion
Romania
Russian Federation
Rwanda
Saint Helena
Saint Kitts and Nevis
Saint Lucia
Saint Pierre and Miquelon
Saint Martin
Saint Vincent and the Grenadines
Samoa
San Marino
Sao Tome and Principe
Saudi Arabia
Senegal
Serbia
Seychelles
Sierra Leone
Singapore
Slovakia
Slovenia
Solomon Islands
Somalia
South Africa
South Georgia and the South Sandwich Islands
Spain
Sri Lanka
Sudan
Suriname
Svalbard and Jan Mayen
Swaziland
Sweden
Switzerland
Syrian Arab Republic
Taiwan
Tajikistan
Tanzania
Thailand
Timor-Leste
Togo
Tonga
Tokelau
Trinidad and Tobago
Tunisia
Turkey
Turkmenistan
Turks and Caicos Islands
Tuvalu
Uganda
Ukraine
United Arab Emirates
United Kingdom
Uruguay
Vanuatu
Uzbekistan
Vatican City
Venezuela
Viet Nam
Virgin Islands, British
Virgin Islands, US
Wake Island
Western Sahara
Wallis and Futuna
Yemen
Zambia
Zimbabwe
Regions

Grid View

The following countries displayed in the grid view of the Geo Dashboard also report data by region.

A – G

Afghanistan, Badakhshan
Afghanistan, Badghis
Afghanistan, Baglan
Afghanistan, Balkh
Afghanistan, Bamian
Afghanistan, Daykondi
Afghanistan, Farah
Afghanistan, Faryab
Afghanistan, Ghazni
Afghanistan, Ghowr
Afghanistan, Helmand
Afghanistan, Herat
Afghanistan, Jowzjan
Afghanistan, Kabul
Afghanistan, Kandahar
Afghanistan, Kapisa
Afghanistan, Khowst
Afghanistan, Konar
Afghanistan, Kondoz
Afghanistan, Laghman
Afghanistan, Lowgar
Afghanistan, Nangarhar
Afghanistan, Nimruz
Afghanistan, Nurestan
Afghanistan, Oruzgan
Afghanistan, Paktia
Afghanistan, Paktika
Afghanistan, Panjshir
Afghanistan, Parvan
Afghanistan, Samangan
Afghanistan, Sar-e Pol
Afghanistan, Takhar
Afghanistan, Vardak
Afghanistan, Zabol
Albania, Berat
Albania, Diber
Albania, Durres
Albania, Elbasan
Albania, Fier
Albania, Gjirokaster
Albania, Korce
Albania, Kukes
Albania, Lezhe
Albania, Shkoder
Albania, Tirane
Albania, Vlore
Algeria, Adrar
Algeria, Ain Defla
Algeria, Ain Temouchent
Algeria, Alger
Algeria, Annaba
Algeria, Batna
Algeria, Bechar
Algeria, Bejaia
Algeria, Biskra
Algeria, Blida
Algeria, Bordj Bou Arreridj
Algeria, Bouira
Algeria, Boumerdes
Algeria, Chef
Algeria, Constantine
Algeria, Djelfa
Algeria, El Bayadh
Algeria, El Oued
Algeria, El Tarf
Algeria, Ghardaia
Algeria, Guelma
Algeria, Illizi
Algeria, Jijel
Algeria, Khenchela
Algeria, Laghouat
Algeria, M'sila
Algeria, Mascara
Algeria, Medea
Algeria, Mila
Algeria, Mostaganem
Algeria, Naama
Algeria, Oran
Algeria, Ouargla
Algeria, Oum el Bouaghi
Algeria, Relizane
Algeria, Saida
Algeria, Selif
Algeria, Sidi Bel Abbes
Algeria, Skikda
Algeria, Souk Ahras
Algeria, Tamanghasset
Algeria, Tebessa
Algeria, Tiaret
Algeria, Tindouf
Algeria, Tipaza
Algeria, Tissemsilt
Algeria, Tizi Ouzou
Algeria, Tlemcen
Andorra, Andorra la Vella
Andorra, Canillo
Andorra, Encamp
Andorra, Escaldes-Engordany
Andorra, La Massana
Andorra, Ordino
Andorra, Sant Julia de Loria
Angola, Bengo
Angola, Benguela
Angola, Bie
Angola, Cabinda
Angola, Cuando Cubango
Angola, Cuanza Norte
Angola, Cuanza Sul
Angola, Cunene
Angola, Huambo
Angola, Huila
Angola, Luanda
Angola, Lunda Norte
Angola, Lunda Sul
Angola, Malanje
Angola, Mexico
Angola, Namibe
Angola, Uige
Angola, Zaire
Antigua and Barbuda, Barbuda
Antigua and Barbuda, Redonda
Antigua and Barbuda, Saint George
Antigua and Barbuda, Saint John
Antigua and Barbuda, Saint Mary
Antigua and Barbuda, Saint Paul
Antigua and Barbuda, Saint Peter
Antigua and Barbuda, Saint Philip
Argentina, Buenos Aires
Argentina, Catamarca
Argentina, Chaco
Argentina, Chubut
Argentina, Cordoba
Argentina, Corrientes
Argentina, Distrito Federal
Argentina, Entre Rios
Argentina, Formosa
Argentina, Jujuy
Argentina, La Pampa
Argentina, La Rioja
Argentina, Mendoza
Argentina, Misiones
Argentina, Neuquen
Azerbaijan,
Bilasuvar
Azerbaijan, Cabrayil
Azerbaijan, Callilabad
Azerbaijan, Daskasan
Azerbaijan, Davaci
Azerbaijan, Fuzuli
Azerbaijan, Gadabay
Azerbaijan, Ganca
Azerbaijan, Goranboy
Azerbaijan, Goycay
Azerbaijan, Haciqabul
Azerbaijan, Imisli
Azerbaijan, Ismayilli
Azerbaijan, Kalbacar
Azerbaijan, Kurdamir
Azerbaijan, Lacin
Azerbaijan, Lankaran
Azerbaijan, Lankaran
Azerbaijan, Lerik
Azerbaijan, Masalli
Azerbaijan, Mingacevir
Azerbaijan, Naftalan
Azerbaijan, Naxcivan
Azerbaijan, Neftcala
Azerbaijan, Oguz
Azerbaijan, Qabala
Azerbaijan, Qax
Azerbaijan, Qazax
Azerbaijan, Qobustan
Azerbaijan, Quba
Azerbaijan, Qubadli
Azerbaijan, Qusar
Azerbaijan, Saatli
Azerbaijan, Sabirabad
Azerbaijan, Saki
Azerbaijan, Saki
Azerbaijan, Salyan
Azerbaijan, Samaxi
Azerbaijan, Samkir
Azerbaijan, Samux
Azerbaijan, Siyazan
Azerbaijan, Sumqayit
Azerbaijan, Susa
Azerbaijan, Susa
Azerbaijan, Tartar
Azerbaijan, Tovuz
Azerbaijan, Ucar
Azerbaijan, Xacmaz
Azerbaijan, Xankandi
Azerbaijan, Xanlar
Azerbaijan, Xizi
Azerbaijan, Xocali
Azerbaijan, Xocavand
Azerbaijan, Yardimli
Azerbaijan, Yevlax
Azerbaijan, Yevlax
Azerbaijan, Zangilan
Azerbaijan, Zaqatala
Azerbaijan, Zardab
Bahrain, Al Asimah
Bahrain, Al Hadd
Bahrain, Al Janubiyah
Bahrain, Al Manama
Bahrain, Al Mintaqah al Gharbiyah
Bahrain, Al Mintaqah al Wusta
Bahrain, Al Mintaqah ash Shamaliyah
Bahrain, Al Muharraq
Bahrain, Al Wusta
Bahrain, Ar Rifa
Bahrain, Ash Shamaliyah
Bahrain, Jidd Hafs
Bahrain, Madinat Hamad
Bahrain, Mintaqat Juzur Hawar
Bahrain, Sitrah
Bangladesh, Barisal
Bangladesh, Chittagong
Bangladesh, Dhaka
Bangladesh, Khulna
Bangladesh, Rajshahi
Bangladesh, Sylhet
Barbados, Christ Church
Barbados, Saint Andrew
Barbados, Saint George
Barbados, Saint James
Barbados, Saint John
Barbados, Saint Joseph
Barbados, Saint Lucy
Barbados, Saint Michael
Barbados, Saint Peter
Barbados, Saint Philip
Barbados, Saint Thomas
Belarus, Brestskaya
Voblasts' Belarus, Homyel'skaya Voblasts' Belarus, Hrodzyenskaya Voblasts' Belarus, Mahilyowskaya Voblasts' Belarus, Minsk Belarus, Minskaya Voblasts' Belarus, Vitsebskaya Voblasts' Belgium, Antwerpen Belgium, Brabant Wallon Belgium, Brussels Hoofdstedelijk Gewest Belgium, Hainaut Belgium, Liege Belgium, Limburg Belgium, Luxembourg Belgium, Namur Belgium, Oost-Vlaanderen Belgium, Vlaams-Brabant Belgium, West-Vlaanderen Belize, Belize Belize, Cayo Belize, Corozal Belize, Orange Walk Belize, Stann Creek Belize, Toledo Benin, Alibori Benin, Atakora Benin, Atlanyique Benin, Borgou Benin, Collines Benin, Donga Benin, Kouffo Benin, Littoral Benin, Mono Benin, Oueme Benin, Plateau Benin, Zou Bermuda, Devonshire Bermuda, Hamilton Bermuda, Paget Bermuda, Pembroke Bermuda, Saint George Bermuda, Sandys Bermuda, Smiths Bermuda, Southampton
Bermuda, Warwick
Bhutan, Bumthang
Bhutan, Chhukha
Bhutan, Chirang
Bhutan, Daga
Bhutan, Geylegphug
Bhutan, Ha
Bhutan, Lhuntshi
Bhutan, Mongar
Bhutan, Paro
Bhutan, Pemagatsel
Bhutan, Punakha
Bhutan, Samchi
Bhutan, Samdrup
Bhutan, Shemgang
Bhutan, Tashigang
Bhutan, Thimphu
Bhutan, Tongsa
Bhutan, Wangdi
Phodrang
Bolivia, Chuquisaca
Bolivia, Cochabamba
Bolivia, El Beni
Bolivia, La Paz
Bolivia, Oruro
Bolivia, Pando
Bolivia, Potosi
Bolivia, Santa Cruz
Bolivia, Tarija
Bosnia and Herzegovina, Federation of
Bosnia and Herzegovina
Bosnia and Herzegovina, Republika Srpska
Botswana, Central
Botswana, Ghanzi
Botswana, Kgalagadi
Botswana, Kgatleng
Botswana, Kweneng
Botswana, North-East
Botswana, North-West
Botswana, South-East
Botswana, Southern
Brazil, Acre
Brazil, Alagoas
Brazil, Amapa
Brazil, Amazonas
Brazil, Bahia
Brazil, Ceará
Brazil, Distrito Federal
Brazil, Espírito Santo
Brazil, Goiás
Brazil, Maranhão
Brazil, Mato Grosso
Brazil, Mato Grosso do Sul
Brazil, Minas Gerais
Brazil, Para
Brazil, Paraiba
Brazil, Parana
Brazil, Pernambuco
Brazil, Piaui
Brazil, Rio de Janeiro
Brazil, Rio Grande do Norte
Brazil, Rio Grande do Sul
Brazil, Rondonia
Brazil, Roraima
Brazil, Santa Catarina
Brazil, Sao Paulo
Brazil, Sergipe
Brazil, Tocantins
Brunei Darussalam, Alibori
Brunei Darussalam, Belait
Brunei Darussalam, Brunei and Muara
Brunei Darussalam, Collines
Brunei Darussalam, Donga
Brunei Darussalam, Kouffo
Brunei Darussalam, Littoral
Brunei Darussalam, Oueme
Brunei Darussalam, Plateau
Brunei Darussalam, Temburong
Brunei Darussalam, Tutong
Brunei Darussalam, Zou
Bulgaria, Blagoevgrad
Bulgaria, Burgas
Bulgaria, Dobrich
Bulgaria, Gabrovo
Bulgaria, Grad Sopiya
Bulgaria, Khaskovo
Bulgaria, Kurdzhali
Bulgaria, Kyustendil
Bulgaria, Lovech
Bulgaria, Mikhailovgrad
Bulgaria, Montana
Bulgaria, Pazardzhik
Bulgaria, Pernik
Bulgaria, Pleven
Bulgaria, Plovdiv
Bulgaria, Razgrad
Bulgaria, Ruse
Bulgaria, Shumen
Bulgaria, Silistra
Bulgaria, Siliven
Bulgaria, Smolyan
Bulgaria, Sofiya
Bulgaria, Stara Zagora
Bulgaria, Turgovishte
Bulgaria, Varna
Bulgaria, Veliko Tarnovo
Bulgaria, Vidin
Bulgaria, Vratsa
Bulgaria, Yambol
Burkina Faso, Bale
Burkina Faso, Bam
Burkina Faso, Banwa
Burkina Faso, Bazega
Burkina Faso, Bougouriba
Burkina Faso, Boulgou
Burkina Faso, Bouliemde
Burkina Faso, Ganzourgou
Burkina Faso, Gnagna
Burkina Faso, Gourma
Burkina Faso, Houet
Burkina Faso, Iboa
Burkina Faso, Kadiogo
Burkina Faso, Kenedougou
Burkina Faso, Kompienga
Burkina Faso, Kompienga
Burkina Faso, Kossi
Burkina Faso, Koulelengo
Burkina Faso, Kouritenga
Burkina Faso, Kourweogo
Burkina Faso, Leraba
Burkina Faso, Loroum
Burkina Faso, Mouhoun
Burkina Faso, Namentenga
Burkina Faso, Naouri
Burkina Faso, Nayala
Burkina Faso, Noumbiel
Burkina Faso, Oubritenga
Burkina Faso, Oudalan
Burkina Faso,
Passore
Burkina Faso, Poni
Burkina Faso,
Sanguie
Burkina Faso,
Samaratenga
Burkina Faso, Seno
Burkina Faso, Sissili
Burkina Faso,
Soum
Burkina Faso,
Sourou
Burkina Faso,
Tapa
Burkina Faso, Tuy
Burkina Faso,
Yagha
Burkina Faso,
Yatenga
Burkina Faso, Ziro
Burkina Faso,
Zoundweogo
Burundi, Bubanza
Burundi, Bujumbura
Burundi, Bururi
Burundi, Cankuzo
Burundi, Cibitoke
Burundi, Gilega
Burundi, Karuzi
Burundi, Kayanza
Burundi, Kirundo
Burundi, Makamba
Burundi, Muramvya
Burundi, Muyinga
Burundi, Mwaro
Burundi, Ngozi
Burundi, Rutana
Burundi, Ruyigi
Cambodia, Banteay Meanchey
Cambodia,
Baldambang
Cambodia,
Baldambang
Cambodia,
Kampong Cham
Cambodia,
Kampong Chhnang
Cambodia,
Kampong Speu
Cambodia,
Kampong Thom
Cambodia, Kampot
Cambodia, Kandal
Cambodia, Koh Kong
Cambodia, Kracheh
Cambodia,
Mondulkiri
Cambodia, Pailin
Cambodia, Phnom Penh
Cambodia, Preah Vihear
Cambodia, Prey Veng
Cape Verde, Sao Vicente
Cape Verde, Tarrafal
Cayman Islands, Creek
Cayman Islands, Eastern
Cayman Islands, Midland
Cayman Islands, South Town
Cayman Islands, Spot Bay
Cayman Islands, Stake Bay
Cayman Islands, West End
Cayman Islands, Western
Central African Republic, Bamingui-Bangoran
Central African Republic, Bangui
Central African Republic, Basse-Kotto
Central African Republic, Cuvette-Ouest
Central African Republic, Haut-Mbomou
Central African Republic, Haute-Kotto
Central African Republic, Kemo
Central African Republic, Lobaye
Central African Republic, Mambere-Kadei
Central African Republic, Mbomou
Central African Republic, Nana-Grebizi
Central African Republic, Nana-Mamber
Central African Republic, Ombella-Mpoko
Central African Republic, Ouaka
Central African Republic, Ouham
Central African Republic, Ouham-Pende
Central African Republic, Sangha-Mbaere
Chad, Batha
Chad, Biltine
Chad,
China, Xinjiang
China, Xizang
China, Yunnan
China, Zhejiang
Colombia,
Amazonas
Colombia, Antioquia
Colombia, Arauca
Colombia, Atlantico
Colombia, Bolivar
Colombia, Bolivar Department
Colombia, Boyaca
Colombia, Boyaca Department
Colombia, Caldas
Colombia, Caldas Department
Colombia, Caqueta
Colombia, Casanare
Colombia, Cauca
Colombia, Cesar
Colombia, Choco
Colombia, Cordoba
Colombia, Cundinamarca
Colombia, Distrito Especial
Colombia, Guainia
Colombia, Guaviare
Colombia, Huila
Colombia, La Guajira
Colombia, Magdalena
Colombia, Magdalena Department
Colombia, Meta
Colombia, Narino
Colombia, Norte de Santander
Colombia, Putumayo
Colombia, Quindio
Colombia, Risaralda
Colombia, San Andres y Providencia
Colombia, Santander
Colombia, Sucre
Colombia, Tolima
Colombia, Valle del Cauca
Colombia, Vaupes
Colombia, Vichada
Comoros, Anjouan
Comoros, Grande Comore
Comoros, Moheli
Congo, Bouenza
Congo, Brazzaville
Congo, Cuvette
Congo, Cuvette-Ouest
Congo, Kouilou
Congo, Lekoumou
Congo, Likouala
Congo, Niari
Congo, Plateaux
Congo, Pool
Congo, Sangha
Congo, The Democratic Republic of the, Bandundu
Congo, The Democratic Republic of the, Bas-Congo
Congo, The Democratic Republic of the, Equateur
Congo, The Democratic Republic of the, Kasai-Oriental
Congo, The Democratic Republic of the, Katanga
Congo, The Democratic Republic of the, Kinshasa
Congo, The Democratic Republic of the, Maniema
Congo, The Democratic Republic of the, Nord-Kivu
Congo, The Democratic Republic of the, Orientale
Congo, The Democratic Republic of the, Sud-Kivu
Costa Rica, Alajuela
Costa Rica, Cartago
Costa Rica, Guanacaste
Costa Rica, Heredia
Costa Rica, Limon
Costa Rica, Puntarenas
Costa Rica, San Jose
Cote D'Ivoire, Agneby
Cote D'Ivoire, Bafing
Cote D'Ivoire, Bas-Sassandra
Cote D'Ivoire, Denguele
Cote D'Ivoire, Dix-Huit Montagnes
Cote D'Ivoire, Fromager
Cote D'Ivoire, Haut-Sassandra
Cote D'Ivoire, Lacs
Cote D'Ivoire, Lagunes
Cote D'Ivoire, Marahoue
Cote D'Ivoire, Moyen-Cavally
Cote D'Ivoire, Moyen-Comoe
Cote D'Ivoire, N'zi-Comoe
Cote D'Ivoire, Savanes
Cote D'Ivoire, Sud-Bandama
Cote D'Ivoire, Sud-Comoe
Cote D'Ivoire, Vallee du Bandama
Cote D'Ivoire, Worodougou
Cote D'Ivoire, Zanzan
Croatia, Bjelovarsko-Bilogorska
Croatia, Brodsko-Posavska
Croatia, Dubrovacko-Neretvanska
Croatia, Grad Zagreb
Croatia, Istarska
Croatia, Karlovacka
Croatia, Koprivnicko-Krizevacka
Croatia, Krapinsko-Zagorska
Croatia, Licko-Senjska
Croatia, Medimurska
Croatia, Osjecko-Baranjska
Croatia, Pozesko-Slavonska
Croatia, Primorsko-Goranska
Croatia, Sibenisko-Kninska
Croatia, Sisacko-Moslavacka
Croatia, Splitsko-Dalmatinska
Croatia, Varazdinska
Croatia, Viroviticko-Podravska
Croatia, Vukovarsko-Srijemska
ka
Croatia, Zadarska
Croatia, Zagrebacka
Cuba, Camaguey
Cuba, Ciego de Avila
Cuba, Cienfuegos
Cuba, Ciudad de la Habana
Cuba, Granma
Cuba, Guantanamo
Cuba, Holguin
Cuba, Isla de la Juventud
Cuba, La Habana
Cuba, Las Tunas
Cuba, Matanzas
Cuba, Pinar del Rio
Cuba, Sancti Spiritus
Cuba, Santiago de Cuba
Cuba, Villa Clara
Cyprus, Famagusta
Cyprus, Kyrenia
Cyprus, Larnaca
Cyprus, Limassol
Cyprus, Nicosia
Cyprus, Paphos
Czech Republic, Hlavni mesto Praha
Czech Republic, Jihocesky kraj
Czech Republic, Jihomoravsky kraj
Czech Republic, Karlovarsky kraj
Czech Republic, Kralovehradecky kraj
Czech Republic, Liberecky kraj
Czech Republic, Moravskoslezsky kraj
Czech Republic, Olomoucky kraj
Czech Republic, Pardubicky kraj
Czech Republic, Plzensky kraj
Czech Republic, Stredo cesky kraj
Czech Republic, Ustecky kraj
Czech Republic, Vysochina
Czech Republic, Zlinsky kraj
Denmark, Hovedstaden
Denmark, Midtjylland
Denmark, Nordjylland
Denmark, Sjælland
Denmark,
Syddanmark
Djibouti, Ali Sabieh
Djibouti, Arta
Djibouti, Dikhil
Djibouti, Djibouti
Djibouti, Obock
Djibouti, Tadjoura
Dominica, Saint Andrew
Dominica, Saint David
Dominica, Saint George
Dominica, Saint John
Dominica, Saint Joseph
Dominica, Saint Luke
Dominica, Saint Mark
Dominica, Saint Patrick
Dominica, Saint Paul
Dominica, Saint Peter
Dominican Republic, Azua
Dominican Republic, Baoruco
Dominican Republic, Barahona
Dominican Republic, Distrito Nacional
Dominican Republic, Duarte
Dominican Republic, El Seibo
Dominican Republic, Elias Pina
Dominican Republic, Espaillat
Dominican Republic, Hato Mayor
Dominican Republic, Independencia
Dominican Republic, La Altagracia
Dominican Republic, La Romana
Dominican Republic, La Vega
Dominican Republic, Maria Trinidad Sanchez
Dominican Republic, Monsenor
Nouel
Dominican Republic, Monte Cristi
Dominican Republic, Monte Plata
Dominican Republic, Pedernales
Dominican Republic, Peravia
Dominican Republic, Peravia
Dominican Republic, Puerto Plata
Dominican Republic, Salcedo
Dominican Republic, Samana
Dominican Republic, San Cristobal
Dominican Republic, San Jose de Ocoa
Dominican Republic, San Juan
Dominican Republic, San Pedro De Macoris
Dominican Republic, Sanchez Ramirez
Dominican Republic, Santiago
Dominican Republic, Santiago Rodriguez
Dominican Republic, Santo Domingo
Dominican Republic, Valverde
Ecuador, Azuay
Ecuador, Bolivar
Ecuador, Canar
Ecuador, Carchi
Ecuador, Chimborazo
Ecuador, Cotopaxi
Ecuador, El Oro
Ecuador, Esmeraldas
Ecuador, Galapagos
Ecuador, Guayas
Ecuador, Imbabura
Ecuador, Loja
Ecuador, Los Rios
Ecuador, Manabi
Ecuador, Morona-Santiago
Ecuador, Napo
Ecuador, Orellana
Ecuador, Pastaza
Ecuador, Pichincha
Ecuador,
Sucumbios
Ecuador,
Tungurahua
Ecuador,
Zamora-Chinchipe
Egypt, Ad
Daqahliyah
Egypt, Al Bahr al
Ahmar
Egypt, Al Buhayrah
Egypt, Al Fayyum
Egypt, Al Gharbiyah
Egypt, Al
Iskandariyah
Egypt, Al Isma‘iliyah
Egypt, Al Jizah
Egypt, Al Minufiyah
Egypt, Al Minya
Egypt, Al Qahirah
Egypt, Al Qalyubiyah
Egypt, Al Wadi al
Jadid
Egypt, As Suways
Egypt, Ash
Sharqiyyah
Egypt, Aswan
Egypt, Asyut
Egypt, Bani Suwayf
Egypt, Bur Sa’id
Egypt, Dumyat
Egypt, Janub Sina’
Egypt, Kafr ash
Shaykh
Egypt, Matruh
Egypt, Qina
Egypt, Shamal Sina’
Egypt, Suhaj
El Salvador,
Ahuachapan
El Salvador,
Cabanas
El Salvador,
Chalatenango
El Salvador,
Cuscatlan
El Salvador, La
Libertad
El Salvador, La Paz
El Salvador, La
Union
El Salvador,
Morazan
El Salvador, San
Miguel
El Salvador, San
Salvador
El Salvador, San
Vicente
El Salvador, Santa
Ana
El Salvador,
Sonsonate
El Salvador,
Usulutan
Equatorial Guinea,
Annobon
Fiji, Western
Finland, Aland
Finland, Eastern
Finland
Finland, Lapland
Finland, Oulu
Finland, Southern
Finland
Finland, Western
Finland
France, Alsace
France, Aquitaine
France, Auvergne
France, Basse-Normandie
France, Bourgogne
France, Bretagne
France, Centre
France, Champagne-Arden
ne
France, Corse
France, Franche-Comte
France, Haute-Normandie
France, Ile-de-France
France, Languedoc-Roussill
on
France, Limousin
France, Lorraine
France, Midi-Pyrenees
France, Nord-Pas-de-Calais
France, Pays de la Loire
France, Picardie
France, Poitou-Charentes
France, Provence-Alpes-Cot
e d'Azur
France, Rhone-Alpes
Gabon, Estuaire
Gabon, Haut-Ogooue
Gabon, Moyen-Ogooue
Gabon, Ngounie
Gabon, Nyanga
Gabon, Ogooue-Ivindo
Gabon, Ogooue-Lolo
Gabon, Ogooue-Maritime
Gabon, Woleu-Ntem
Gambia, Banjul
Gambia, Central
River
Gambia, Lower
River
Gambia, North
Bank
Gambia, Upper
River
Gambia, Western
Georgia, Abashis
Raioni
Georgia, Abkhazia
Georgia, Adigenis
Raioni
Georgia, Ajaria
Georgia, Akhalgoris
Raioni
Georgia, Akhal'ak'is
Raioni
Georgia, Akhalts'ikhis Raioni
Georgia, Akhmetis
Raioni
Georgia, Ambrolauris Raioni
Georgia, Aspindzis
Raioni
Georgia, Bagdat'is
Raioni
Georgia, Bolnisis
Raioni
Georgia, Borjomis
Raioni
Georgia, Ch'khorotsqus
Raioni
Georgia, Ch'okhatauris
Raioni
Georgia, Chia'tura
Georgia, Dedop'listsqaros
Raioni
Georgia, Dmanisis
Raioni
Georgia, Dusheti's
Raioni
Georgia, Gardabani Raioni
Georgia, Gori
Georgia, Goris
Raioni
Georgia, Gurjaanis
Raioni
Georgia, Javis
Raioni
Georgia, K'arelis
Raioni
Georgia, K'ut'aisi
Georgia, Kaspis
Raioni
Georgia, Kharagaulis Raioni
Georgia, Khashuris
Raioni
Georgia, Khobis
Raioni
Georgia, Khonis
Raioni
Georgia, Lagodekhis Raioni
Georgia,
Lanch'khut'is Raioni
Georgia, Lentekhis Raioni
Georgia, Marneulis Raioni
Georgia, Martvilis Raioni
Georgia, Mestis Raioni
Georgia, Mts'khet'is Raioni
Georgia, Ninotsmindis Raioni
Georgia, Onis Raioni
Georgia, Ozuget'is Raioni
Georgia, P'ot'i Raioni
Georgia, Qazbegis Raioni
Georgia, Qvarlis Raioni
Georgia, Rust'avi Raioni
Georgia, Sach'kheris Raioni
Georgia, Sagarejos Raioni
Georgia, Samtrediis Raioni
Georgia, Senakis Raioni
Georgia, Sighnaghis Raioni
Georgia, T'bilisi Raioni
Georgia, T'elavis Raioni
Georgia, T'erjolis Raioni
Georgia, T'enitsqaros Raioni
Georgia, Tianet'is Raioni
Georgia, Tqibli Raioni
Georgia, Ts'ageris Raioni
Georgia, Tsalenjikhis Raioni
Georgia, Tsalkis Raioni
Georgia, Tsqaltubo Raioni
Georgia, Vanis Raioni
Georgia, Zestap'onis Raioni
Georgia, Zugdidi Raioni
Georgia, Zugdidis Raioni
Germany, Baden-Wurttemberg
Germany, Bayern
Germany, Berlin
Germany, Brandenburg
Germany, Bremen
Germany, Hamburg
Germany, Hessen
Germany, Mecklenburg-Vorpo
mmern
Germany,
Niedersachsen
Germany,
Nordrhein-Westfalen
Germany,
Rheinland-Pfalz
Germany, Saarland
Germany, Sachsen
Germany,
Sachsen-Anhalt
Germany,
Schleswig-Holstein
Germany,
Thuringen
Ghana, Ashanti
Ghana,
Brong-Ahafo
Ghana, Central
Ghana, Eastern
Ghana, Greater
Accra
Ghana, Northern
Ghana, Upper East
Ghana, Upper West
Ghana, Volta
Ghana, Western
Greece, Aitolia kai Akarnania
Greece, Akhaia
Greece, Argolis
Greece, Arkadhia
Greece, Arta
Greece, Attiki
Greece,
Dhodhekanisos
Greece, Drama
Greece, Evritania
Greece, Evros
Greece, Evvoia
Greece, Florina
Greece, Fokis
Greece, Fthiotis
Greece, Grevena
Greece, Ilia
Greece, Imathia
Greece, Ioannina
Greece, Iraklion
Greece, Karditsa
Greece, Kastoria
Greece, Kavala
Greece, Kefallinia
Greece, Kerkira
Greece, Katakdhiki
Greece, Khania
Greece, Khois
Greece, Kikladhes
Greece, Kilkis
Greece, Korinthia
Greece, Kozani
Greece, Lakonia
Greece, Larisa
Greece, Lasithi
Greece, Lesvos
Greece, Levkas
Greece, Magnisia
Greece, Messinia
Greece, Pella
Greece, Pieria
Greece, Preveza
Greece, Rethimni
Greece, Rodhopi
Greece, Samos
Greece, Serrai
Greece, Thesprotia
Greece, Thessaloniki
Greece, Trikala
Greece, Voiotia
Greece, Xanthi
Greece, Zakinthos
Greenland, Nordgronland
Greenland, Ostgronland
Greenland, Vestgronland
Grenada, Saint Andrew
Grenada, Saint David
Grenada, Saint George
Grenada, Saint John
Grenada, Saint Mark
Grenada, Saint Patrick
Guatemala, Alta Verapaz
Guatemala, Baja Verapaz
Guatemala, Chimaltenango
Guatemala, Chiquimula
Guatemala, El Progreso
Guatemala, Escuintla
Guatemala, Guatemala
Guatemala, Huehuetenango
Guatemala, Izabal
Guatemala, Jalapa
Guatemala, Jutiapa
Guatemala, Peten
Guatemala, Quetzaltenango
Guatemala, Sacatepequez
Guatemala, San Marcos
Guatemala, Santa Rosa
Guatemala, Solola
Guatemala, Suchitepequez
Guatemala, Totonicapan
Guatemala, Zacapa
Guinea-Bissau, Bafata
Guinea-Bissau, Baflo
Guinea-Bissau, Biombo
Guinea-Bissau, Bissau
Guinea-Bissau, Bolama
Guinea-Bissau, Cacheu
Guinea-Bissau, Gabu
Guinea-Bissau, Olo
Guinea-Bissau, Quinara
Guinea-Bissau, Tombali
Guinea, Beyla
Guinea, Boffa
Guinea, Boke
Guinea, Conakry
Guinea, Coyah
Guinea, Dabola
Guinea, Dalaba
Guinea, Dinguiraye
Guinea, Dubreka
Guinea, Faranah
Guinea, Forecariah
Guinea, Fria
Guinea, Gaoual
Guinea, Gueckedou
Guinea, Kankan
Guinea, Kerouane
Guinea, Kindia
Guinea, Kissidougou
Guinea, Koubia
Guinea, Koundara
Guinea, Kouroussa
Guinea, Labe
Guinea, Lelouma
Guinea, Lola
Guinea, Macenta
Guinea, Mali
Guinea, Mamou
Guinea, Mandiana
Guinea, Nzerekore
Guinea, Pita
Guinea, Sigui
Guinea, Telimele
Guinea, Tougue
Guinea, Yomou
Guyana,
Barima-Waini
Guyana,
Cuyuni-Mazaruni
Guyana,
Demerara-Mahaica
Guyana, East
Berbice-Corentyne
Guyana, Essequibo
Islands-West
Demerara
Guyana,
Mahaica-Berbice
Guyana,
Pomeroon-Supenna
Guyana, Potaro-Siparuni
Guyana, Upper
Demerara-Berbice
Guyana, Upper
Takutu-Upper
Essequibo

H – M
Haiti, Artibonite
Haiti, Centre
Haiti, Grand’ Anse
Haiti, Nippes
Haiti, Nord
Haiti, Nord-Est
Haiti, Nord-Ouest
Haiti, Ouest
Haiti, Sud
Haiti, Sud-Est
Honduras, Atlantida
Honduras, Choluteca
Honduras, Colon
Honduras, Comayagua
Honduras, Copan
Honduras, Cortes
Honduras, El Paraíso
Honduras, Francisco Morazán
Honduras, Gracias a Dios
Honduras, Intibuca
Honduras, Islas de la Bahía
Honduras, La Paz
Honduras, Lempira
Honduras, Olancho
Honduras, Santa Barbara
Honduras, Valle
Honduras, Yoro
Hungary, Bacs-Kiskun
Hungary, Baranya
Hungary, Bekes
Hungary, Békescsaba
Hungary, Borsod-Abaúj-Zemplén
Hungary, Budapest
Hungary, Csongrad
Hungary, Debrecen
Hungary, Dunaujvaros
Hungary, Eger
Hungary, Erd
Hungary, Fejér
Hungary, Győr
Hungary, Győr-Moson-Sopron
Hungary,
Hajdu-Bihar
Hungary, Heves
Hungary,
Hodmezovasarhely
Hungary,
Jasz-Nagy kun-Szol
nok
Hungary, Kaposvar
Hungary,
Kecskemet
Hungary,
Komarom-Esztergo
m
Hungary, Miskolc
Hungary,
Nagykanyiza
Hungary, Nograd
Hungary,
Nyregyhaza
Hungary, Pecs
Hungary, Pest
Hungary,
Salgotarjan
Hungary, Somogy
Hungary, Sopron
Hungary,
Szabolcs-Szatmar-
Bereg
Hungary, Szeged
Hungary,
Szekesfehervar
Hungary, Szekszard
Hungary, Szolnok
Hungary,
Szombathely
Hungary,
Tatabanya
Hungary, Tolna
Hungary, Vas
Hungary, Veszprem
Hungary, Veszprem
Hungary, Zala
Hungary,
Zalaegerszeg
Iceland, Arnarssysla
Iceland,
Austur-Hunavatnss
ysla
Iceland,
Austur-Skaftafellssy
sla
Iceland,
Borgarfjarardarsysla
Iceland,
Eyjafjardarsysla
Iceland,
Gulfringusysla
Iceland, Kjosarsysla
Iceland, Myrasysla
Iceland,
Nordur-Mulasysla
Iceland,
Nordur-Tingeyjarss
yla
Iceland, Norourland
Eystra
Iceland, Norourland
Vestra
Iceland,
Rangarvallasysla
Iceland,
Skagafjardarsysla
Iceland,
Snæfellsnes- og
Hnappadalassysla
Iceland,
Strandasysla
Iceland,
Sudur-Mulasysla
Iceland,
Sudur-Tingeyjarsyla
Iceland, Suurland
Iceland, Suurnes
Iceland, Vestfjord
Iceland,
Vestur-Bardastrandararsysla
Iceland,
Vestur-Hunavatnssysla
Iceland,
Vestur-Isafjardarsysla
Iceland,
Vestur-Skaftafellssysla
Iceland, Vesturland
India, Andaman and
Nicobar Islands
India, Andhra
Pradesh
India, Arunachal
Pradesh
India, Assam
India, Bihar
India, Chandigarh
India, Chhattisgarh
India, Dadra and
Nagar Haveli
India, Daman and
Diu
India, Delhi
India, Goa
India, Gujarat
India, Haryana
India, Himachal
Pradesh
India, Jammu and
Kashmir
India, Jharkhand
India, Karnataka
India, Kerala
India, Lakshadweep
India, Madhya
Pradesh
India, Maharashtra
India, Manipur
India, Meghalaya
India, Mizoram
India, Nagaland
India, Orissa
India, Puducherry
India, Punjab
India, Rajasthan
India, Sikkim
India, Tamil Nadu
India, Tripura
India, Uttar Pradesh
India, Uttarakhand
India, West Bengal
Indonesia, Aceh
Indonesia, Bali
Indonesia, Banten
Indonesia, Bengkulu
Indonesia, Gorontalo
Indonesia, Irian Jaya Barat
Indonesia, Jakarta
Indonesia, Jambi
Indonesia, Jawa Barat
Indonesia, Jawa Tengah
Indonesia, Jawa Timur
Indonesia, Kalimantan Barat
Indonesia, Kalimantan Selatan
Indonesia, Kalimantan Tengah
Indonesia, Kalimantan Timur
Indonesia, Kepulauan Bangka Belitung
Indonesia, Kepulauan Riau
Indonesia, Lampung
Indonesia, Maluku Utara
Indonesia, Nusa Tenggara Barat
Indonesia, Nusa Tenggara Timur
Indonesia, Papua
Indonesia, Papua Barat
Indonesia, Riau
Indonesia, Riau Barat
Indonesia, Sulawesi Barat
Indonesia, Sulawesi Selatan
Indonesia, Sulawesi Tengah
Indonesia, Sulawesi Tenggara
Indonesia, Sulawesi Utara
Indonesia, Sumatera Barat
Indonesia,
Sumatera Selatan
Indonesia,
Sumatera Selatan
Indonesia,
Sumatera Utara
Indonesia,
Yogyakarta
Iran, Islamic
Republic of, Ardabil
Iran, Islamic
Republic of,
Azarbajjan-e
Bakhtari
Iran, Islamic
Republic of,
Bakhtaran
Iran, Islamic
Republic of,
Bushehr
Iran, Islamic
Republic of, Chahar
Mahal va Bakhtari
Iran, Islamic
Republic of, East
Azerbaijan
Iran, Islamic
Republic of,
Esfahan
Iran, Islamic
Republic of, Fars
Iran, Islamic
Republic of, Gilan
Iran, Islamic
Republic of,
Golestan
Iran, Islamic
Republic of,
Hamadan
Iran, Islamic
Republic of,
Hormozgan
Iran, Islamic
Republic of, Ilam
Iran, Islamic
Republic of,
Kerman
Iran, Islamic
Republic of,
Kerman
Iran, Islamic
Republic of,
Khorasan
Iran, Islamic
Republic of,
Khorasan-e Janubi
Iran, Islamic
Republic of,
Khorasan-e Razavi
Iran, Islamic
Republic of,
Khorasan-e Shemali
Iran, Islamic
Republic of,
Khuzestan
Iran, Islamic
Republic of,
Kohkiluyeh va Boyer Ahmadi
Iran, Islamic Republic of, Kordestan
Iran, Islamic Republic of, Lorestan
Iran, Islamic Republic of, Markazi
Iran, Islamic Republic of, Mazandaran
Iran, Islamic Republic of, Qazvin
Iran, Islamic Republic of, Qom
Iran, Islamic Republic of, Semnan
Iran, Islamic Republic of, Semnan Province
Iran, Islamic Republic of, Sistan va Baluchestan
Iran, Islamic Republic of, Tehran
Iran, Islamic Republic of, Yazd
Iran, Islamic Republic of, Zanjan
Iran, Islamic Republic of, Zanjan
Iraq, Al Anbar
Iraq, Al Basra
Iraq, Al Muthanna
Iraq, Al Qadisiyah
Iraq, An Najaf
Iraq, Arbil
Iraq, As Sulaymaniyah
Iraq, At Ta'mim
Iraq, Babil
Iraq, Baghdad
Iraq, Dahuk
Iraq, Dhi Qar
Iraq, Diyala
Iraq, Karbala'
Iraq, Maysan
Iraq, Ninawa
Iraq, Salah ad Din
Iraq, Wasit
Ireland, Carlow
Ireland, Cavan
Ireland, Clare
Ireland, Cork
Ireland, Donegal
Ireland, Dublin
Ireland, Galway
Ireland, Kerry
Ireland, Kildare
Ireland, Kilkenny
Ireland, Laois
Ireland, Leitrim
Ireland, Limerick
Ireland, Longford
Ireland, Louth
Ireland, Mayo
Ireland, Meath
Ireland, Monaghan
Ireland, Offaly
Ireland, Roscommon
Ireland, Sligo
Ireland, Tipperary
Ireland, Waterford
Ireland, Westmeath
Ireland, Wexford
Ireland, Wicklow
Israel, HaDarom
Israel, HaMerkaz
Israel, HaZafon
Israel, Hefa
Israel, Tel Aviv
Israel, Yerushalayim
Italy, Abruzzi
Italy, Basilicata
Italy, Calabria
Italy, Campania
Italy, Emilia-Romagna
Italy, Friuli-Venezia Giulia
Italy, Lazio
Italy, Liguria
Italy, Lombardia
Italy, Marche
Italy, Molise
Italy, Piemonte
Italy, Puglia
Italy, Sardegna
Italy, Sicilia
Italy, Toscana
Italy, Trentino-Alto Adige
Italy, Umbria
Italy, Valle d'Aosta
Italy, Veneto
Jamaica, Clarendon
Jamaica, Hanover
Jamaica, Kingston
Jamaica, Manchester
Jamaica, Portland
Jamaica, Saint Andrew
Jamaica, Saint Ann
Jamaica, Saint Catherine
Jamaica, Saint

Copyright © AppDynamics 2012-2019
Elizabeth
Jamaica, Saint
James
Jamaica, Saint
Mary
Jamaica, Saint
Thomas
Jamaica, Trelawny
Jamaica, Westmoreland
Japan, Aichi
Japan, Akita
Japan, Aomori
Japan, Chiba
Japan, Ehime
Japan, Fukui
Japan, Fukuoka
Japan, Fukushima
Japan, Gifu
Japan, Gumma
Japan, Hiroshima
Japan, Hokkaido
Japan, Hyogo
Japan, Ibaraki
Japan, Ishikawa
Japan, Iwate
Japan, Kagawa
Japan, Kagoshima
Japan, Kanagawa
Japan, Kochi
Japan, Kumamoto
Japan, Kyoto
Japan, Mie
Japan, Miyagi
Japan, Miyazaki
Japan, Nagano
Japan, Nagasaki
Japan, Nara
Japan, Niigata
Japan, Okinawa
Japan, Osaka
Japan, Saga
Japan, Saitama
Japan, Shiga
Japan, Shimane
Japan, Shizuoka
Japan, Tochigi
Japan, Tokushima
Japan, Tokyo
Japan, Tottori
Japan, Toyama
Japan, Wakayama
Japan, Yamagata
Japan, Yamaguchi
Japan, Yamanashi
Jordan, Al Balqa'
Jordan, Al Karak
Jordan, Al Mafraq
Jordan, Amman
Jordan, Amman Governorate
Jordan, At Tafilah
Jordan, Az Zarqa
Jordan, Irbid
Jordan, Ma
Kazakhstan, Almaty
Kazakhstan, Almaty
Kazakhstan,
Aqmola
Kazakhstan, Aqtobe
Kazakhstan, Astana
Kazakhstan, Atyrau
Kazakhstan,
Bayqonyr
Kazakhstan, East
Kazakhstan
Kazakhstan,
Pavlodar
Kazakhstan,
Qaraghandy
Kazakhstan,
Qostanay
Kazakhstan,
Qyzylorda
Kazakhstan, South
Kazakhstan
Kazakhstan, West
Kazakhstan
Kazakhstan,
Zhambyl
Kenya, Central
Kenya, Coast
Kenya, Eastern
Kenya, Nairobi Area
Kenya,
North-Eastern
Kenya, Nyanza
Kenya, Rift Valley
Kenya, Western
Kiribati, Gilbert
Islands
Kiribati, Line Islands
Kiribati, Phoenix
Islands
Korea, Democratic
People's Republic
of, Chagang-do
Korea, Democratic
People's Republic
of, Hamgyong-bukto
Korea, Democratic
People's Republic
of, Hamgyong-namdo
Korea, Democratic
People's Republic
of, Hwanghae-bukto
Korea, Democratic
People's Republic
of, Hwanghae-namdo
Korea, Democratic
People's Republic
of, Kaesong-si
Korea, Democratic
People's Republic
of, Kangwon-do
Korea, Democratic
People's Republic
of, Najin Sonbong-si
Korea, Democratic
People's Republic
of, Namp'o-si
Korea, Democratic
People's Republic
of, P'yongan-bukto
Korea, Democratic
People's Republic
of, P'yongan-namdo
Korea, Democratic
People's Republic
of, P'yongyang-si
Korea, Democratic
People's Republic
of, Yanggang-do
Korea, Republic of,
Ch'ungch'ong-bukto
Korea, Republic of,
Ch'ungch'ong-namdo
Korea, Republic of,
Cheju-do
Korea, Republic of,
Cholla-bukto
Korea, Republic of,
Cholla-namdo
Korea, Republic of,
Inch''on-jikhalsi
Korea, Republic of,
Kangwon-do
Korea, Republic of,
Kwangju-jikhalsi
Korea, Republic of,
Kyonggi-do
Korea, Republic of,
Kyongsang-bukto
Korea, Republic of,
Kyongsang-namdo
Korea, Republic of,
Pusan-jikhalsi
Korea, Republic of,
Seoul Teukbyeolsi
Korea, Republic of,
Seoul-teukpyolsi
Korea, Republic of,
Taegu-jikhalsi
Korea, Republic of,
Taejon-jikhalsi
Korea, Republic of,
Ulsan-gwangyoksi
Kuwait, Al Ahmadi
Kuwait, Al
Farwaniyah
Kuwait, Al Jahra
Kuwait, Al Kuwait
Kuwait, Hawalli
Kuwait, Mubarak al
Kabir
Kyrgyzstan, Batken
Kyrgyzstan, Bishkek
Kyrgyzstan, Chuy
Kyrgyzstan, Jalal-Abad
Kyrgyzstan, Naryn
Kyrgyzstan, Osh
Kyrgyzstan, Talas
Latvia, Madonas
Latvia, Ogres
Latvia, Preili
Latvia, Rezekne
Latvia, Rezeknes
Latvia, Riga
Latvia, Rigas
Latvia, Saldus
Latvia, Talsu
Latvia, Tukuma
Latvia, Valkas
Latvia, Valmieras
Latvia, Ventspils
Latvia, Ventspils
Lebanon, Aakk
Lebanon, Al Janub
Lebanon, Baalbek-Hermel
Lebanon, Beqaa
Lebanon, Beqaa
Lebanon, Beyrouth
Lebanon, Liban-Nord
Lebanon, Liban-Nord
Lebanon, Liban-Sud
Lebanon, Mont-Liban
Lebanon, Nabatiye
Lesotho, Berea
Lesotho, Butha-Buthe
Lesotho, Leribe
Lesotho, Mafeteng
Lesotho, Maseru
Lesotho, Mohales Hoek
Lesotho, Mokhotlong
Lesotho, Qachas Nek
Lesotho, Quthing
Lesotho, Thaba-Tseka
Liberia, Bong
Liberia, Gbarpolu
Liberia, Grand Bassa
Liberia, Grand Cape Mount
Liberia, Grand Cape Mount
Liberia, Grand Gedeh
Liberia, Lofa
Liberia, Lofa
Liberia, Margibi
Liberia, Maryland
Liberia, Maryland
Liberia, Monrovia
Liberia, Montserrado
Liberia, Nimba
Liberia, River Cess
Liberia, River Gee
Liberia, Sino
Libyan Arab Jamahiriya,
Ajdabiya
Libyan Arab
Jamahiriya, Al
Aziziyah
Libyan Arab
Jamahiriya, Al Fatih
Libyan Arab
Jamahiriya, Al Jabal al Akhdar
Libyan Arab
Jamahiriya, Al Jufra
Libyan Arab
Jamahiriya, Al Khums
Libyan Arab
Jamahiriya, Al Kufran
Libyan Arab
Jamahiriya, An
Nuqat al Khams
Libyan Arab
Jamahiriya, Ash Shati'
Libyan Arab
Jamahiriya, Awbari
Libyan Arab
Jamahiriya, Az Zawiyah
Libyan Arab
Jamahiriya, Banghazi
Libyan Arab
Jamahiriya, Darna
Libyan Arab
Jamahiriya, Ghadamis
Libyan Arab
Jamahiriya, Gharyan
Libyan Arab
Jamahiriya, Misratah
Libyan Arab
Jamahiriya, Murzuq
Libyan Arab
Jamahiriya, Sabha
Libyan Arab
Jamahiriya, Sawfajjin
Libyan Arab
Jamahiriya, Surt
Libyan Arab
Jamahiriya, Tarabulus
Libyan Arab
Jamahiriya, Tarhunah
Libyan Arab
Jamahiriya, Tubruq
Libyan Arab
Jamahiriya, Yafran
Libyan Arab
Jamahiriya, Zlitan
Liechtenstein,
Balzers
Liechtenstein,
Eschen
Liechtenstein,
Gamprin
Liechtenstein,
Gbarpolu
Liechtenstein,
Mauren
Liechtenstein,
Planken
Liechtenstein, River
Gee
Liechtenstein,
Ruggell
Liechtenstein,
Schaan
Liechtenstein,
Schellenberg
Liechtenstein,
Triesen
Liechtenstein,
Triesenberg
Liechtenstein,
Vaduz
Lithuania, Alytaus
Apskritis
Lithuania, Kauno
Apskritis
Lithuania, Klaipedos
Apskritis
Lithuania,
Marijampoles
Apskritis
Lithuania,
Panevezio Apskritis
Lithuania, Siauliu
Apskritis
Lithuania, Taurages
Apskritis
Lithuania, Telsiu
Apskritis
Lithuania, Utenos
Apskritis
Lithuania, Vilniaus
Apskritis
Luxembourg,
Diekirch
Luxembourg,
Grevenmacher
Luxembourg,
Macau, Ilhas
Macau, Macau
Macedonia,
Aracinovo
Macedonia, Bac
Macedonia, Belcista
Macedonia, Berovo
Macedonia, Bistrica
Macedonia, Bitola
Macedonia, Blatec
Macedonia,
Bogdanci
Macedonia,
Bogomila
Macedonia,
Bogovinje
Macedonia,
Bosilovo
Macedonia,
Brvenica
Macedonia, Cair
Macedonia, Capari
Macedonia, Caska
Macedonia, Cegrane
Macedonia, Centar
Macedonia, Centar Zupa
Macedonia, Cesinovo
Macedonia, Cucer-Sandevo
Macedonia, Debar
Macedonia, Delcevo
Macedonia, Delogozdi
Macedonia, Demir Hisar
Macedonia, Demir Kapija
Macedonia, Dobrusevo
Macedonia, Dolna Banjica
Macedonia, Dolneni
Macedonia, Dorche Petrov
Macedonia, Drugovo
Macedonia, Dzepciste
Macedonia, Gazi Baba
Macedonia, Gevgelija
Macedonia, Gostivar
Macedonia, Gradsko
Macedonia, Ilindeni
Macedonia, Izvor
Macedonia, Jegunovce
Macedonia, Kamenjane
Macedonia, Karbinci
Macedonia, Karpos
Macedonia, Kavadarci
Macedonia, Kicevo
Macedonia, Kisela Voda
Macedonia, Kicevce
Macedonia, Kocani
Macedonia,Konce
Macedonia, Kondovo
Macedonia, Konopiste
Macedonia, Kosel
Macedonia, Kratovo
Macedonia, Kriva Palanka
Macedonia,
Srbinovo, Macedonia, Star
Dorjan, Macedonia, Staravina
Macedonia, Staro Nagoricane
Macedonia, Stip
Macedonia, Struga
Macedonia, Strumica
Macedonia, Studenicani
Macedonia, Suto Orizari
Macedonia, Sveti Nikole
Macedonia, Tearce
Macedonia, Tetovo
Macedonia, Topolcani
Macedonia, Valandovo
Macedonia, Vasilevo
Macedonia, Veles
Macedonia, Velesta
Macedonia, Vevcani
Macedonia, Vinica
Macedonia, Vitoliste
Macedonia, Vranestica
Macedonia, Vrapciste
Macedonia, Vratnica
Macedonia, Vrutok
Macedonia, Zajas
Macedonia, Zelenikovo
Macedonia, Zelino
Macedonia, Zitose
Macedonia, Zletovo
Macedonia, Zrnovci
Madagascar, Antananarivo
Madagascar, Antsiranana
Madagascar, Fianarantsoa
Madagascar, Mahajanga
Madagascar, Toamasina
Madagascar, Tolara
Malawi, Balaka
Malawi, Blantyre
Malawi, Chikwawa
Malawi, Chiradzulu
Malawi, Chilipa
Malawi, Dedza
Malawi, Dowa
Malawi, Karonga
Malawi, Kasungu
Malawi, Likoma
Malawi, Lilongwe
Malawi, Machinga
Malawi, Mangochi
Malawi, Mchinji
Malawi, Mulanje
Malawi, Mwanza
Malawi, Mzimba
Malawi, Nhaka Bay
Malawi, Nkhotakota
Malawi, Nsanje
Malawi, Ntcheu
Malawi, Ntchisi
Malawi, Phalombe
Malawi, Rumphi
Malawi, Salima
Malawi, Thyolo
Malawi, Zomba
Malaysia, Johor
Malaysia, Kedah
Malaysia, Kelantan
Malaysia, Kelantan
Malaysia, Kuala Lumpur
Malaysia, Labuan
Malaysia, Melaka
Malaysia, Negeri Sembilan
Malaysia, Pahang
Malaysia, Perak
Malaysia, Perlis
Malaysia, Pulau Pinang
Malaysia, Putrajaya
Malaysia, Sabah
Malaysia, Sarawak
Malaysia, Selangor
Malaysia, Terengganu
Maldives, Alif
Maldives, Baa
Maldives, Dhaalu
Maldives, Faafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Maldives, Gaafu
Mali, Bamako
Mali, Gao
Mali, Kayes
Mali, Kidj</p>
Mauritania, Assaba
Mauritania, Brakna
Mauritania, Dakhlet
Nouadhibou
Mauritania, Gorgol
Mauritania, Guidimaka
Mauritania, Hodh Ech Chargui
Mauritania, Hodh El Gharbi
Mauritania, Inchiri
Mauritania, Tagant
Mauritania, Tiris Zemmour
Mauritania, Trarza
Mauritius, Agalega Islands
Mauritius, Black River
Mauritius, Cargados Carajos
Mauritius, Flacq
Mauritius, Grand Port
Mauritius, Moka
Mauritius, Pamplemousses
Mauritius, Plaines Wilhems
Mauritius, Port Louis
Mauritius, Riviere du Rempart
Mauritius, Rodrigues
Mauritius, Savanne
Mexico, Aguascalientes
Mexico, Baja California
Mexico, Baja California Sur
Mexico, Campeche
Mexico, Chiapas
Mexico, Chihuahua
Mexico, Coahuila de Zaragoza
Mexico, Colima
Mexico, Distrito Federal
Mexico, Durango
Mexico, Guanajuato
Mexico, Guerrero
Mexico, Hidalgo
Mexico, Jalisco
Mexico, Mexico
Mexico, Michoacan de Ocampo
Mexico, Morelos
Mexico, Nayarit
Mexico, Nuevo Leon
Mexico, Oaxaca
Mexico, Puebla
Mexico, Queretaro de Arteaga
Mexico, Quintana
Roo
Mexico, San Luis
Potosi
Mexico, Sinaloa
Mexico, Sonora
Mexico, Tabasco
Mexico, Tamaulipas
Mexico, Tlaxcala
Mexico, Veracruz-Llave
Mexico, Yucatan
Mexico, Zacatecas
Micronesia, Chuuk
Micronesia, Kosrae
Micronesia, Pohnpei
Micronesia, Yap
Moldova, Republic of, Anenii Noi
Moldova, Republic of, Balti
Moldova, Republic of, Basarabeasca
Moldova, Republic of, Bender
Moldova, Republic of, Briceni
Moldova, Republic of, Cahul
Moldova, Republic of, Calarasi
Moldova, Republic of, Cantemir
Moldova, Republic of, Causeni
Moldova, Republic of, Chisinau
Moldova, Republic of, Cimișlia
Moldova, Republic of, Criuleni
Moldova, Republic of, Donduseni
Moldova, Republic of, Drochia
Moldova, Republic of, Dubasari
Moldova, Republic of, Edinet
Moldova, Republic of, Falesti
Moldova, Republic of, Florești
Moldova, Republic of, Gagauzia
Moldova, Republic of, Gîlău
Moldova, Republic of, Hincești
Moldova, Republic of, Ialoveni
Moldova, Republic of, Leova
Moldova, Republic of, Nisporeni
Moldova, Republic of, Ocnița
Moldova, Republic
<table>
<thead>
<tr>
<th>City</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rezina</td>
<td>Moldova, Republic</td>
</tr>
<tr>
<td>Riscani</td>
<td>Moldova, Republic</td>
</tr>
<tr>
<td>Singerei</td>
<td>Moldova, Republic</td>
</tr>
<tr>
<td>Soldanesti</td>
<td>Moldova, Republic</td>
</tr>
<tr>
<td>Soroca</td>
<td>Moldova, Republic</td>
</tr>
<tr>
<td>Stinga Nistrului</td>
<td>Moldova, Republic</td>
</tr>
<tr>
<td>Straseni</td>
<td>Moldova, Republic</td>
</tr>
<tr>
<td>Taraclia</td>
<td>Moldova, Republic</td>
</tr>
<tr>
<td>Stefan-Voda</td>
<td>Moldova, Republic</td>
</tr>
<tr>
<td>Telenesti</td>
<td>Moldova, Republic</td>
</tr>
<tr>
<td>Ungheni</td>
<td>Monaco, La</td>
</tr>
<tr>
<td>Condamine</td>
<td>Monaco, Monaco</td>
</tr>
<tr>
<td>Monte-Carlo</td>
<td>Monaco, Monte-Carlo</td>
</tr>
<tr>
<td>Arhangay</td>
<td>Mongolia, Bayan-Olgii</td>
</tr>
<tr>
<td>Bulgan</td>
<td>Mongolia, Bayanhongor</td>
</tr>
<tr>
<td>Darhan</td>
<td>Mongolia, Darhan</td>
</tr>
<tr>
<td>Dornod</td>
<td>Mongolia, Dornogovi</td>
</tr>
<tr>
<td>Dundgovi</td>
<td>Mongolia, Dundgovi</td>
</tr>
<tr>
<td>Dzavhan</td>
<td>Mongolia, Dzavhan</td>
</tr>
<tr>
<td>Erdenet</td>
<td>Mongolia, Erdenet</td>
</tr>
<tr>
<td>Govi-Altay</td>
<td>Mongolia, Govi-Altay</td>
</tr>
<tr>
<td>Govisumber</td>
<td>Mongolia, Govisumber</td>
</tr>
<tr>
<td>Hentiy</td>
<td>Mongolia, Hentiy</td>
</tr>
<tr>
<td>Hovd</td>
<td>Mongolia, Hovd</td>
</tr>
<tr>
<td>Hovsgol</td>
<td>Mongolia, Hovsgol</td>
</tr>
<tr>
<td>Omnogovi</td>
<td>Mongolia, Omnogovi</td>
</tr>
<tr>
<td>Orhon</td>
<td>Mongolia, Orhon</td>
</tr>
<tr>
<td>Ovorhangay</td>
<td>Mongolia, Ovorhangay</td>
</tr>
<tr>
<td>Selenge</td>
<td>Mongolia, Selenge</td>
</tr>
<tr>
<td>Suhbaatar</td>
<td>Mongolia, Suhbaatar</td>
</tr>
<tr>
<td>Tov</td>
<td>Mongolia, Tov</td>
</tr>
<tr>
<td>Ulaanbaatar</td>
<td>Mongolia, Ulaanbaatar</td>
</tr>
<tr>
<td>Uvs</td>
<td>Mongolia, Uvs</td>
</tr>
<tr>
<td>Montserrat, Saint Anthony</td>
<td>Montserrat, Saint Georges</td>
</tr>
<tr>
<td>Montserrat, Saint Peter</td>
<td>Montserrat, Saint Peter</td>
</tr>
<tr>
<td>Morocco</td>
<td>Montserrat, Saint Peter</td>
</tr>
<tr>
<td>Chaouia-Ouardigha</td>
<td>Montserrat, Saint Peter</td>
</tr>
</tbody>
</table>
Morocco,
Doukkala-Abda
Morocco,
Fes-Boulemane
Morocco,
Gharb-Chrarda-Beni Hssen
Morocco, Grand Casablanca
Morocco,
Guelmim-Es Smara
Morocco, La
Morocco,
Marrakech-Tensift-Al Haouz
Morocco,
Meknes-Tafilalet
Morocco, Oriental
Morocco,
Rabat-Sale-Zemmour-Zaer
Morocco,
Souss-Massa-Draa
Morocco,
Tadla-Azilal
Morocco,
Tanger-Tetouan
Morocco, Taza-Al Hoceima-Taounate
Mozambique, Cabo Delgado
Mozambique, Gaza
Mozambique, Inhambane
Mozambique, Manica
Mozambique, Maputo
Mozambique, Maputo
Mozambique, Nampula
Mozambique, Niassa
Mozambique, Sofala
Mozambique, Tete
Mozambique, Zambezia
Myanmar, Chin State
Myanmar, Irrawaddy
Myanmar, Kachin State
Myanmar, Karan State
Myanmar, Kayah State
Myanmar, Magwe
Myanmar, Mandalay
Myanmar, Mon State
Myanmar, Pegu
Myanmar, Rakhine State
Myanmar, Rangoon
Myanmar, Sagaing
Myanmar, Shan
State
Myanmar,
Tenasserim
Myanmar, Yangon

N - S
Namibia, Bethanië
Namibia,
Boesmanland
Namibia, Caprivi
Namibia, Caprivi
Oos
Namibia,
Damaraland
Namibia, Erongo
Namibia, Gobabis
Namibia,
Grootfontein
Namibia, Hardap
Namibia,
Hereroland Oos
Namibia,
Hereroland Wes
Namibia, Kaokoland
Namibia, Karas
Namibia, Karasburg
Namibia, Karibib
Namibia, Kavango
Namibia,
Keetmanshoop
Namibia, Kunene
Namibia, Luderitz
Namibia, Maltahohe
Namibia, Mariental
Namibia, Namaland
Namibia,
Ongwena
Namibia,
Okahandja
Namibia, Okavango
Namibia, Omaheke
Namibia, Omaruru
Namibia, Omusati
Namibia, Oshana
Namibia, Oshikoto
Namibia,
Otjiwarongo
Namibia,
Otjozondjupa
Namibia, Outjo
Namibia, Owambo
Namibia, Rehoboth
Namibia,
Swakopmund
Namibia, Tsumeb
Namibia, Windhoek
Nauru, Aiwo
Nauru, Anabar
Nauru, Anetan
Nauru, Anibare
Nauru, Bati
Nauru, Boe
Nauru, Buada
Nauru, Denigomodu
Nauru, Ewa
Nauru, Ijuw
Nauru, Meneng
Nauru, Nibok
Nauru, Uaboe
Nauru, Yaren
Nepal, Bagmati
Nepal, Bheri
Nepal, Dhaulagiri
Nepal, Gandaki
Nepal, Janakpur
Nepal, Karnali
Nepal, Kosi
Nepal, Lumbini
Nepal, Mahakali
Nepal, Mechi
Nepal, Narayani
Nepal, Rapti
Nepal, Sagarmatha
Nepal, Seti
Netherlands,
Drenthe
Netherlands,
Flevoland
Netherlands,
Friesland
Netherlands,
Gelderland
Netherlands,
Groningen
Netherlands,
Limburg
Netherlands,
Noord-Brabant
Netherlands,
Noord-Holland
Netherlands,
Overijssel
Netherlands,
Utrecht
Netherlands,
Zeeland
Netherlands,
Zuid-Holland
New Zealand,
Auckland
New Zealand, Bay of Plenty
New Zealand,
Canterbury
New Zealand,
Chatham Islands
New Zealand,
Gisborne
New Zealand,
Hawke's Bay
New Zealand,
Manawatu-Wanganui
New Zealand,
Marlborough
New Zealand,
Nelson
New Zealand,
Northland
New Zealand,
Otago
New Zealand,
Southland
New Zealand,
Taranaki
New Zealand,
Waikato
New Zealand,
Wellington
New Zealand, West Coast
Nicaragua,
Autonoma Atlantico Norte
Nicaragua, Boaco
Nicaragua, Carazo
Nicaragua,
Chinandega
Nicaragua, Chontales
Nicaragua, Esteli
Nicaragua, Granada
Nicaragua, Jinotega
Nicaragua, Leon
Nicaragua, Madriz
Nicaragua,
Managua
Nicaragua, Masaya
Nicaragua, Matagalpa
Nicaragua, Nueva Segovia
Nicaragua, Region Autonoma Atlantico Sur
Nicaragua, Rio San Juan
Nicaragua, Rivas
Nicaragua, Zelaya
Niger, Agadez
Niger, Diffa
Niger, Dosso
Niger, Maradi
Niger, Niamey
Niger, Niamey
Niger, Tahoua
Niger, Zinder
Nigeria, Abia
Nigeria, Adamawa
Nigeria, Akwa Ibom
Nigeria, Anambra
Nigeria, Bauchi
Nigeria, Bayelsa
Nigeria, Benue
Nigeria, Borno
Nigeria, Cross River
Nigeria, Delta
Nigeria, Ebonyi
Nigeria, Edo
Nigeria, Ekiti
Nigeria, Enugu
Nigeria, Federal Capital Territory
Nigeria, Gombe
Nigeria, Imo
Nigeria, Jigawa
Nigeria, Kaduna
Nigeria, Kano
Nigeria, Katsina
Nigeria, Kebbi
Nigeria, Kogi
Nigeria, Kwara
Nigeria, Lagos
Nigeria, Nassarawa
Nigeria, Niger
Nigeria, Ogun
Nigeria, Ondo
Nigeria, Osun
Nigeria, Oyo
Nigeria, Plateau
Nigeria, Rivers
Nigeria, Sokoto
Nigeria, Taraba
Nigeria, Yobe
Nigeria, Zamfara
Norway, Akershus
Norway, Aust-Agder
Norway, Buskerud
Norway, Finnmark
Norway, Hedmark
Norway, Hordaland
Norway, More og Romsdal
Norway, Nord-Trøndelag
Norway, Nordland
Norway, Oppland
Norway, Oslo
Norway, Ostfold
Norway, Rogaland
Norway, Sogn og Fjordane
Norway, Sor-Trøndelag
Norway, Telemark
Norway, Troms
Norway, Vest-Agder
Norway, Vestfold
Oman, Ad Dakhiliyah
Oman, Al Batinah
Oman, Al Wusta
Oman, Ash Sharqiyah
Oman, Az Zahirah
Oman, Masqat
Oman, Musandam
Oman, Zufar
Pakistan, Azad Kashmir
Pakistan, Balochistan
Pakistan, Federally Administered Tribal Areas
Pakistan, Islamabad
Pakistan, North-West Frontier Province
Pakistan, Northern Areas
Pakistan, Punjab
Pakistan, Sindh
Palestinian Territory, Occupied, Gaza
Palestinian Territory, Occupied,
West Bank
Panama, Bocas del Toro
Panama, Chiriqui
Panama, Cocle
Panama, Colon
Panama, Darien
Panama, Herrera
Panama, Los Santos
Panama, Panama
Panama, San Blas
Panama, Veraguas
Papua New Guinea, Central
Papua New Guinea, Chimbu
Papua New Guinea, East New Britain
Papua New Guinea, East Sepik
Papua New Guinea, Eastern Highlands
Papua New Guinea, Enga
Papua New Guinea, Gulf
Papua New Guinea, Madang
Papua New Guinea, Manus
Papua New Guinea, Milne Bay
Papua New Guinea, Morobe
Papua New Guinea, National Capital
Papua New Guinea, New Ireland
Papua New Guinea, North Solomons
Papua New Guinea, Northern
Papua New Guinea, Sandaun
Papua New Guinea, Southern Highlands
Papua New Guinea, West New Britain
Papua New Guinea, Western
Paraguay, Alto
Paraguay
Paraguay, Alto
Parana
Paraguay, Amambay
Paraguay, Boqueron
Paraguay, Caaguazu
Paraguay, Caazapa
Paraguay, Canindeyu
Paraguay, Central
Paraguay, Chaco
Paraguay,
Concepcion
Paraguay,
Cordillera
Paraguay, Guair¨a
Paraguay, Itapua
Paraguay, Misiones
Paraguay,
Neembuc¨u
Paraguay, Nueva
Asuncion
Paraguay,
Paraguari
Paraguay,
Presidente Hayes
Paraguay, San
Pedro
Peru, Amazonas
Peru, Ancash
Peru, Apurimac
Peru, Arequipa
Peru, Ayacucho
Peru, Cajamarca
Peru, Callao
Peru, Cusco
Peru, Huanacavelica
Peru, Huanuco
Peru, Ica
Peru, Junin
Peru, La Libertad
Peru, Lambayeque
Peru, Lima
Peru, Loreto
Peru, Madre de
Dios
Peru, Moquegua
Peru, Pasco
Peru, Piura
Peru, Puno
Peru, San Martin
Peru, Tacna
Peru, Tumbes
Peru, Ucayali
Philippines, Abra
Philippines, Agusan
del Norte
Philippines, Agusan
del Sur
Philippines, Aklan
Philippines, Albay
Philippines, Angeles
Philippines, Antique
Philippines, Aurora
Philippines, Bacolod
Philippines, Bago
Philippines, Baguio
Philippines, Bais
Philippines, Basilan
Philippines, Basilan
City
Philippines, Bataan
Philippines,
Batanes
Philippines,
Batangas
Philippines,
Batangas City
Philippines,
Benguet
Philippines, Bohol
Philippines,
Bukidnon
Philippines, Bulacan
Philippines, Butuan
Philippines,
Cabanatuan
Philippines, Cadiz
Philippines,
Cagayan
Philippines,
Cagayan de Oro
Philippines,
Calbayog
Philippines,
Caloocan
Philippines,
Camarines Norte
Philippines,
Camarines Sur
Philippines,
Camiguin
Philippines,
Canlaon
Philippines, Capiz
Philippines,
Catanduanes
Philippines, Cavite
Philippines, Cavite
City
Philippines, Cebu
Philippines, Cebu
City
Philippines,
Cotabato
Philippines,
Dagupan
Philippines, Davao
Philippines, Davao
City
Philippines, Davao
del Sur
Philippines, Davao
Oriental
Philippines, Dipolog
Philippines,
Dumaguete
Philippines, Eastern
Samar
Philippines, General
Santos
Philippines,
Gingoog
Philippines, Ifugao
Philippines, Iligan
Philippines, Ilocos
Norte
Philippines, Ilocos
Sur
Philippines, Iloilo
Philippines, Iloilo
City
Philippines, Iriga
Philippines, Isabela
Philippines,
Kalinga-Apayao
Philippines, La
Carloeta
Philippines, La
Union
Philippines, Laguna
Philippines, Lanao
del Norte
Philippines, Lanao
del Sur
Philippines, Laoag
Philippines,
Lapu-Lapu
Philippines, Legaspi
Philippines, Leyte
Philippines, Lipa
Philippines, Lucena
Philippines,
Maguindanao
Philippines,
Mandaue
Philippines, Manila
Philippines, Marawi
Philippines,
Marinduque
Philippines,
Masbate
Philippines, Mindoro
Occidental
Philippines, Mindoro
Oriental
Philippines,
Misamis Occidental
Philippines,
Misamis Oriental
Philippines,
Mountain
Philippines, Naga
Philippines, Negros
Occidental
Philippines, Negros
Occidental
Philippines, Negros
Oriental
Philippines, North
Cotabato
Philippines,
Northern Samar
Philippines, Nueva
Ecija
Philippines, Nueva
Vizcaya
Philippines,
Olongapo
Philippines, Ormoc
Philippines,
Oroquieta
Philippines, Ozamis
Philippines,
Pagadian
Philippines,
Palawan
Philippines, Palayan
Philippines,
Pampanga
Philippines,
Pangasinan
Philippines, Pasay
Philippines, Puerto Princesa
Philippines, Quezon
City
Philippines, Quirino
Philippines, Rizal
Philippines, Romblon
Philippines, Roxas
Philippines, Samar
Philippines, San Carlos
Philippines, San Carlos
Philippines, San Jose
Philippines, San Pablo
Philippines, Silay
Philippines, Siquijor
Philippines, Sorsogon
Philippines, South Cotabato
Philippines, Southern Leyte
Philippines, Sultan Kudarat
Philippines, Sulu
Philippines, Surigao
Philippines, Surigao del Norte
Philippines, Surigao del Sur
Philippines, Tacloban
Philippines, Tagaytay
Philippines, Tagbilaran
Philippines, Tangub
Philippines, Tarlac
Philippines, Tawi-Tawi
Philippines, Toledo
Philippines, Trece Martires
Philippines, Zambales
Philippines, Zamboanga
Philippines, Zamboanga del Norte
Philippines, Zamboanga del Sur
Poland, Dolnoslaskie
Poland, Kujawsko-Pomorskie
Poland, Lodzkie
Poland, Lubelskie
Poland, Lubuskie
Poland, Malopolskie
Poland, Mazowieckie
Poland, Opolskie  
Poland, Podkarpackie  
Poland, Podlaskie  
Poland, Pomorskie  
Poland, Slaskie  
Poland, Swietokrzyskie  
Poland, Warminsko-Mazurskie  
Poland, Wielkopolskie  
Poland, Zachodniopomorskie  
Portugal, Aveiro  
Portugal, Azores  
Portugal, Beja  
Portugal, Braga  
Portugal, Braganca  
Portugal, Castelo Branco  
Portugal, Coimbra  
Portugal, Evora  
Portugal, Faro  
Portugal, Guarda  
Portugal, Leiria  
Portugal, Lisboa  
Portugal, Madeira  
Portugal, Portalegre  
Portugal, Porto  
Portugal, Santarem  
Portugal, Setubal  
Portugal, Viana do Castelo  
Portugal, Vila Real  
Portugal, Viseu  
Qatar, Ad Dawhah  
Qatar, Al Ghuwariyah  
Qatar, Al Jumaliyah  
Qatar, Al Khawr  
Qatar, Al Wakrah  
Qatar, Al Wakrah Municipality  
Qatar, Ar Rayyan  
Qatar, Jariyan al Batnah  
Qatar, Madinat ach Shamal  
Qatar, Umm Sa'id  
Qatar, Umm Salal  
Romania, Alba  
Romania, Arad  
Romania, Arges  
Romania, Bacau  
Romania, Bihor  
Romania, Bistrita-Nasaud  
Romania, Botosani  
Romania, Braila  
Romania, Brasov  
Romania, Bucuresti  
Romania, Buzau  
Romania, Calarasi  
Romania, Caras-Severin
Romania, Cluj
Romania, Constanta
Romania, Covasna
Romania, Dambovita
Romania, Dolj
Romania, Galati
Romania, Giurgiu
Romania, Gorj
Romania, Harghita
Romania, Hunedoara
Romania, Ialomita
Romania, Iasi
Romania, Ilfov
Romania, Maramures
Romania, Mehedinți
Romania, Mures
Romania, Neamt
Romania, Olt
Romania, Prahova
Romania, Salaj
Romania, Satu Mare
Romania, Sibiu
Romania, Suceava
Romania, Teleorman
Romania, Timis
Romania, Tulcea
Romania, Valcea
Romania, Vaslui
Romania, Vrancea
Russian Federation, Adygeya
Russian Federation, Aginsky Buryatsky AO
Russian Federation, Amur
Russian Federation, Arkhangelsk
Russian Federation, Astrakhan’
Russian Federation, Bashkortostan
Russian Federation, Belgorod
Russian Federation, Bryansk
Russian Federation, Buryat
Russian Federation, Chechnya
Russian Federation, Chechnya Republic
Russian Federation, Chelyabinsk
Russian Federation, Chita
Russian Federation, Chukot
Russian Federation, Chuvashia
Russian Federation,
Dagestan
Russian Federation,
Evenk
Russian Federation,
Gorno-Altay
Russian Federation,
Ingush
Russian Federation,
Irkutsk
Russian Federation,
Ivanovo
Russian Federation,
Kabardino-Balkar
Russian Federation,
Kaliningrad
Russian Federation,
Kalmyk
Russian Federation,
Kaluga
Russian Federation,
Kamchatka
Russian Federation,
Karelia
Russian Federation,
Kemerovo
Russian Federation,
Khabarovsk
Russian Federation,
Khakass
Russian Federation,
Khanty-Mansi
Russian Federation,
Kirov
Russian Federation,
Komi
Russian Federation,
Komi-Permyak
Russian Federation,
Koryak
Russian Federation,
Kostroma
Russian Federation,
Krasnodar
Russian Federation,
Krasnoyarsk
Russian Federation,
Krasnoyarsk Krai
Russian Federation,
Kurgan
Russian Federation,
Kursk
Russian Federation,
Leningrad
Russian Federation,
Lipetsk
Russian Federation,
Magadan
Russian Federation,
Mariy-El
Russian Federation,
Mordovia
Russian Federation,
Moscow City
Russian Federation,
Moskva
Russian Federation, Murmansk
Russian Federation, Nenets
Russian Federation, Nizhegorod
Russian Federation, North Ossetia
Russian Federation, Novgorod
Russian Federation, Novosibirsk
Russian Federation, Omsk
Russian Federation, Orel
Russian Federation, Orenburg
Russian Federation, Penza
Russian Federation, Perm'
Russian Federation, Perm'skiy Kray
Russian Federation, Primor'ye
Russian Federation, Pskov
Russian Federation, Rostov
Russian Federation, Ryazan'
Russian Federation, Saint Petersburg
City
Russian Federation, Sakha
Russian Federation, Sakhalin
Russian Federation, Samara
Russian Federation, Saratov
Russian Federation, Smolensk
Russian Federation, Stavropol'
Russian Federation, Sverdlovsk
Russian Federation, Tambovskaya oblast
Russian Federation, Tatarstan
Russian Federation, Taymyr
Russian Federation, Tomsk
Russian Federation, Tula
Russian Federation, Tuva
Russian Federation, Tver'
Russian Federation, Tyumen'
Russian Federation, Udmurt
Nevis, Saint
Thomas Lowland
Saint Kitts and
Nevis, Saint
Thomas Middle
Island
Saint Kitts and
Nevis, Trinity
Palmetto Point
Saint Lucia,
Anse-la-Raye
Saint Lucia,
Castries
Saint Lucia,
Choiseul
Saint Lucia,
Dauphin
Saint Lucia,
Dennery
Saint Lucia,
Gros-Islet
Saint Lucia, Laborie
Saint Lucia, Micoud
Saint Lucia, Praslin
Saint Lucia,
Soufriere
Saint Lucia,
Vieux-Fort
Saint Vincent and
the Grenadines,
Charlotte
Saint Vincent and
the Grenadines,
Grenadines
Saint Vincent and
the Grenadines,
Saint Andrew
Saint Vincent and
the Grenadines,
Saint David
Saint Vincent and
the Grenadines,
Saint George
Saint Vincent and
the Grenadines,
Saint Patrick
Samoa,
Aiga-i-le-Tai
Samoa, Atua
Samoa, Fa
Samoa, Gaga
Samoa,
Gagaifomauga
Samoa, Palauli
Samoa, Satupa
Samoa, Tuamasaga
Samoa, Va
Samoa, Vaisigano
San Marino,
Acquaviva
San Marino, Borgo
Maggiore
San Marino,
Chiesanuova
San Marino,
Domagnano
San Marino,
Faetano
San Marino, Fiorentino
San Marino, Monte
Giardino
San Marino, San Marino
San Marino, Serravalle
Sao Tome and Principe, Principe
Sao Tome and Principe, Sao Tome
Saudi Arabia, Al Bahah
Saudi Arabia, Al Hudud ash Shamaliyah
Saudi Arabia, Al Jawf
Saudi Arabia, Al Jawf
Saudi Arabia, Al Madinah
Saudi Arabia, Al Qasim
Saudi Arabia, Al Qurayyat
Saudi Arabia, Ar Riyad
Saudi Arabia, Ash Sharqiyah
Saudi Arabia, Ha’il
Saudi Arabia, Jizan
Saudi Arabia, Makka
Saudi Arabia, Najran
Saudi Arabia, Tabuk
Senegal, Dakar
Senegal, Diourbel
Senegal, Fatick
Senegal, Kaolack
Senegal, Kolda
Senegal, Louga
Senegal, Matam
Senegal, Saint-Louis
Senegal, Tambacounda
Senegal, Thies
Senegal, Ziguinchor
Serbia, Kosovo
Serbia, Vojvodina
Seychelles, Anse aux Pins
Seychelles, Anse Boileau
Seychelles, Anse Etoile
Seychelles, Anse Louis
Seychelles, Anse Royale
Seychelles, Baie Lazare
Seychelles, Baie Sainte Anne
Seychelles, Beau Vallon
Seychelles, Bel Air
Seychelles, Bel Ombre
Seychelles, Cascade
Seychelles, Glacis
Seychelles, Grand' Anse
Seychelles, Grand' Anse
Seychelles, La Digue
Seychelles, La Rivière Anglaise
Seychelles, Mont Buxton
Seychelles, Mont Fleuri
Seychelles, Plaisance
Seychelles, Pointe La Rue
Seychelles, Port Glaud
Seychelles, Saint Louis
Seychelles, Takamaka
Sierra Leone, Eastern
Sierra Leone, Northern
Sierra Leone, Southern
Sierra Leone, Western Area
Slovakia, Banska Bystrica
Slovakia, Bratislava
Slovakia, Kosice
Slovakia, Nitra
Slovakia, Presov
Slovakia, Trencin
Slovenia, Trnava
Slovenia, Zilina
Slovenia, Ajdovscina
Slovenia, Beltinci
Slovenia, Bled
Slovenia, Bohinj
Slovenia, Borovnica
Slovenia, Bovec
Slovenia, Brda
Slovenia, Brezice
Slovenia, Brezovica
Slovenia, Celje
Slovenia, Cerkljne na Gorenjskem
Slovenia, Cerknica
Slovenia, Cerkno
Slovenia, Crenovci
Slovenia, Crna na Koroskem
Slovenia, Crnomelj
Slovenia, Divaca
Slovenia,
Dobrepolje, Slovenia, Dobrova-Horjul-Polhov Gradec, Slovenia, Dol pri Ljubljani, Slovenia, Domzale, Slovenia, Dornava, Slovenia, Dravograd, Slovenia, Duplek, Slovenia, Gorenja Vas-Poljane, Slovenia, Gorisnica, Slovenia, Gornja Radgona, Slovenia, Gornji Grad, Slovenia, Gornji Petrovci, Slovenia, Grosuplje, Slovenia, Hrastnik, Slovenia, Hrpelje-Kozina, Slovenia, Idrija, Slovenia, Ig, Slovenia, Ilirska Bistrica, Slovenia, Ivancna Gorica, Slovenia, Izola-Isola, Slovenia, Jesenice, Slovenia, Jursinci, Slovenia, Kamnik, Slovenia, Kanal, Slovenia, Kidricevo, Slovenia, Kobarid, Slovenia, Kobilje, Slovenia, Kocevje, Slovenia, Komen, Slovenia, Koper-Capodistria, Slovenia, Kozje, Slovenia, Kranj, Slovenia, Kranjska Gora, Slovenia, Krsko, Slovenia, Kungota, Slovenia, Kuzma, Slovenia, Lasko, Slovenia, Lenart, Slovenia, Litija, Slovenia, Ljubljana, Slovenia, Ljubno, Slovenia, Ljutomer, Slovenia, Logatec, Slovenia, Loska Dolina, Slovenia, Loski Potok, Slovenia, Luce, Slovenia, Lukovica, Slovenia, Majsperk, Slovenia, Maribor, Slovenia, Medvode, Slovenia, Menges, Slovenia, Metlika, Slovenia, Mezica
Slovenia, Miren-Kostanjevica
Slovenia, Mislinja
Slovenia, Moravce
Slovenia, Moravske Toplice
Slovenia, Mozirje
Slovenia, Murska Sobota
Slovenia, Muta
Slovenia, Naklo
Slovenia, Nazarje
Slovenia, Nova Gorica
Slovenia, Novo Mesto
Slovenia, Odranci
Slovenia, Ormoz
Slovenia, Osilnica
Slovenia, Pesnica
Slovenia, Piran
Slovenia, Pivka
Slovenia, Podcetrtek
Slovenia, Postojna
Slovenia, Preddvor
Slovenia, Ptuj
Slovenia, Puconci
Slovenia, Racam
Slovenia, Radece
Slovenia, Radenci
Slovenia, Radlje ob Dravi
Slovenia, Radovljica
Slovenia, Ribnica
Slovenia, Rogaska Slatina
Slovenia, Rogasovci
Slovenia, Rogatec
Slovenia, Ruse
Slovenia, Semic
Slovenia, Sencur
Slovenia, Sentij
Slovenia, Sentjernej
Slovenia, Sentjur pri Celju
Slovenia, Sevnica
Slovenia, Sezana
Slovenia, Skocjan
Slovenia, Skofja Loka
Slovenia, Skofljica
Slovenia, Slovenj Gradec
Slovenia, Slovenska Bistrica
Slovenia, Slovenske Konjice
Slovenia, Smarje pri Jelšah
Slovenia, Smartno ob Paki
Slovenia, Sostanj
Slovenia, Starše
Slovenia, Store
Slovenia, Sveti Jurij
Slovenia, Tolmin
Slovenia, Trbovlje
Slovenia, Trebnje
Slovenia, Trzic
Slovenia, Turnisce
Slovenia, Velenje
Slovenia, Velike Lasce
Slovenia, Videm
Slovenia, Vipava
Slovenia, Vitanje
Slovenia, Vodice
Slovenia, Vojnik
Slovenia, Vrhnika
Slovenia, Vuzenica
Slovenia, Zagorje ob Savi
Slovenia, Zalec
Slovenia, Zavrc
Slovenia, Zelezniki
Slovenia, Ziri
Slovenia, Zrece
Solomon Islands, Central
Solomon Islands, Choiseul
Solomon Islands, Guadalcanal
Solomon Islands, Isabel
Solomon Islands, Makira
Solomon Islands, Malaita
Solomon Islands, Rennell and Bellona
Solomon Islands, Temotu
Solomon Islands, Western
Somalia, Awdal
Somalia, Bakool
Somalia, Banaadir
Somalia, Bari
Somalia, Bay
Somalia, Galguduud
Somalia, Gedo
Somalia, Hiiraan
Somalia, Jubbada Dhexe
Somalia, Jubbada Hoose
Somalia, Shabeellaha Hoose
Somalia, Sool
Somalia, Togdheer
Somalia, Woqooyi Galbeed
Somalia, Woqooyi Galbeed
South Africa, Eastern Cape
South Africa, Free State
South Africa, Gauteng
South Africa, KwaZulu-Natal
South Africa, Limpopo
South Africa, Mpumalanga
South Africa, North-West
South Africa, Northern Cape
South Africa, Western Cape
Spain, Andalucia
Spain, Aragon
Spain, Asturias
Spain, Canarias
Spain, Cantabria
Spain, Castilla y Leon
Spain, Castilla-La Mancha
Spain, Catalonia
Spain, Comunidad Valenciana
Spain, Extremadura
Spain, Galicia
Spain, Islas Baleares
Spain, La Rioja
Spain, Madrid
Spain, Murcia
Spain, Navarra
Spain, Pais Vasco
Sri Lanka, Amparai
Sri Lanka, Anuradhapura
Sri Lanka, Badulla
Sri Lanka, Batticaloa
Sri Lanka, Central
Sri Lanka, Colombo
Sri Lanka, Galle
Sri Lanka, Gampaha
Sri Lanka, Hambantota
Sri Lanka, Jaffna
Sri Lanka, Kalutara
Sri Lanka, Kandy
Sri Lanka, Kegalla
Sri Lanka, Kurunegala
Sri Lanka, Mannar
Sri Lanka, Matale
Sri Lanka, Matara
Sri Lanka, Moneragala
Sri Lanka, Mullaitivu
Sri Lanka, North Central
Sri Lanka, North Western
Sri Lanka, Northern
Sri Lanka, Nuwara Eliya
Sri Lanka, Polonnaruwa
Sri Lanka, Puttalam
Sri Lanka, Ratnapura
Sri Lanka, Sabaragamuwa
Sri Lanka, Southern
Sri Lanka, Trincomalee
Sri Lanka, Uva
Sri Lanka, Vavuniya
Sri Lanka, Western
Sudan, Al Istiwa'iyah
Sudan, Al Khartum
Sudan, Al Wahadah State
Sudan, Al Wusta
Sudan, Ash Shamiyeh
Sudan, Ash Sharqiyah
Sudan, Bahr al Ghazal
Sudan, Central Equatoria State
Sudan, Darfur
Sudan, Kurdushan
Sudan, Upper Nile
Suriname, Brokopondo
Suriname, Commewijne
Suriname, Coronie
Suriname, Marowijne
Suriname, Nickerie
Suriname, Para
Suriname, Paramaribo
Suriname, Saramacca
Suriname, Sipaliwini
Suriname, Wanica
Swaziland, Hhohho
Swaziland, Lubombo
Swaziland, Manzini
Swaziland, Praslin
Swaziland, Shiselweni
Sweden, Blekinge Lan
Sweden, Dalarnas Lan
Sweden, Gavleborgs Lan
Sweden, Gotlands Lan
Sweden, Hallands Lan
Sweden, Jamtlands
Lan
Sweden,
Jonkopings Lan
Sweden, Kalmar
Lan
Sweden,
Kronobergs Lan
Sweden,
Norrbottens Lan
Sweden, Orebro
Lan
Sweden,
Ostergotlands Lan
Sweden, Skane Lan
Sweden,
Sodermanlands Lan
Sweden,
Stockholms Lan
Sweden, Uppsala
Lan
Sweden, Varmlands Lan
Sweden,
Vasterbottens Lan
Sweden,
Vasternorrlands Lan
Sweden,
Vastmanlands Lan
Sweden, Vastra
Gotaland
Switzerland, Aargau
Switzerland,
Ausser-Rhoden
Switzerland,
Basel-Landschaft
Switzerland,
Basel-Stadt
Switzerland, Bern
Switzerland,
Fribourg
Switzerland,
Geneve
Switzerland, Glarus
Switzerland,
Graubunden
Switzerland,
Inner-Rhoden
Switzerland, Jura
Switzerland, Luzern
Switzerland,
Neuchatel
Switzerland,
Nidwalden
Switzerland,
Obwalden
Switzerland, Sankt
Gallen
Switzerland,
Schaffhausen
Switzerland,
Schwyz
Switzerland,
Solothurn
Switzerland,
Thurgau
Switzerland, Ticino
Switzerland, Uri
Switzerland, Valais
Switzerland, Vaud
Switzerland, Zug
Switzerland, Zurich
Syrian Arab Republic, Al Hasakah
Syrian Arab Republic, Al Ladhiqiyah
Syrian Arab Republic, Al Qunaytrah
Syrian Arab Republic, Ar Raqqah
Syrian Arab Republic, As Suwayda’
Syrian Arab Republic, Dar al Qunaytirah
Syrian Arab Republic, Dayr az Zawr
Syrian Arab Republic, Dimashq
Syrian Arab Republic, Halab
Syrian Arab Republic, Hamah
Syrian Arab Republic, Idlib
Syrian Arab Republic, Rif Dimashq
Syrian Arab Republic, Tartus

T-Z
Taiwan, Fu-chien
Taiwan, Kao-hsiung
Taiwan, T'ai-pei
Taiwan, T'ai-wan
Tajikistan, Khatlon
Tajikistan, Kuhistan
Badakhshan
Tajikistan, Sughd
Tanzania, Arusha
Tanzania, Dar es Salaam
Tanzania, Dodoma
Tanzania, Iringa
Tanzania, Kagera
Tanzania, Kigoma
Tanzania, Kilimanjaro
Tanzania, Lindi
Tanzania, Manyara
Tanzania, Mara
Tanzania, Mbeya
Tanzania, Morogoro
Tanzania, Mtwara
Tanzania, Mwanza
Tanzania, Pemba North
Tanzania, Pemba

Copyright © AppDynamics 2012-2019
South
Tanzania, Pwani
Tanzania, Rukwa
Tanzania, Ruvuma
Tanzania, Shinyanga
Tanzania, Singida
Tanzania, Tabora
Tanzania, Tanga
Tanzania, Zanzibar
Central
Tanzania, Zanzibar
North
Tanzania, Zanzibar
Urban
Thailand, Amnat Charoen
Thailand, Ang Thong
Thailand, Buriram
Thailand, Chachoengsao
Thailand, Chai Nat
Thailand, Chaiyaphum
Thailand, Chanthaburi
Thailand, Chiang Mai
Thailand, Chiang Rai
Thailand, Chon Buri
Thailand, Chumphon
Thailand, Kalasin
Thailand, Kamphaeng Phet
Thailand, Kanchanaburi
Thailand, Khon Kaen
Thailand, Krabi
Thailand, Krung Thep
Thailand, Lampang
Thailand, Lamphun
Thailand, Loei
Thailand, Lop Buri
Thailand, Mae Hong Son
Thailand, Maha Sarakham
Thailand, Mukdahan
Thailand, Nakhon Nayok
Thailand, Nakhon Pathom
Thailand, Nakhon Phanom
Thailand, Nakhon Phanom
Thailand, Nakhon Ratchasima
Thailand, Nakhon Sawan
Thailand, Nakhon Si Thammarat
Thailand, Nan
Thailand, Narathiwat
Thailand, Nong Bua Lamphu
Thailand, Nong Khai
Thailand, Nonthaburi
Thailand, Pathum Thani
Thailand, Pattani
Thailand, Phangnga
Thailand, Phatthalung
Thailand, Phayao
Thailand, Phetchabun
Thailand, Phetchaburi
Thailand, Phichit
Thailand, Phitsanulok
Thailand, Phra Nakhon Si Ayutthaya
Thailand, Phrae
Thailand, Phuket
Thailand, Prachin Buri
Thailand, Prachuap Khiri Khan
Thailand, Ranong
Thailand, Ratchaburi
Thailand, Rayong
Thailand, Roi Et
Thailand, Sa Kaeo
Thailand, Sakon
Nakhon
Thailand, Samut Prakan
Thailand, Samut Sakhon
Thailand, Samut Songkhram
Thailand, Saraburi
Thailand, Satun
Thailand, Sing Buri
Thailand, Sisaket
Thailand, Songkhla
Thailand, Sukhothai
Thailand, Suphan Buri
Thailand, Surat Thani
Thailand, Surin
Thailand, Tak
Thailand, Trang
Thailand, Trat
Thailand, Ubon Ratchathani
Thailand, Ubon Ratchathani
Thailand, Udon Thani
Thailand, Uthai Thani
Thailand, Uttaradit
Thailand, Yala
Thailand, Yasothon
The Bahamas,
Acklins and
Crooked Islands
The Bahamas,
Bimini
The Bahamas, Cat Island
The Bahamas, Exuma
The Bahamas, Freeport
The Bahamas, Fresh Creek
The Bahamas,
Governor's Harbour
The Bahamas, Green Turtle Cay
The Bahamas, Harbour Island
The Bahamas, High Rock
The Bahamas, Inagua
The Bahamas, Kemps Bay
The Bahamas, Long Island
The Bahamas, Marsh Harbour
The Bahamas, Mayaguana
The Bahamas, New Providence
The Bahamas, Nichollstown and Berry Islands
The Bahamas, Ragged Island
The Bahamas, Rock Sound
The Bahamas, San Salvador and Rum Cay
The Bahamas, Sandy Point
Togo, Centrale
Togo, Kara
Togo, Maritime
Togo, Plateaux
Togo, Savanes
Tonga, Ha
Tonga, Tongatapu
Tonga, Vava
Trinidad and Tobago, Arima
Trinidad and Tobago, Caroni
Trinidad and Tobago, Mayaro
Trinidad and Tobago, Nariva
Trinidad and Tobago,
Port-of-Spain
Trinidad and
<table>
<thead>
<tr>
<th>Country</th>
<th>City</th>
<th>Country</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turkey</td>
<td>Denizli</td>
<td>Turkmenistan</td>
<td>Ahal</td>
</tr>
<tr>
<td></td>
<td>Diyarbakir</td>
<td></td>
<td>Balkan</td>
</tr>
<tr>
<td></td>
<td>Duzce</td>
<td></td>
<td>Dashoguz</td>
</tr>
<tr>
<td></td>
<td>Edirne</td>
<td></td>
<td>Lebap</td>
</tr>
<tr>
<td></td>
<td>Elazig</td>
<td></td>
<td>Turkmkenistan</td>
</tr>
<tr>
<td></td>
<td>Erzincan</td>
<td></td>
<td>Turkmkenistan</td>
</tr>
<tr>
<td></td>
<td>Erzurum</td>
<td></td>
<td>Turkmkenistan</td>
</tr>
<tr>
<td></td>
<td>Eskisehir</td>
<td></td>
<td>Turkmkenistan</td>
</tr>
<tr>
<td></td>
<td>Gaziantep</td>
<td></td>
<td>Turkmkenistan, Mary</td>
</tr>
<tr>
<td></td>
<td>Giresun</td>
<td></td>
<td>Uganda, Adjumani</td>
</tr>
</tbody>
</table>
Uganda, Apac
Uganda, Arua
Uganda, Bugiri
Uganda,
Bundibugyo
Uganda, Bushenyi
Uganda, Busia
Uganda, Gulu
Uganda, Hoima
Uganda, Iganga
Uganda, Jinja
Uganda, Kabarole
Uganda,
Kaberamaido
Uganda, Kalangala
Uganda, Kampala
Uganda, Kamuli
Uganda,
Kamwenge
Uganda, Kanungu
Uganda,
Kapchorwa
Uganda, Kasese
Uganda, Katakiw
Uganda, Kayunga
Uganda, Kibale
Uganda, Kiboga
Uganda, Kisoro
Uganda, Kitgum
Uganda, Kotido
Uganda, Kumi
Uganda, Kyenjojo
Uganda, Lira
Uganda, LuwERO
Uganda, Masaka
Uganda, Masindi
Uganda, Mayuge
Uganda, Mbale
Uganda, Mbarara
Uganda, Moroto
Uganda, Moyo
Uganda, Mpigi
Uganda, Mubende
Uganda, Mukono
Uganda,
Nakapiripirit
Uganda,
Nakasongola
Uganda, Nebbi
Uganda, Ntungamo
Uganda, Pader
Uganda, Pallisa
Uganda, Rakai
Uganda, Rukungiri
Uganda,
Sembabule
Uganda, Sironko
Uganda, Soroti
Uganda, Tororo
Uganda, Wakiso
Uganda, Yumbe
Ukraine, Cherkas'ka
Oblast'
Ukraine,
Chernihivs'ka
Oblast'
Ukraine,
Chernivets'ka
United Kingdom, Bury
United Kingdom, Caerphilly
United Kingdom, Calderdale
United Kingdom, Cambridgeshire
United Kingdom, Camden
United Kingdom, Cardiff
United Kingdom, Carmarthenshire
United Kingdom, Carlinfargus
United Kingdom, Castlereagh
United Kingdom, Ceredigion
United Kingdom, Cheshire
United Kingdom, Clackmannanshire
United Kingdom, Coleraine
United Kingdom, Conway
United Kingdom, Cookstown
United Kingdom, Cornwall
United Kingdom, Coventry
United Kingdom, Craigavon
United Kingdom, Croydon
United Kingdom, Cumbria
United Kingdom, Darlington
United Kingdom, Denbighshire
United Kingdom, Derby
United Kingdom, Derbyshire
United Kingdom, Derry
United Kingdom, Devon
United Kingdom, Doncaster
United Kingdom, Dorset
United Kingdom, Down
United Kingdom, Dudley
United Kingdom, Dumfries and Galloway
United Kingdom, Dundee City
United Kingdom, Dungannon
United Kingdom,
Durham
United Kingdom,
Ealing
United Kingdom,
East Ayrshire
United Kingdom,
East
Dunbartonshire
United Kingdom,
East Lothian
United Kingdom,
East Renfrewshire
United Kingdom,
East Riding of
Yorkshire
United Kingdom,
East Sussex
United Kingdom,
Edinburgh
United Kingdom,
Eilean Siar
United Kingdom,
Enfield
United Kingdom,
Essex
United Kingdom,
Falkirk
United Kingdom,
Fermanagh
United Kingdom,
Fife
United Kingdom,
Flintshire
United Kingdom,
Gateshead
United Kingdom,
Glasgow City
United Kingdom,
Gloucestershire
United Kingdom,
Greenwich
United Kingdom,
Gwynedd
United Kingdom,
Hackney
United Kingdom,
Harrow
United Kingdom,
Hartlepool
United Kingdom,
Havering
United Kingdom,
Herefordshire
United Kingdom,
Hertford
United Kingdom,
Highland
United Kingdom,
Hillingdon
United Kingdom, Hounslow
United Kingdom, Inverclyde
United Kingdom, Isle of Anglesey
United Kingdom, Isle of Wight
United Kingdom, Islington
United Kingdom, Kensington and Chelsea
United Kingdom, Kent
United Kingdom, Kingston upon Hull
United Kingdom, Kingston upon Thames
United Kingdom, Kirklees
United Kingdom, Knowsley
United Kingdom, Lambeth
United Kingdom, Lancashire
United Kingdom, Larne
United Kingdom, Leeds
United Kingdom, Leicester
United Kingdom, Leicestershire
United Kingdom, Lewisham
United Kingdom, Limavady
United Kingdom, Lincolnshire
United Kingdom, Lisburn
United Kingdom, Liverpool
United Kingdom, London
United Kingdom, Luton
United Kingdom, Magherafelt
United Kingdom, Manchester
United Kingdom, Medway
United Kingdom, Merthyr Tydfil
United Kingdom, Merton
United Kingdom, Middlesbrough
United Kingdom, Midlothian
United Kingdom, Milton Keynes
United Kingdom, Monmouthshire
United Kingdom, Moray
United Kingdom, Moyle
United Kingdom, Neath Port Talbot
United Kingdom, Newcastle upon Tyne
United Kingdom, Newham
United Kingdom, Newport
United Kingdom, Newry and Mourne
United Kingdom, Newtownabbey
United Kingdom, Norfolk
United Kingdom, North Ayrshire
United Kingdom, North Down
United Kingdom, North East Lincolnshire
United Kingdom, North Lanarkshire
United Kingdom, North Lincolnshire
United Kingdom, North Somerset
United Kingdom, North Tyneside
United Kingdom, North Yorkshire
United Kingdom, Northamptonshire
United Kingdom, Northumberland
United Kingdom, Nottingham
United Kingdom, Nottinghamshire
United Kingdom, Oldham
United Kingdom, Omagh
United Kingdom, Orkney
United Kingdom, Oxfordshire
United Kingdom, Pembrokeshire
United Kingdom, Perth and Kinross
United Kingdom, Peterborough
United Kingdom, Plymouth
United Kingdom, Poole
United Kingdom, Portsmouth
United Kingdom, Powys
United Kingdom, Reading
United Kingdom, Redbridge
United Kingdom, Redcar and Cleveland
United Kingdom, Renfrewshire
United Kingdom, Rhondda Cynon Taff
United Kingdom, Richmond upon Thames
United Kingdom, Rochdale
United Kingdom, Rotherham
United Kingdom, Rutland
United Kingdom, Salford
United Kingdom, Sandwell
United Kingdom, Scottish Borders
United Kingdom, Selton
United Kingdom, Sheffield
United Kingdom, Shetland Islands
United Kingdom, Shropshire
United Kingdom, Slough
United Kingdom, Solihull
United Kingdom, Somerset
United Kingdom, South Ayrshire
United Kingdom, South Gloucestershire
United Kingdom, South Lanarkshire
United Kingdom, South Tyneside
United Kingdom, Southampton
United Kingdom, Southend-on-Sea
United Kingdom, Southwark
United Kingdom, St. Helens
United Kingdom, Staffordshire
United Kingdom, Stirling
United Kingdom, Stockport
United Kingdom, Stockton-on-Tees
United Kingdom, Stoke-on-Trent
United Kingdom, Strabane
United Kingdom,
Suffolk
United Kingdom,
Sunderland
United Kingdom,
Surrey
United Kingdom,
Sutton
United Kingdom,
Swansea
United Kingdom,
Swindon
United Kingdom,
Tameside
United Kingdom,
Telford and Wrekin
United Kingdom,
Thurrock
United Kingdom,
Torbay
United Kingdom,
Torfaen
United Kingdom,
Tower Hamlets
United Kingdom,
Trafford
United Kingdom,
Vale of Glamorgan
United Kingdom,
Wakefield
United Kingdom,
Walsall
United Kingdom,
Waltham Forest
United Kingdom,
Wandsworth
United Kingdom,
Warrington
United Kingdom,
Warwickshire
United Kingdom,
West Berkshire
United Kingdom,
West
Dumbartonshire
United Kingdom,
West Lothian
United Kingdom,
West Sussex
United Kingdom,
Westminster
United Kingdom,
Wigan
United Kingdom,
Wiltshire
United Kingdom,
Windsor and
Maidenhead
United Kingdom,
Wirral
United Kingdom,
Wokingham
United Kingdom,
Wolverhampton
United Kingdom,
Worcestershire
United Kingdom,
Wrexham
United Kingdom,
York
United States,
Alabama
United States,
Alaska
United States,
American Samoa
United States,
Arizona
United States,
Arkansas
United States,
Armed Forces
Americas
United States,
Armed Forces
Europe
United States,
Armed Forces
Pacific
United States,
California
United States,
Colorado
United States,
Connecticut
United States,
Delaware
United States,
District of Columbia
United States,
Federated States of
Micronesia
United States,
Florida
United States,
Georgia
United States,
Guam
United States,
Hawaii
United States,
Idaho
United States,
Illinois
United States,
Indiana
United States, Iowa
United States,
Kansas
United States,
Kentucky
United States,
Louisiana
United States,
Maine
United States,
Marshall Islands
United States,
Maryland
United States,
Massachusetts
United States,
Michigan
United States,
Minnesota
United States,
Mississippi United States,
Missouri United States,
Montana United States,
Nebraska United States,
Nevada United States, New Hampshire
United States, New Jersey
United States, New Mexico
United States, New York
United States, North Carolina
United States, North Dakota
United States, Northern Mariana Islands
United States, Ohio United States, Oklahoma
United States, Oregon
United States, Palau
United States, Pennsylvania
United States, Puerto Rico
United States, Rhode Island
United States, South Carolina
United States, South Dakota
United States, Tennessee
United States, Texas
United States, Utah
United States, Vermont
United States, Virgin Islands
United States, Virginia
United States, Washington
United States, West Virginia
United States, Wisconsin
United States, Wyoming
Uruguay, Artigas
Uruguay, Canelones
Uruguay, Cerro Largo
Uruguay, Colonia
Uruguay, Durazno
Uruguay, Flores
Uruguay, Florida
Uruguay, Lavalleja
Uruguay, Maldonado
Uruguay, Montevideo
Uruguay, Paysandu
Uruguay, Rio Negro
Uruguay, Rivera
Uruguay, Rocha
Uruguay, Salto
Uruguay, San Jose
Uruguay, Soriano
Uruguay, Tacuarembo
Uruguay, Treinta y Tres
Uzbekistan, Andijon
Uzbekistan, Bukhoro
Uzbekistan, Farghona
Uzbekistan, Jizzakh
Uzbekistan, Khorazm
Uzbekistan, Namangan
Uzbekistan, Nawoiy
Uzbekistan, Qashqadaryo
Uzbekistan, Qoraqalpoghiston
Uzbekistan, Samarqand
Uzbekistan, Sirdaryo
Uzbekistan, Surkhondaryo
Uzbekistan, Toshkent
Uzbekistan, Toshkent
Vanuatu, Ambrym
Vanuatu, Aoba
Vanuatu, Efate
Vanuatu, Epi
Vanuatu, Malakula
Vanuatu, Malampa
Vanuatu, Paama
Vanuatu, Penama
Vanuatu, Pentecote
Vanuatu, Sanma
Vanuatu, Shefa
Vanuatu, Shepherd
Vanuatu, Tafea
Vanuatu, Torba
Venezuela, Amazonas
Venezuela, Anzoategui
Venezuela, Apure
Venezuela, Aragua
Venezuela, Barinas
Venezuela, Bolivar
Venezuela, Carabobo
Venezuela, Cojedes
Venezuela, Delta
Amacuro,
Venezuela,
Dependencias
Federales
Venezuela, Distrito
Federal
Venezuela, Falcon
Venezuela, Guarico
Venezuela, Lara
Venezuela, Merida
Venezuela, Miranda
Venezuela,
Monagas
Venezuela, Nueva
Esparta
Venezuela,
Portuguesa
Venezuela, Sucre
Venezuela, Tachira
Venezuela, Trujillo
Venezuela, Vargas
Venezuela, Yaracuy
Venezuela, Zulia
Vietnam, An Giang
Vietnam, An Giang
Vietnam, Ba
Ria-Vung Tau
Vietnam, Ben Tre
Vietnam, Binh Dinh
Vietnam, Binh
Thuan
Vietnam, Can Tho
Vietnam, Cao Bang
Vietnam, Da Nang
Vietnam, Dac Lac
Vietnam, Dak Lak
Vietnam, Dak Nong
Vietnam, Dien Bien
Vietnam, Dong Nai
Vietnam, Dong
Thap
Vietnam, Dong
Thap
Vietnam, Ha Giang
Vietnam, Ha Nam
Vietnam, Ha Noi
Vietnam, Ha Tay
Vietnam, Ha Tinh
Vietnam, Hai Duong
Vietnam, Hai Phong
Vietnam, Hau Giang
Vietnam, Ho Chi Minh
Vietnam, Ho Chi Minh
Vietnam, Hoa Binh
Vietnam, Hung Yen
Vietnam, Khanh
Hoa
Vietnam, Kien Giang
Vietnam, Kien Giang
Vietnam, Kon Tum
Vietnam, Lai Chau
Vietnam, Lam Dong
Vietnam, Lang Son
Vietnam, Lao Cai
Vietnam, Long An
Vietnam, Nam Dinh
Vietnam, Nam Ha
Vietnam, Nghe An
Vietnam, Ninh Binh
Vietnam, Ninh Thuan
Vietnam, Phu Tho
Vietnam, Phu Yen
Vietnam, Quang Binh
Vietnam, Quang Nam
Vietnam, Quang Ngai
Vietnam, Quang Ninh
Vietnam, Quang Tri
Vietnam, Quang Tri
Vietnam, Soc Trang
Vietnam, Son La
Vietnam, Song Be
Vietnam, Tay Ninh
Vietnam, Thai Binh
Vietnam, Thai Nguyen
Vietnam, Thanh Hoa
Vietnam, Thua Thien
Vietnam, Tien Giang
Vietnam, Tra Vinh
Vietnam, Tuyen Quang
Vietnam, Vinh Long
Vietnam, Vinh Phu
Vietnam, Vinh Puc Province
Yemen, Abyan
Yemen, Adan
Yemen, Al Bayda'
Yemen, Al Ghaydah
Yemen, Al Hudaydah
Yemen, Al Jawf
Yemen, Al Mahrah
Yemen, Al Mahwit
Yemen, Dhamar
Yemen, Hadramawt
Yemen, Hajjah
Yemen, Ibb
Yemen, Lahij
Yemen, Ma'rib
Yemen, Sa
Yemen, San
Yemen, Shabwah
Yemen, Ta
Zambia, Central
Zambia, Copperbelt
Zambia, Eastern
Zambia, Luapula
Zambia, Lusaka
Zambia, North-Western
Zambia, Northern
Zambia, Southern
Zambia, Western
Zimbabwe,
Bulawayo
Zimbabwe, Harare
Zimbabwe, Manicaland
Zimbabwe, Mashonaland Central
Zimbabwe, Mashonaland East
Zimbabwe, Mashonaland West
Zimbabwe, Masvingo
Zimbabwe, Matabeleland North
Zimbabwe, Matabeleland South
Zimbabwe, Midlands

Map View

The following countries displayed in the map view of the Geo Dashboard also report data by region.

A – G

Australia, Australian Capital Territory
Australia, New South Wales
Australia, Queensland
Australia, South Australia
Australia, Tasmania
Australia, Victoria
Australia, Western Australia
Australia
Belgium, Antwerpen
Belgium, Hainaut
Belgium, Liege
Belgium, Limburg
Belgium, Luxembourg
Belgium, Namur
Belgium, Oost-Vlaanderen
Belgium, West-Vlaanderen
Belgium, Brabant
Wallon
Belgium, Brussels Hoofdstedelijk Gewest
Belgium, Vlaams-Brabant
Belgium, Flanders
Belgium, Wallonia

Brazil, Acre
Brazil, Alagoas
Brazil, Amapa
Brazil, Amazonas
Brazil, Bahia
China, Jiangxi
China, Jilin
China, Liaoning
China, Nei Mongol
China, Ningxia
China, Qinghai
China, Shaanxi
China, Shandong
China, Shanghai
China, Shanxi
China, Sichuan
China, Tianjin
China, Xinjiang
China, Xizang
China, Yunnan
China, Zhejiang
France, Alsace
France, Aquitaine
France, Auvergne
France, Basse-Normandie
France, Bourgogne
France, Bretagne
France, Centre
France, Champagne-Ardenne
France, Corse
France, Franche-Comté
France, Haute-Normandie
France, Ile-de-France
France, Languedoc-Roussillon
France, Limousin
France, Lorraine
France, Midi-Pyrénées
France, Nord-Pas-de-Calais
France, Pays de la Loire
France, Picardie
France, Poitou-Charentes
France, Provence-Alpes-Côte d'Azur
France, Rhône-Alpes
Germany, Baden-Württemberg
Germany, Bayern
Germany, Berlin
Germany, Brandenburg
Germany, Bremen
Germany, Hamburg
Germany, Hessen
Germany, Mecklenburg-Vorpommern
Germany, Niedersachsen
Germany,
Nordrhein-Westfalen
Germany,
Rheinland-Pfalz
Germany, Saarland
Germany, Sachsen
Germany, Sachsen-Anhalt
Germany, Schleswig-Holstein
Germany, Thuringen

H – M

India, Andaman and Nicobar Islands
India, Andhra Pradesh
India, Arunachal Pradesh
India, Assam
India, Bihar
India, Chandigarh
India, Chhattisgarh
India, Dadra and Nagar Haveli
India, Daman and Diu
India, Delhi
India, Goa
India, Gujarat
India, Haryana
India, Himachal Pradesh
India, Jammu and Kashmir
India, Jharkhand
India, Karnataka
India, Kerala
India, Lakshadweep
India, Madhya Pradesh
India, Maharashtra
India, Manipur
India, Meghalaya
India, Mizoram
India, Nagaland
India, Orissa
India, Puducherry
India, Punjab
India, Rajasthan
India, Sikkim
India, Tamil Nadu
India, Tripura
India, Uttar Pradesh
India, Uttarakhand
India, West Bengal

Indonesia, Aceh
Indonesia, Bali
Indonesia, Banten
Indonesia, Bengkulu
Indonesia, Gorontalo
Indonesia, Irian
Jaya Barat
Indonesia, Jakarta Raya
Indonesia, Jambi
Indonesia, Jawa Barat
Indonesia, Jawa Tengah
Indonesia, Jawa Timur
Indonesia, Kalimantan Barat
Indonesia, Kalimantan Selatan
Indonesia, Kalimantan Tengah
Indonesia, Kalimantan Timur
Indonesia, Kepulauan Bangka Belitung
Indonesia, Kepulauan Riau
Indonesia, Lampung
Indonesia, Maluku
Indonesia, Maluku Utara
Indonesia, Nusa Tenggara Barat
Indonesia, Nusa Tenggara Timur
Indonesia, Papua
Indonesia, Riau
Indonesia, Sulawesi Barat
Indonesia, Sulawesi Selatan
Indonesia, Sulawesi Tengah
Indonesia, Sulawesi Tenggara
Indonesia, Sulawesi Utara
Indonesia, Sumatera Barat
Indonesia, Sumatera Selatan
Indonesia, Sumatera Utara
Indonesia, Yogyakarta
Italy, Abruzzi
Italy, Aosta Valley
Italy, Basilicata
Italy, Calabria
Italy, Campania
Italy, Corsica (Corse) - France
Italy, Emilia-Romagna
Italy, Friuli-Venezia Giulia
Italy, Lazio
Italy, Liguria
Italy, Lombardia
Italy, Malta
Italy, Marche
Italy, Molise
Italy, Piemonte
Italy, Puglia
Italy, San Marino
Italy, Sardinia
Italy, Sicily
Italy, Tuscany
Italy, Trentino-Alto Adige
Italy, Umbria
Italy, Vatican City (Holy See)
Italy, Veneto

Japan, Aichi
Japan, Akita
Japan, Aomori
Japan, Chiba
Japan, Ehime
Japan, Fukui
Japan, Fukuoka
Japan, Fukushima
Japan, Gifu
Japan, Gunma
Japan, Hiroshima
Japan, Hokkaido
Japan, Hyogo
Japan, Ibaraki
Japan, Ishikawa
Japan, Iwate
Japan, Kagawa
Japan, Kagoshima
Japan, Kanagawa
Japan, Kochi
Japan, Kumamoto
Japan, Kyoto
Japan, Mie
Japan, Miyagi
Japan, Miyazaki
Japan, Nagano
Japan, Nagasaki
Japan, Nara
Japan, Niigata
Japan, Oita
Japan, Okayama
Japan, Okinawa
Japan, Osaka
Japan, Saga
Japan, Saitama
Japan, Shiga
Japan, Shimane
Japan, Shizuoka
Japan, Tochigi
Japan, Tokushima
Japan, Tokyo
Japan, Tottori
Japan, Toyama
Japan, Wakayama
Japan, Yamagata
Japan, Yamaguchi
Japan, Yamanashi

Malaysia, Johor
Malaysia, Kedah
Malaysia, Kelantan
Malaysia, Kuala Lumpur
Malaysia, Labuan
Malaysia, Melaka
Malaysia, Negeri Sembilan
Malaysia, Pahang
Malaysia, Perak
Malaysia, Perlis
Malaysia, Pulau Pinang
Malaysia, Putrajaya
Malaysia, Sabah
Malaysia, Sarawak
Malaysia, Selangor
Malaysia, Terengganu

N – S

Netherlands,
Drenthe
Netherlands,
Friesland
Netherlands,
Gelderland
Netherlands,
Groningen
Netherlands,
Limburg
Netherlands,
Noord-Brabant
Netherlands,
Noord-Holland
Netherlands,
Utrecht
Netherlands,
Zeeland
Netherlands,
Zuid-Holland
Netherlands,
Overijssel
Netherlands,
Flevoland

Poland,
Dolnoslaskie
Poland,
Kujawsko-Pomorskie
Poland, Lodzkie
Poland, Lubelskie
Poland, Lubuskie
Poland, Malopolskie
Poland,
Mazowieckie
Poland, Opolskie
Poland,
Podkarpackie
Poland, Podlaskie
Poland, Pomorskie
Poland, Slaskie
Poland,
Swietokrzyskie
Poland,
Warminsko-Mazurskie
Poland,
Wielkopolskie
Poland,
Zachodniopomorskie

Russian Federation,
Altai Krai
Russian Federation,
Altai Republic
Russian Federation,
Amur Oblast
Russian Federation,
Arkhangelsk Oblast
Russian Federation,
Astrakhan Oblast
Russian Federation,
Belgorod Oblast
Russian Federation,
Bryansk Oblast
Russian Federation,
Buryat Republic
Russian Federation,
Chechen Republic
Russian Federation,
Chelyabinsk Oblast
Russian Federation,
Chukotka
Autonomous Okrug
Russian Federation,
Chuvash Republic
Russian Federation,
Irkutsk Oblast
Russian Federation,
Ivanovo Oblast
Russian Federation,
Jewish Autonomous
Oblast
Russian Federation,
Kabardino-Balkar
Republic
Russian Federation,
Kaliningrad Oblast
Russian Federation,
Kaluga Oblast
Russian Federation,
Kamchatka Krai
Russian Federation,
Karelia
Russian Federation,
Karelia
Russian Federation,
Khabarovsk Krai
Russian Federation,
Khanty-Mansi
Autonomous
Okrug—Yugra
Russian Federation,
Kirov Oblast
Russian Federation,
Komi Republic
Russian Federation,
Kostroma Oblast
Russian Federation,
Krasnodar Krai
Russian Federation,
Krasnoyarsk Krai
Russian Federation,
Kurgan Oblast
Russian Federation,
Kursk Oblast
Russian Federation,
Leningrad Oblast
Russian Federation,
Lipetsk Oblast
Russian Federation,
Magadan Oblast
Russian Federation,
Mari El Republic
Russian Federation,
Moscow
Russian Federation,
Moscow Oblast
Russian Federation,
Murmansk Oblast
Russian Federation,
Nenets
Autonomous Okrug
Russian Federation,
Nizhny Novgorod Oblast
Russian Federation,
Novgorod Oblast
Russian Federation,
Novosibirsk Oblast
Russian Federation,
Omsk Oblast
Russian Federation,
Orenburg Oblast
Russian Federation,
Oryol Oblast
Russian Federation,
Penza Oblast
Russian Federation,
Perm Krai
Russian Federation,
Primorsky Krai
Russian Federation,
Pskov Oblast
Russian Federation,
Republic of Adygea
Russian Federation,
Republic of Bashkortostan
Russian Federation,
Republic of Dagestan
Russian Federation,
Republic of Ingushetia
Russian Federation,
Republic of Kalmykia
Russian Federation,
Republic of Karelia
Russian Federation,
Republic of Khakassia
Russian Federation,
Republic of Mordovia
Russian Federation,
Republic of North Ossetia-Alania
Russian Federation,
Republic of Tatarstan
Russian Federation, Rostov Oblast
Russian Federation, Ryazan Oblast
Russian Federation, Saint Petersburg
Russian Federation, Sakha (Yakutia) Republic
Russian Federation, Sakhalin Oblast
Russian Federation, Samara Oblast
Russian Federation, Saratov Oblast
Russian Federation, Smolensk Oblast
Russian Federation, Stavropol Krai
Russian Federation, Sverdlovsk Oblast
Russian Federation, Tambov Oblast
Russian Federation, Tomsk Oblast
Russian Federation, Tula Oblast
Russian Federation, Tver Oblast
Russian Federation, Tyumen Oblast
Russian Federation, Tyva Republic
Russian Federation, Udmurt Republic
Russian Federation, Ulyanovsk Oblast
Russian Federation, Vladimir Oblast
Russian Federation, Volgograd Oblast
Russian Federation, Vologda Oblast
Russian Federation, Voronezh Oblast
Russian Federation, Yamalo-Nenets Autonomous Okrug
Russian Federation, Yaroslavl Oblast
Russian Federation, Zabaykalsky Krai

Spain, Islas Baleares
Spain, La Rioja
Spain, Madrid
Spain, Murcia
Spain, Navarra
Spain, Asturias
Spain, Cantabria
Spain, Andalucia
Spain, Aragon
Spain, Canarias
Spain, Castilla-La Mancha
Spain, Castilla y Leon
Spain, Catalonia
Spain, Extremadura
Spain, Galicia
Spain, Pais Vasco
Spain, Comunidad Valenciana

T – Z
Taiwan, Fu-chien
Taiwan, Kao-hsiung
Taiwan, Tai-pei
Taiwan, Tai-wan

Thailand, Amnat Charoen
Thailand, Ang Thong
Thailand, Buriram
Thailand, Chachoengsao
Thailand, Chai Nat
Thailand, Chalayaphum
Thailand, Chanthaburi
Thailand, Chiang Mai
Thailand, Chiang Rai
Thailand, Chon Buri
Thailand, Chumphon
Thailand, Kalasin
Thailand, Kamphaeng Phet
Thailand, Kanchanaburi
Thailand, Khon Kaen
Thailand, Krabi
Thailand, Lampang
Thailand, Lamphun
Thailand, Loei
Thailand, Lop Buri
Thailand, Mae Hong Son
Thailand, Maha Sarakham
Thailand, Mukdahan
Thailand, Nakhon Nayok
Thailand, Nakhon Pathom
Thailand, Nakhon Phanom
Thailand, Nakhon Ratchasima
Thailand, Nakhon Sawan
Thailand, Nakhon Si Thammarat
Thailand, Nan
Thailand, Narathiwat
Thailand, Nong Bua Lamphu
Thailand, Nong Khai
Thailand, Nonthaburi
Thailand, Pathum Thani
Thailand, Pattani
Thailand, Phangnga
Thailand, Phatthalung
Thailand, Phayao
Thailand, Phetchabun
Thailand, Phetchaburi
Thailand, Phichit
Thailand, Phitsanulok
Thailand, Phrae
Thailand, Phra Nakhon Si Ayutthaya
Thailand, Phuket
Thailand, Prachin Buri
Thailand, Prachuap Khiri Khan
Thailand, Ranong
Thailand, Ratchaburi
Thailand, Rayong
Thailand, Roi Et
Thailand, Sa Kaeo
Thailand, Sakon
Nakhon Thailand, Samut Prakan
Thailand, Samut Sakhon
Thailand, Samut Songkhram
Thailand, Saraburi
Thailand, Satun
Thailand, Sang Buri
Thailand, Sisaket
Thailand, Songkhla
Thailand, Sukhothai
Thailand, Suphan Buri
Thailand, Surat Thani
Thailand, Surin
Thailand, Tak
Thailand, Trang
Thailand, Trat
Thailand, Ubon Ratchathani
Thailand, Udon Thani
Thailand, Uthai Thani
Thailand, Uttaradit
Thailand, Yala
Thailand, Krung Thep
Turkey, Adana
Turkey, Adyamang
Turkey,
Afyonkarahisar
(Afyon)g
Turkey, Arig
Turkey, Aksarayg
Turkey, Amasyag
Turkey, Ankarag
Turkey, Antalyag
Turkey, Ardahang
Turkey, Artving
Turkey, Ayding
Turkey, Balkesirg
Turkey, Bartng
Turkey, Batmang
Turkey, Bayburtg
Turkey, Bilecikg
Turkey, Bingölg
Turkey, Bitlisg
Turkey, Bolug
Turkey, Burdurug
Turkey, Bursag
Turkey, Çanakkaleg
Turkey, Çankrg
Turkey, Çorumg
Turkey, Denizlig
Turkey, Diyarbakrg
Turkey, Düzceg
Turkey, Edirneg
Turkey, Elazg
Turkey, Erzincang
Turkey, Erzurumg
Turkey, Eskihehirg
Turkey, Gaziantepg
Turkey, Giresung
Turkey,
Gümühaneg
Turkey, Hakkarig
Turkey, Hatayg
Turkey, Idirg
Turkey, Ispartag
Turkey, stanbulug
Turkey, zmirg
Turkey,
Kahramanmarag
Turkey, Karabükkg
Turkey, Karamang
Turkey, Karsg
Turkey,
Kastamonug
Turkey, Kayserig
Turkey, Kilisg
Turkey, Kırıkkaleg
Turkey, Kırıklarelig
Turkey, Kırıhisrg
Turkey, Kocaelig
Turkey, Konyag
Turkey, Kütahyag
Turkey, Malatyag
Turkey, Manisag
Turkey, Marding
Turkey, Mersing
Turkey, Mulag
Turkey, Mug
Turkey, Nevşehirg
Turkey, Nideg
Turkey, Orduug
Turkey, Osmaniye
Turkey, Rizeg
Turkey, Sakaryag
Turkey, Samsung
Turkey, anlurfag
Turkey, Sirtg
Turkey, Sinopg
Turkey, makg
Turkey, Silasg
Turkey, Tekirdag
Turkey, Tokatg
Turkey, Trabzong
Turkey, Tunceligen
Turkey, Uakg
Turkey, Vang
Turkey, Yalovag
Turkey, Yozgatg
Turkey, Zonguldakg

United Kingdom, Channel Islands
United Kingdom, England
United Kingdom, Ireland
United Kingdom, Isle of Man
United Kingdom, Northern Ireland

United States, Alabama
United States, Alaska
United States, Arizona
United States, Arkansas
United States, California
United States, Colorado
United States, Connecticut
United States, Delaware
United States, District of Columbia
United States, Florida
United States, Georgia
United States, Hawaii
United States, Idaho
United States, Illinois
United States, Indiana
United States, Iowa
United States, Kansas
United States, Kentucky
United States, Louisiana
United States,
Maine
United States, Maryland
United States, Massachusetts
United States, Michigan
United States, Minnesota
United States, Mississippi
United States, Missouri
United States, Montana
United States, Nebraska
United States, Nevada
United States, New Hampshire
United States, New Jersey
United States, New Mexico
United States, New York
United States, North Carolina
United States, North Dakota
United States, Ohio
United States, Oklahoma
United States, Oregon
United States, Pennsylvania
United States, Rhode Island
United States, South Carolina
United States, South Dakota
United States, Tennessee
United States, Texas
United States, Utah
United States, Vermont
United States, Virginia
United States, Washington
United States, West Virginia
United States, Wisconsin
United States, Wyoming
Troubleshoot Browser RUM

Related pages:
- Create a HAR File to Troubleshoot Web Download Issues

Read the following pages to find ways to solve common problems.

- License Problems
- Browser RUM Metrics Not Reported
- Pages Not Monitored
- Injection Problems
- Browser Snapshot Problems
- Connection Problems
License Problems

On this page:
- No EUM Account License Key
- Browser RUM Does Not Work in Production

Related pages:
- Browser RUM Licenses

The following sections discuss how to solve common license problems.

No EUM Account License Key

The EUM account license is separate from the Controller license. Although the license file includes the EUM license key, the EUM account license is provisioned separately.

Controllers cannot share an EUM license key, but applications can.

If you need an EUM license, call your AppDynamics sales representative or email salesops@appdynamics.com.

Browser RUM Does Not Work in Production

If you are running more than one Controller, each instance requires a unique license. Make sure you are not trying to use the same license on more than one Controller.
Browser RUM Metrics Not Reported

On this page:
- Check for Load
- Ensure Pages Are Instrumented
- Check Connectivity
- Examine Beacons for Issues
- Use Developer Tools to Example Your Application

The sections below provide a high-level overview to checking your Browser RUM setup.

**Check for Load**

Make sure there is load running on your app for the pages that you want to monitor for the selected period.

When Browser RUM discovers a new page, iframe, or Ajax call for the first time, there will be a several minute delay before data for it appears in the Controller UI.

- It takes up to two or three minutes for the agent to register with the Controller. No metrics are captured during this time.
- After the page is registered and traffic starts flowing, it can take an additional two or three minutes for the metrics to appear in the AppDynamics console.
- After that, the delay between an end-user click and the appearance of Browser RUM data is two to three minutes.

**Confirm Browser RUM Is Enabled**

Make sure you enable Browser RUM for the app. Enable it if it is disabled. See [Enable and Disable Browser Monitoring](#).

**Ensure Pages Are Instrumented**

Examine the source of your web page to ensure that you have instrumented the pages that you want to monitor.

**Check Connectivity**

Make sure there is connectivity from the browser to the EUM Cloud. See [Connection Problems](#) for information about troubleshooting connectivity.

**Examine Beacons for Issues**

If beacons are not present or their status is not valid, verify your Browser RUM configuration. See [Set Up and Access Browser RUM](#).

To examine your instrumented application for issues, verify the following:

- The JavaScript Agent extension (*adrum.ext*) is loaded and its status is OK (200).
- The EUM beacon is sent:
  - As a .gif request—older browsers
  - A GET request for *adrum.gif* is loaded, and its status is OK (200).
  - The *ky* parameter in the beacon is set to your application key.
  - As a CORS-based request—deprecated
  - A POST request has been sent with a status of OK (200).
  - The app key is sent in the body of the beacon.
  - As a CORS-based request
  - A POST request has been sent with a status of OK (200).
  - The app key is sent in the URL.

Beacons for using image requests have an inherent length limitation. Because of this, URLs longer than 180 characters, page names longer than 50 characters, and user data (key and value combined) longer than 113 characters are not supported in these browsers.

**Use Developer Tools to Example Your Application**
You can use the developer tools in your web browser to examine your application.

This screenshot shows a .gif-based beacon in Chrome.

This screenshot shows a CORS-based beacon with a key in the URL path.

Create a HAR file to confirm that the browser is downloading the expected resources for your web page.
Pages Not Monitored

On this page:
- Check Exclude Rules
- Check Automatic Injection Configuration
- Configure the JavaScript Agent to Monitor Older Browsers

You may find that some pages have not been monitored as expected. The sections below may help you determine the reason that these pages have not been monitored.

Verify Pages Have Been Injected

If only some web pages are not reporting data, first verify that those pages have been injected with the JavaScript Agent. See Verifying that the JavaScript Agent for Browser RUM was injected.

Check Exclude Rules

If the agent has been injected, the page may have been excluded from monitoring by custom exclude rules. You can check and modify these rules.

To access custom exclude rules for pages:

1. Click Configuration > Instrumentation.
2. Click the Base Pages and iFrames tab.
3. From Base Pages and iFrames, check the custom rules under the Exclude Rules section.

4. To examine and/or modify a custom exclude rule select it in the list and click the icon:

5. If you want to remove a custom exclude rule, select and click Delete.

Check Automatic Injection Configuration

In addition, certain pages could have been excluded by the injection configuration. This can happen when automatic injection is used...
with a limited set of pages enabled for injection. If you used automatic injection, check your automatic injection configuration to see if the missing pages are enabled for injection.

Examine the Request Match Rules and Request Exclude Rules lists under Only enable Automatic Injection for certain Pages in the Automatic JavaScript Injection tab. See Automatic Injection of the JavaScript Agent. Pages can also be bypassed by assisted injection using injection rules when an injection rule specifies only classes and methods to be injected. If you used assisted injection with injection rules, check your injection rules. See Assisted Injection.

**Configure the JavaScript Agent to Monitor Older Browsers**

The JavaScript Agent by default supports modern browser and no longer tests or certifies the collection and reporting of data from older browsers. If you find that older browsers are accessing your web pages and that they are not being monitored, you can configure the JavaScript Agent to monitor older browser with the following configuration.

```javascript
(function(config){
  ...
  "beacon": { "neverSendImageBeacon": false },
  ...
})(window['adrum-config'] || (window['adrum-config'] = {}));
```
Ajax Requests Not Monitored

On this page:
- Check Exclude Rules
- Verify That XHR Objects Were Reused Correctly
- Check Injection Configuration

You may find that some Ajax requests have not been monitored as expected. The sections below may help you determine the reason that these pages have not been monitored.

**Check Exclude Rules**

If the agent has been injected, the Ajax event may have been excluded from monitoring by custom exclude rules. You can check and modify these rules.

To access custom exclude rules for pages:

1. Click [Configuration > Instrumentation](#).
2. From the AJAX tab, check the custom rules under the Exclude Rules section.

3. To examine and/or modify a custom exclude rule select it in the list and click the **[icon](#)**:

4. If you want to remove a custom exclude rule, select and click **Delete**.

**Verify That XHR Objects Were Reused Correctly**

Browser RUM can monitor multiple Ajax calls made from a reused XMLHttpRequest object, but if the XMLHttpRequest object is
reused (opened) before the last request is completed, the JavaScript agent cannot collect all the metrics for all the Ajax requests.

**Incorrect Method to Reuse XMLHttpRequest Object**

In the example below, the `xhr` object is making a request to `/multi-bt` before the request to `/xhr/uptime` has completed, so the JavaScript Agent only reports the second Ajax request to `/multi-bt`.

```javascript
function reuseXHRBeforeReady() {
    var xhr = newXhr();
    xhr.open("GET", "/xhr/uptime");
    xhr.onreadystatechange = safe(function() {
        if (xhr.readyState == 4) {
            if (typeof(console) !== "undefined" && console !== null && console.log)
                console.log("onreadystatechange for xhr1: " +
                xhr.responseText);
            document.getElementById("result").innerHTML =
            xhr.responseText;
        }
    });
    xhr.send(null);
    xhr.open("GET", "/multi-bt");
    xhr.onreadystatechange = safe(function() {
        if (xhr.readyState == 4) {
            if (typeof(console) !== "undefined" && console !== null &&
            console.log)
                console.log("onreadystatechange for xhr2: " +
                xhr.responseText);
            document.getElementById("result").innerHTML =
            xhr.responseText;
        }
    });
    xhr.send(null);
}
```

When an Ajax request cannot be reported because the `XMLHttpRequest` object is reused (opened) before the last request is completed, the JavaScript Agent logs that an Ajax request was not reported in a message similar to the one below. You will most likely not have to modify your JavaScript so that new Ajax requests are only made after previous requests are completed.

**EXT~** The reused XHR object calls open() before finishes last job. No event of XHR to " + this._adrumAjaxT.url() + ": is reported.

**Correct Method to Reuse XMLHttpRequest Object**

The correct method to reuse the `XMLHttpRequest` object to make more than one request is to nest the calls so that new calls are only
made when previous calls have been completed. In the example below, the call to /xhr/multi-bt2 is only made when the call to /multi-bt has completed, so both calls are reported as Ajax requests.

```javascript
function reuseXHR() {
    var xhr = newXhr();
    xhr.open("GET", "/xhr/uptime");
    xhr.onreadystatechange = safe(function() {
        if (xhr.readyState == 4) {
            document.getElementById("result").innerHTML =
            xhr.responseText;
            xhr.open("GET", "/multi-bt");
            xhr.onreadystatechange = safe(function() {
                if (xhr.readyState == 4) {
                    if (typeof(console) !== "undefined" && console !==
null && console.log)
                        console.log("onreadystatechange for xhr2: " +
            xhr.responseText);
            document.getElementById("result").innerHTML =
            xhr.responseText;

            // Reuse xhr again
            xhr.open("GET", "/xhr/multi-bt2");
            xhr.onreadystatechange = safe(function() {
                console.log("/xhr/multi-bt2 responded.");
            });
            xhr.send(null);
        }
    });
    xhr.send(null);
})
xhr.send(null);
}
```

**Check Injection Configuration**

In addition, certain Ajax requests could have been filtered out by the injection configuration. This can happen when the injection uses XHR filters to limit which Ajax requests are monitored. Check your injection configuration to verify that the filtering did not prevent the calls from being monitored.
Injection Problems

On this page:

- Verify Injection
- Verify Automatic Injection is Enabled
- Verify Assisted Injection is Enabled
- Change Your Injection Strategy
- Fix Page Rendering Issues

You must inject the JavaScript Agent for Browser RUM into every page that you want to monitor.

**Verify Injection**

View the source of your web page. When automatic or assisted injection is used, you should see the script for the JavaScript Agent inline in the web page. When manual injection is used, you will see:

```html
<script src="/path_to_adrum.js"/>
```

If you used manual injection, use the normal procedures that you use to verify other types of code changes in your web pages. Keep in mind that various caches, such as the server page, CDN or browser caches, can prevent the page from actually being reloaded. If you cannot get manual injection to work, try one of the other injection schemes if they are available for your platform. See Inject the JavaScript Agent for information about the various injection strategies.

**Verify Automatic Injection is Enabled**

To verify that automatic injection is enabled:

1. Open the business application that is injecting the JavaScript Agent.
2. Click Configuration > User Experience App Integration > JavaScript Agent Injection.
3. From Inject the JavaScript Agent configured for this Browser App, confirm that your browser app receiving the automatic injection is selected.
4. Select the Automatic Injection tab.
5. Confirm that the Enable Automatic Injection of JavaScript checkbox is checked.
6. Verify that automatic injection is enabled for all of the business transactions that you want to monitor. If some of those business transactions are in the Automatic injection possible, but not enabled list, move them to the Automatic injection enabled list. If the business transaction that you want to monitor does not appear in either list, automatic injection is not possible for that business transaction.
7. Click Save if you made any changes.

See Automatic Injection of the JavaScript Agent for information about creating and enabling injection rules.

**Verify Assisted Injection is Enabled**

To verify that assisted attribute injection is enabled:

1. Open the business application that is injecting the JavaScript Agent.
2. Click Configuration > User Experience App Integration > JavaScript Agent Injection.
3. From Inject the JavaScript Agent configured for this Browser App, confirm that your browser app receiving the automatic injection is selected.
4. Select the Configure JavaScript Injection tab.
5. Confirm that the Request Attribute Injection is checked.
6. Click Save if you made any changes.

To verify that assisted injection using injection rules is enabled:

1. Open the business application that is injecting the JavaScript Agent.
2. Click Configuration > User Experience App Integration > JavaScript Agent Injection.
3. From Inject the JavaScript Agent configured for this Browser App, confirm that your browser app receiving the automatic
injection is selected.

4. Select the Configure JavaScript Injection tab.

5. Confirm that there are enabled injection rules in the table under Create Injection Rules.

6. From the Where to Inject JavaScript tab, confirm that the rules are correct.

7. From the Inject for these Business Transactions tab, confirm that the rules are applied for the desired business transactions.

8. Click Save if you made any changes.

See Assisted Injection for information about creating and enabling injection rules.

**Change Your Injection Strategy**

If you try one way to inject the JavaScript Agent and it does not work, it is best to undo the current injection configuration before implementing another one.

- To undo automatic injection, clear the Enable Automatic Injection of JavaScript check box.
- To undo assisted injection using attribute injection, clear the Request Attribute Injection check box, and manually delete the code snippets.
- To undo manual injection, manually delete the JavaScript Agent code from your web pages.
- To undo assisted injection using injection rules, clear the Enable check box for each injection rule in the injection rules list.

If multiple copies of the agent exist on a page, the second copy does not execute.

**Fix Page Rendering Issues**

The rendering of pages in some browsers, in particular, Internet Explorer (IE), may be adversely affected by the JavaScript Agent injection.

After you have confirmed that the JavaScript Agent injection is causing the issue, try the following:

1. Inject the JavaScript Agent after the last <meta> element.
2. Inject the JavaScript Agent just before the closing <head> element.

If you are using automatic injection, see Troubleshoot Automatic Injection.

**Inject After Last Meta Element**

To avoid rendering issues and capture resource timing metrics, you are recommended to inject the JavaScript injection right after the last <meta> element in the <head> as shown below.
Inject Before the Closing Head Element

On rare occasions, some browsers, in particular, Internet Explorer (IE), will still render the page incorrectly because of where the JavaScript Agent is injected. In such cases, as a last resort, try injecting the JavaScript Agent right before the closing `<head>` element:
<head>
  <meta charset="UTF-8">
  <meta name="description" content="Fancy Web Page">
  <meta name="keywords" content="HTML,CSS,XML,JavaScript">
  <meta name="author" content="John Doe">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <script src="https://code.jquery.com/jquery-<version>.js" integrity="sha256-DZAnKJ/6XZ9si04Hgrsxu/8s717jcIzLy3oi35EouyE=" crossorigin="anonymous"></script>
  <script src="js/custom-script.js"></script>
  <link type="text/css" rel="stylesheet" href="css/custom-style.css" />
  ...
  // Injection of the JavaScript Agent before the closing <head> tag
  <script charset='UTF-8'>
    window['adrum-start-time'] = new Date().getTime();
    (function(config){
      config.appKey = 'SH-AAB-AAC-XAR';
      config.adrumExtUrlHttp = 'http://cdn.appdynamics.com';
      config.adrumExtUrlHttps = 'https://cdn.appdynamics.com';
      config.beaconUrlHttp = 'http://eum-col.appdynamics.com';
      config.beaconUrlHttps = 'https://eum-col.appdynamics.com';
      config.xd = {enable : false};
    })(window['adrum-config'] || (window['adrum-config'] = {}));
  </script>
  <script src='//cdn.appdynamics.com/adrum/adrum-latest.js' type='text/javascript' charset='UTF-8'/>
</head>

Troubleshoot Automatic Injection Issues

If you are using the .NET Agent to automatically inject the JavaScript Agent into pages and the pages are not rendering correctly, try the following:

1. Use manual injection and make sure that the JavaScript Agent is injected after the last <meta> element.
2. If you can't use manual injection, add the registered node property `eum-header-beforeendofheadtag` to a Web tier or particular nodes. This node property configures the .NET Agent in the specified tier or nodes to inject the JavaScript Agent right before the closing <head> tag.
## Browser Snapshot Problems

The following sections are intended to help you troubleshoot problems with browser snapshots.

### No Browser Snapshots

If you do not see any browser snapshots, it is possible that browser snapshot collection has been disabled. If periodic collection and error collection and slow collection are all disabled, the agent does not collect any browser snapshots. See [Configure Browser Snapshot Collection](#).

Also, check the thresholds that define slow end-user experience. AppDynamics collects browser snapshots only for slow-performing requests, so if the thresholds are set too high, no requests are flagged as slow. See [Configure Browser RUM Performance Thresholds](#).

### No Correlation between Browser Snapshots and Business Transactions

You get server-side correlation with browser snapshots only if the business transactions associated with the browser snapshot are running on application servers instrumented with AppDynamics app agents. This could explain why you do not see any or do not see all of the business transactions that you expect to see. Check which of your servers are instrumented by app agents and which are not. You may need to get more AppDynamics app agent licenses to get correlation.

If the app servers are all instrumented with AppDynamics app agents, it is possible that the business transactions that you expect to see were not injected with the JavaScript Agent for Browser Monitoring. This can happen when automatic injection is used with a limited set of business transactions enabled for injection. If you used automatic injection, check your automatic injection configuration to see if the missing business transactions are enabled for injection. See [Verifying that the JavaScript Agent for Browser RUM was injected and Automatic Injection of the JavaScript Agent](#) and [Automatically Inj ecting the JavaScript Agent](#). Business transactions can also be excluded with assisted injection using injection rules when an injection rule specifies only certain business transactions to be injected. If you used assisted injection with injection rules, check your injection rules. See [Assisted Injection](#).

### No Transaction Snapshots Associated with Browser Snapshots

Even if all your app servers are instrumented with AppDynamics app agents, it is possible that no associated transaction snapshots were captured at the time of the browser snapshot. For example, if no transactions were slow at the time of the browser snapshot, you probably will not see any transaction snapshots. See [Transaction Snapshots](#) for information about when transaction snapshots are captured. You can modify transaction snapshot capture.

On the browser side, if a browser snapshot is associated with a transaction snapshot, you will see it in the [Transaction Snapshots](#) section of the browser snapshots. See [Business Transactions in Browser Snapshots](#). On the server side, if a transaction snapshot is associated with a browser snapshot, you will see an EUEM GUID in the [ADDITIONAL DATA](#) tab in the transaction snapshot. See [Transaction Snapshots](#).

### Not Getting Full Timing Data for Business Transactions Associated with Browser Snapshots

To ensure full business transaction timing information using older agents and controllers (3.8 and before), you need to inject the JavaScript footer for Browser RUM into the footer of your web pages. Manual injection of the agent does not inject into the footer, so you need to use another injection method to get this functionality.

See [Getting Full Timing Data for Associated Business Transactions](#) and [Choosing Your Injection Method](#).

### No Correlation between Cross Domain AJAX Requests and Business Transactions

Your JavaScript Agent may not be able to correlate Ajax requests with business transactions because browsers restrict access to certain HTTP response headers. Your application, however, can explicitly give browsers access to HTTP headers by sending the HTTP response header `Access-Control-Expose-Headers` with a list of the headers that you want the JavaScript Agent to access.

The process of correlating cross-domain Ajax requests and business transactions consists of the following two steps:

1. Set the agent, tier, or application node property `always-add-eum-metadata-in-http-headers` to `true`. If this property doesn’t exist, you need to add and register it as described in [Add a Registered Node Property](#).
2. Modify the application code, so that the returned response header includes the field `Access-Control-Expose-Headers` as shown below:

```
Access-Control-Expose-Headers:
ADRUM_0,ADRUM_1,ADRUM_2,ADRUM_3,ADRUM_4,ADRUM_5,ADRUM_6,ADRUM_7,ADRUM_8,ADRUM_9,ADRUM_10,ADRUM_11,ADRUM_12,ADRUM_13,ADRUM_14,ADRUM_15,ADRUM_16,ADRUM_17,ADRUM_18
```
Connection Problems

On this page:
- Verify Connectivity
- Unblock Your Firewall

If your browser cannot connect to the AppDynamics EUM cloud and you use an on-premises Controller, but not an on-premises EUM Server, it is possible that:
- You have no Internet connectivity
- A firewall is blocking the port

**Verify Connectivity**

Navigate to SaaS Domains and select your geolocation. Find the EUM API Domain and add the following endpoint: `eumaggregator/ping`

If you get a “ping” in the window, you should be able to connect to the EUM Cloud.

If you are using IE on a Windows system, make sure the browser itself does not have a proxy (with authentication) set up. If it does, the test link may work but not the actual connection.

Make sure you have also unblocked any firewalls and verified the keystore entries as described below.

**Unblock Your Firewall**

The Controller needs to be able to use HTTP over SSL (HTTPS) on port 443 to reach the EUM Cloud Aggregator in your region.

Navigate to SaaS Domains and select your geolocation. Find the EUM API Domain and add the following endpoint: `eumaggregator/ping`

If your Controller is behind a firewall, you can either open your Controller’s firewall or use a forward proxy.

To open the firewall, see the instructions unique to your firewall.

You only need to open the firewall for the specific host and port, not for the entire *.eum-appdynamics.com domain.

**Use a forward proxy:**

1. Set up an HTTP proxy to your SaaS EUM API Server. Navigate to SaaS Domains and select your geolocation. Find the EUM API Domain and add the following endpoint: `eumaggregator/ping`

   This is a cleartext/pass-through proxy. Authentication is not supported on the first level. If the client network itself requires authentication, you must set up an intermediate proxy between your Controller and this proxy to pass on the credentials you need to get out of your network.

2. Configure the HTTP proxy host and port in the `<Controller-Installation-Directory>/appserver/glassfish/domains/domain1/config/domain.xml file`

   See Configure Controller HTTP Proxy Settings for details.

3. Restart the Controller’s app server.
JavaScript Errors

The sections below provide helpful tips for finding and solving common JavaScript errors.

Enable Error Reporting

It is possible that reporting is disabled or that certain JavaScript or Ajax errors that you would like to be reported have been configured to be ignored. See Configure JavaScript and Ajax Error Detection.

If another script on your monitored pages sets the JavaScript window.onerror event, this setting can interfere with Browser RUM error capture. See Handle the window.onerror Event to learn how to catch those errors.

Visualize JavaScript Errors

You can use Analytics to query your Browser Request event data and create a JavaScript exception and error summary dashboard. To learn how to create the dashboard, see Visualize JavaScript Errors.

View Error Information for Cross-Domain JavaScript

By default and for security reasons, most modern browsers do not provide access to the error information in window.onerror for scripts that are loaded from other domains. To access this error information, you need to enable cross-origin resource sharing (CORS). Firefox and Chrome both allow you to enable CORS by making small changes to your server and script element in your HTML.

If you have not enabled CORS, you will only see the error message "CROSSORIGIN" shown below:

<table>
<thead>
<tr>
<th>Script Origin</th>
<th>Line #</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>CROSSORIGIN</td>
<td>0</td>
<td>Script error</td>
</tr>
</tbody>
</table>

Once CORS is enabled, you will able to see error details like those shown here.

Also, when you can navigate to your page from Chrome and then open JavaScript Console > Network > Headers, you'll see the HTTP response header access-control-allow-origin: *.

In the following sections, we're going to show you how to make changes to access the error information in window.onerror cross-domain scripts.

Server Change

To enable CORS, you need to add this header to responses that are not from the same domain as the caller:

Access-Control-Allow-Origin: *

Script Tag Change
The `script` element has a new non-standard attribute called `crossorigin`. The most secure value for `crossorigin` would be `anonymous`. So, you'll have to modify your script tags to look like the following.

```html
<script src="http://sub.domain.com/adrum.js"
crossorigin="anonymous"></script>
```

**Browser Support for CORS**

In the future, we expect most browsers to enable CORS. Internet Explorer 10+ already has native support for CORS and reports errors to `window.onerror` for both local and cross-domain scripts. See CORS: Browser support to see a list of browsers/layout engines that support CORS.
Browser Synthetic Monitoring

On this page:

- Access Synthetic
- Limitations

Related pages:

- Browser Synthetic Monitoring Versus Browser Real User Monitoring
- Get Started with Browser Synthetic Monitoring

Search the Browser Synthetic topics:

Browser Synthetic Monitoring uses geographically distributed Synthetic Agents to continuously test key user workflows in your application. This allows you to monitor the correctness and performance of multi-step flows independently of the user-generated load.

There are two ways of using Synthetic:

- **Scheduled jobs**: Use this to test your pages on a recurring basis to ensure they continue to perform well. Upload a Python WebDriver script to exercise multi-step workflows. Using this feature requires a Synthetic Pro license. See Browser Synthetic Licenses for more information.
- **On-demand snapshots**: Use this to collect performance data right now. This is useful for ad-hoc analysis. This feature is available with both Synthetic Lite and Pro licenses.

There are two deployment types for Synthetic Agents:

- **Synthetic Hosted Agents**: AppDynamics hosts these Synthetic Agents. This deployment is recommended for testing public websites.
- **Synthetic Private Agents**: You install and host Synthetic Agents on your machines. This enables you to test internal sites and services. For more information, see Synthetic Private Agent Deployment.

Separate licenses are required for the Synthetic Hosted Agent and the Synthetic Private Agent. See Synthetic Agent Licenses for details.

Access Synthetic

1. Open the browser application that you want to test with Synthetic or create a new one if you’re just getting started.
2. Select one of the menu options under **SYNTHETIC** in the left navigation bar.
Scheduled Jobs

The **Jobs**, **Sessions**, and **Pages** options are only displayed if you have a Browser Synthetic Pro license.

- **Jobs**: Create and manage recurring jobs. See [Get Started with Browser Synthetic Monitoring](#).
- **Sessions**: View individual executions of your jobs (i.e., “sessions”) and perform an analysis to understand where the problems are.
- **Pages**: View aggregated page-level statistics collected from Synthetic. This is particularly useful for comparing Browser RUM with Synthetic.

You can also see some synthetic results in the context of real user traffic in the [Browser App Dashboard](#).

On-Demand Snapshots

Create and view on-demand snapshots using **On-Demand**.

Limitations

**Browser Extensions**

Browser Synthetic Monitoring does not support monitoring for the browser extensions *Adobe Flash*, *Microsoft Silverlight*, or *Java applets*.

**Single-Page Applications (SPAs)**

Browser Synthetic Monitoring does not fully support SPA. Instead of reporting the base page and virtual pages separately like Browser
RUM does, Browser Synthetic Monitor reports all the virtual pages with the base page, so you will only see one page in the Synthetic Sessions.

**Multiple Window Tests**

Browser Synthetic Monitoring supports running tests in multiple windows for Firefox and Chrome, but *not* for Internet Explorer Browser.
Browser Synthetic Monitoring Versus Browser Real User Monitoring

Browser SUM (Browser SUM) and Browser Real-User Monitoring (Browser RUM) both have the same goal of helping you improve your user's experience. The difference between the two is how they are used to accomplish this goal.

The following sections will discuss the differences and how to use the two types of monitoring together.

Identify Issues

Although Browser RUM can detect certain kinds of problems (like JavaScript exceptions), it cannot comprehensively test for functional correctness. For example, you may want to verify that your online store has reasonable prices a list of items. If your site is down entirely, then the JavaScript Agent will never be loaded, so errors or verifications will not be reported. Fortunately, Browser SUM will keep running, discover the error, alert you, and provide detailed information about the problem.

Control Environmental Factors

Performance analysis in Browser RUM is complicated by the wide range of hardware, browsers, operating systems, and networks with which users access your site. Browser SUM uses consistent hardware, software, and network configurations, so if you see deviations in performance, you can be fairly certain a problem exists.

Use Synthetic Metrics and Screenshots to Understand the User Experience

Browser SUM can collect certain data that Browser RUM can't. For example, Browser SUM provides screenshots, which help you see what's going on. You can also use the Visually Complete and related metrics to understand how users experience page load time.

Performance Versus Workflows

Browser RUM excels at capturing the full breadth of performance that your real users experience. Browser SUM gives you confidence that your key workflows are always working and performing.

Differences Between Synthetic and Browser RUM Metrics

Although Browser SUM and Browser RUM report similar metrics, you should be wary of comparing them because of these differences:

- hardware
- network connections
- different browsers
- browser caching is not present in synthetic sessions

If you see a sudden change in any of those metrics, however, you should compare the results of Browser SUM and Browser RUM to see if there is an existing problem.
Get Started with Browser Synthetic Monitoring

On this page:
- Create a Job
- Monitor Job Trends
- View the Results
- Understand the Results

Related pages:
- Configure Synthetic Jobs
- Synthetic Scripts
- Synthetic Sessions

This page covers the basics of using Browser Synthetic Monitoring. You’ll learn how to get to the Jobs page, create a job that measures a URL, view the results from the Sessions page, and finally how to edit and delete your job.

Create a Job

1. Open one of your browser applications.
2. Click Jobs to open the Jobs page.
3. If you don’t have any previous jobs, click Create a New Job. If you have previous jobs, click Add.
4. From the New Job dialog, do the following:
   a. For Specify your Test, enter a URL in the URL field and “Example Job” in the Name field.
   b. For Choose Browsers, select Chrome.
   c. For Choose Locations, click + and select any location.
   d. Click Save.

Monitor Job Trends

From the Synthetic Job Dashboard, you can quickly monitor the availability and performance trends of synthetic jobs.

Availability Trends

The Availability Trend column that displays the status of the executed jobs as shown in the screenshot below. The Availability Trend shows the session status based on the coloring codes listed in Availability Trend Status Codes. The session status is not based on health rules. See Session Status for a detailed description of each status.

The session status is not based on health rules. See Session Status for a detailed description of each status.
Availability Trend Status Codes

The **Availability Trend** shows the session status based on the coloring codes listed in the following table.

<table>
<thead>
<tr>
<th>Availability Trend Status Color</th>
<th>Session Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Green</td>
<td>OK</td>
</tr>
<tr>
<td>Yellow</td>
<td>WARNING</td>
</tr>
<tr>
<td>Red</td>
<td>FAILED</td>
</tr>
<tr>
<td>Dark gray</td>
<td>BROKEN</td>
</tr>
<tr>
<td>Light gray</td>
<td>INTERNAL_ERROR</td>
</tr>
</tbody>
</table>

Understanding the Availability Metric

The Availability metric is the percentage of successful sessions for a given time range. That is the percentage of sessions with status as OK or WARNING within the given time range.

The **View Options** enables you to view the availability over periods not restricted to the specified time frame. The following list describes the availability options:

- **Availability Last-24-Hours** - The availability for the last 24 hours from the moment the page is refreshed/opened.
- **Availability Month-to-Date** - The availability from the starting moment (12 AM) of the current month according to user's timezone until the current time (page refreshed/opened).
- **Availability Week-to-Date** - The availability from the starting moment (12 AM) of the current week Monday according to user's timezone, considering week starts on Monday, till now (page refreshed/opened).

Performance Trends

The **Performance Trend** column displays the performance threshold status of the executed jobs as shown in the screenshot below. The **Performance Trend** column only displays successful sessions: those with the status of OK or WARNING. The performance trend statuses are based on the configured performance thresholds for the synthetic job.

For example, if the **Availability Trend** column shows 5 OK and 5 FAILED status counts (50% green, 50% red), then the corresponding **Performance Trend** column will show only 5 OK counts (100% green). If the **Availability Trend** column has 5 failed and 5 broken (50% red, 50% gray) jobs, that is all unsuccessful, then the corresponding **Performance Trend** column will show no count that is an empty bucket (light gray).

Performance Trend Status Codes

As with the availability status, colors are used in the trend column to indicate performance status:

<table>
<thead>
<tr>
<th>Performance Trend Status Color</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Green</td>
<td>No threshold violated: Green.</td>
</tr>
</tbody>
</table>
### Yellow
Warning threshold violated.

### Red
Critical threshold violated.

A job can have a warning violation for one metric and a critical violation for another metric. In this case, the critical violation will override the warning violation.

### View the Results

1. From the **Jobs** page, verify that your job has a check next to it indicating that the job has been enabled.
2. After a few minutes, you should see a time that your job was last run and some metrics.
3. Select your job and click **Details**.
4. You should now see the results for your job on the **Sessions** page.

### Understand the Results

You can understand the results in several different ways and levels from the **Sessions** page. You can sort session results by criteria and then examine the details of a specific session in the **Session Details**. From the **Charts** tab of the **Sessions** page, you can then view widgets representing aggregated session data in charts and a configurable set of fields for each session.

The following sections discuss how to use **Sessions** and **Session Details** to understand the results of your jobs.

#### Examine Session Details

**Examine Session Details**

You can double-click any session to get a summary of the session, page details, and a breakdown of details for each page in the session. You’ll be able to view how the page loaded, the resources that were requested, and the business transaction for each page. You can use this information to find where network latency occurs or where scripts break.

For example, from the **Session Details** dialog below, you can determine that the visually complete (when the user would see the completely loaded page) took around 16 seconds and then examine the duration of the page load from the HTTP request to the first byte, from the first byte to DOM ready, and finally from DOM ready to visually complete.
Get a Quick Overview

From the Sessions page, you can click the Charts tab to view widgets representing aggregated session data. For example, the Lowest Availability widget shows you what locations are the least available, so you can confine and focus on issues in specific locations.

### Lowest Availability

- England: 49.05%
- San Jose: 50.00%
- Delhi: 53.60%
- Amsterdam: 55.94%
- Hong Kong: 52.94%

### Availability Trend

52.6% Availability Trend

### Average Session Visual Time

9,486 ms Average Session Visual Time
Synthetic Jobs

On this page:
- Synthetic Job Locations
- Synthetic Job Queue
- Maturity Level for Scheduled Jobs
- Synthetic Job Execution
- Synthetic Job Throttling

Related pages:

Synthetic jobs are configurable, scheduled tests consisting of URL measurements or synthetic scripts. To create a job, see Get Started with Browser Synthetic Monitoring.

Synthetic Job Locations

For URL measurements and synthetic script jobs, the Synthetic Agent is always run in the configured browser location. The synthetic script, however, is not run on the same machine or location as the configured browser for security reasons. See Synthetic Agent Locations for more details.

Synthetic Job Queue

When you create a job, it is added to a queue based on the creation timestamp. Scheduled jobs are assigned a maturity level that influences when they will be executed in the queue. On-demand jobs are placed in the same queue but are not assigned a maturity level.

Maturity Level for Scheduled Jobs

Jobs are assigned a maturity level based on when they are created and added to the queue. The following table describes the two supported maturity levels.

<table>
<thead>
<tr>
<th>Maturity Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Junior</td>
<td>This maturity level consists of the synthetic jobs that were just created or updated within the last two hours. Once two hours have passed since the last update or creation of the job, the job will be promoted to Senior. The promotion from Junior to Senior is only performed when the required capacity has been allocated to avoid negatively affecting the currently executing jobs.</td>
</tr>
<tr>
<td>Senior</td>
<td>This maturity level consists of scheduled jobs that have been created or updated more than two hours ago. The synthetic jobs with the maturity level of Senior have been allocated resources to be executed.</td>
</tr>
</tbody>
</table>

Synthetic Job Execution

The execution order of synthetic jobs depends on the job priority and the job creation time. The Synthetic Server job marks the priority of a job based on several factors: the job type, job maturity, and your Browser Synthetic Monitoring license.

Job Execution Order

The following table shows the job priority based on the job type, license type, and job seniority. The actual job execution order depends on the job priority and job creation timestamp.

<table>
<thead>
<tr>
<th>Job Priority</th>
<th>Job Type</th>
<th>License Type</th>
<th>Job Seniority</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Scheduled</td>
<td>Pro</td>
<td>Senior</td>
</tr>
<tr>
<td></td>
<td>Scheduled</td>
<td>Trial</td>
<td>Senior</td>
</tr>
<tr>
<td>On-Demand</td>
<td>Pro</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Synthetic Job Throttling

Synthetic jobs are only executed if they are in the job queue. If the max job queue size has been reached, however, no additional synthetic jobs will be added to the queue until the queue size decreases. The max queue size is defined by the priority, license, type, and maturity level.

The following table shows how the maximum queue size is calculated:

<table>
<thead>
<tr>
<th>Max Queue Size</th>
<th>Job Priority</th>
<th>Job Type</th>
<th>Job Maturity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Synthetic Agents * 10 (Pro)</td>
<td>High</td>
<td>Scheduled</td>
<td>Senior</td>
</tr>
<tr>
<td>Number of Synthetic Agents * 10 * 0.5 (Trial)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Synthetic Agents * 10 * 0.25</td>
<td>On-Demand</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Number of Synthetic Agents * 10 * 0.5</td>
<td>Low</td>
<td>Scheduled</td>
<td>Junior</td>
</tr>
</tbody>
</table>

### Job Execution Errors

Customer is seeing different error messages with synthetic jobs and will like to understand their meaning and get more information to help with checking the issue at their end.

<table>
<thead>
<tr>
<th>Error Message</th>
<th>Cause of Error</th>
<th>Error Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skipped; still waiting for a previous job execution</td>
<td>An attempt was made to queue a high-priority measurement request from a scheduled job before the previous measurement request from the same job and location-browser combination has been processed.</td>
<td>TARDY</td>
</tr>
<tr>
<td>Skipped while new capacity is being added</td>
<td>An attempt was made to queue a low-priority measurement request from a scheduled job before the previous measurement request from the same job and location-browser combination has been processed.</td>
<td>TARDY_ONBOARDING</td>
</tr>
<tr>
<td>Skipped while new capacity is being added</td>
<td>An attempt was made to queue a measurement request from a junior job beyond the maximum respective queue capacity.</td>
<td>ONBOARDING</td>
</tr>
<tr>
<td>Testing location is overloaded</td>
<td>An attempt was made to queue a measurement request from a senior job beyond the maximum respective queue capacity.</td>
<td>THROTTLED</td>
</tr>
</tbody>
</table>
Configure Synthetic Jobs

On this page:
- Choose Locations
- Choose Browsers
- Customize Connection Speeds
- Schedule Jobs
- Set Timeouts
- Configure Availability Rules
- Configure Performance Thresholds

Related Pages:
- Get Started with Browser Synthetic Monitoring
- Alerts for Browser Synthetic Monitoring

There are many ways to configure a scheduled job. You can schedule when, where, and how often the job runs, set timeouts, configure performance thresholds to trigger warning and critical events, and customize connection speeds. Before configuring a job, first read *Get Started with Browser Synthetic Jobs*.

The following sections demonstrate how to use the Controller UI to configure a job.

**Choose Locations**

You can choose one or more locations in which the job will run. This can also be configured to test all of the locations every time or one location each time the job is run.

---

Hosted Agent Locations and Synthetic Private Locations

If you have deployed a *Synthetic Private Agent Deployment*, you can choose from *Hosted Agent Locations* or *Private Agent Locations*. The *Hosted Agent Locations* are where AppDynamics hosts public synthetic agents. The *Private Agent Locations* are where you are hosting your synthetic private agents.
Choose Browsers

A job can run on one or more different types of browsers. For mobile browsers, you can specify a platform, such as the iPhone 7 Plus or Pixel. When a job runs on mobile browsers, it is actually running a Chrome browser emulator with the specified platform properties.

Customize Connection Speeds

The connection speed helps keep performance consistent and realistic. If you choose Native Connection, the job will run at the maximum speed available to AppDynamics data centers. The default is Cable (5/1 Mbps 28ms RTT).

Schedule Jobs
When you create or edit your job, you can define the job schedule. The job will run within the specified timezone. When you choose a timezone, the job schedule will not be affected by time changes in any other timezone, such as daylight saving time. The default timezone is GMT.

### Choose a Schedule

- **Run every**
  - **15 Minutes**

**More Options**

- **Timezone**
  - **[GMT+00:00] Greenwich Mean Time**

- **Between**
  - **12:00am** and **11:00pm**

- **From**
  - **05/01/19 12:14 PM** to **05/01/19 12:14 PM**

- **Only on These Days**
  - **S M T W T F S**

### Set Timeouts

Synthetic jobs consume licenses based on the duration of the job, so you can set a timeout to help limit the license consumption. If the job times out, data will still be collected until the timeout expires. The default suggestion for job timeout is 15 seconds for a URL and 30 seconds for a script. You can adjust the timeout as preferred, but keep in mind that short timeouts might result in unsuccessful sessions, and long timeouts might result in overconsumption of licenses.

### Configure Availability Rules

You can configure the synthetic job to check the availability of pages and resources. For example, you can set the session status for when a page fails to load or when a resource is missing or inaccessible.

The screenshot below is a configuration to treat the session as **Failed** if any page fails to load and to treat the session as **Warning** if any resource fails to load. If the job navigates to a page with an HTTP 4xx or 5xx status code, the session status will be set to **FAILED**. This will trigger a Critical Event. If a resource fails to load, the session status will be set to **WARNING**, which will trigger a Warning Event. To set up alerts, see [Alerts for Browser Synthetic Monitoring](#).

### Configure Availability Rules

- **If any page fails to load**
  - treat session as **Failed**
- **If any resource fails to load**
  - treat session as **Warning**

Use the **Advanced Options** to ignore or only consider selected resources. For example, the screenshot below might be used to ignore the font resource (which browsers may not have) and only consider high-impact resources like CSS, JavaScript, etc.
Configure Availability Rules

If any page fails to load treat session as Failed
If any resource fails to load treat session as Warning

Advanced Options

Ignore resource when URL contains
comma-separated list
Only consider resources when URL contains
comma-separated list

Favicons are always ignored when checking the availability of resources.

Configure Performance Thresholds

You can specify the job performance thresholds to send alerting events. If you check one or both of the "Automatically retest after warning events" or "Automatically retest after critical events" boxes, and the event is generated, then the job will be triggered to re-execute immediately per unconfirmed warning or critical event. For example, if the Critical Events threshold is exceeded, the event Critical Started will be triggered and the job will rerun until the event Critical Continues is confirmed.

Configure Performance Thresholds (optional)

Trigger a warning event when

Add Threshold

End User Response Time is greater than 5000 ms after retest

Automatically retest after warning events

Trigger a critical event when

Add Threshold

End User Response Time is greater than 10000 ms after retest

Automatically retest after critical events

From the Events page shown below, you can see that the results of the first and successive tests exceeded the threshold for synthetic availability and triggered the Error Started, Error confirmed after restart, Error Continues, and Problem Ended events.
Synthetic Scripts

On this page:

- Benefits of Using Synthetic Scripts
- How Scripts Are Run
- Supported Python Libraries
- Get Help

Browser Synthetic Monitoring allows you to write Python scripts using the Selenium WebDriver library to create scheduled tests that can mimic end-user actions in remote browsers in supported locations.

You can use synthetic scripts to test performance, features, end-user flows, and the end-user experience.

For example, your synthetic script might do the following:

1. Visit your website.
2. Search for products.
3. Add items to a shopping cart.
4. Check out the shopping cart and assert that the total price of the cart is correct.
5. Enter payment details.
6. Complete the purchase.

Benefits of Using Synthetic Scripts

Using synthetic scripts, you can:

- find broken user flows
- improve the performance of your site by finding where response times in your web applications are slow
- test new features

How Scripts Are Run

The entire process can be simplified into the following steps:

1. You add your script to a synthetic job in the Controller UI.
2. The synthetic job is transmitted to the Synthetic servers, where it is scheduled to be run.
3. When it's time for your job to run, a container is created just for your script.
4. Your script runs in the container, controlling a web browser using Remote WebDriver.
5. After the script finishes running, the agent collects browser metrics and sends them to the Controller UI.

The Controller UI receives and displays the metrics from your synthetic job.

Supported Python Libraries

In addition to the libraries included in the standard distribution of Python 2.7, you can also use the following:

- selenium
- Requests
- pystftp

Get Help

In the Knowledge Base forum, read our Synthetic Scripts FAQ. If you can't find the answer to your question, submit your questions to the community.
Write Your First Script

On this page:
- Write Your Script
- Create a Synthetic Job with Your Script
- View Results

Related pages:
- Locate DOM Elements
- Wait for DOM Elements
- Work with Screenshots
- Add Logs to Troubleshoot
- Verify Program Correctness

The sections below quickly show you how to create a synthetic job and view the results.

Write Your Script

Before you create a Synthetic Job with a custom script, you will need to write a Python WebDriver script and get it working locally. For a good introduction to scripting, we recommend reading Getting Started in the Selenium Python Bindings documentation.

Also, before you actually run your scripts in Synthetic Jobs, you should be aware of the differences between Python and synthetic scripts as you'll probably have to make some modifications.

Test Scripts Locally

One of the best ways to take advantage of utilities to help you write scripts and then run them locally. To run scripts locally, complete the installation instructions given in the Selenium Python Bindings documentation and then run the script as you would any Python script.

Tools for Writing Scripts

If you are comfortable with WebDriver, but just want help with the tedious work of finding DOM selectors, you can use AppDynamics' WebDriver Scripting Assistant, which is a Chrome extension.

If you want a full-fledged script recorder, you can use the Firefox add-on Katalon Recorder (Selenium IDE for Firefox 55+). The resulting script will work with Browser Synthetic, but as with most auto-generated code, expect to make manual changes to improve it.

Understand the Differences Between Python and Synthetic Scripts

When you are ready to write synthetic scripts, you should be aware that although Browser Synthetic can run unmodified Python scripts, there are few differences:

- The driver initialization code can be removed because the variable `driver` is automatically created and initialized with the browser selected in the Job creation UI. Thus, the example code below that initializes a driver would be replaced by our logic that then returns the variable `driver`.

```python
from selenium import webdriver

driver = webdriver.Firefox()
```

If your script initializes a driver, it is okay to keep it: Synthetic will just ignore your
Only one driver can be run within a synthetic session. Any attempt at creating new drivers will return the automatically instantiated driver (a singleton).

Scripts interacting with multiple windows will work; however, the session results will merge the results for all windows into one waterfall.

Create a Synthetic Job with Your Script

Once you have completed and tested your scripts, the next step is to create a Synthetic Job with your script:

1. From the New Job dialog, check the Run a script radio button.
2. Enter My First Script in the Name field.
3. Take a look at the sample WebDriver script, but don't change it.
4. For Choose Browsers, select Chrome and Firefox.
5. For Choose Locations, add a couple of different locations.
6. Click Save.

View Results

You have two types of views for the results. From the Jobs page, where you create a job, you can see your job and a summary of the results after the job has been run for both Synthetic Hosted/Private Agents.

You can also see more detailed results from the Sessions page. You can get there by either double-clicking your job or by clicking Sessions in the left navigation bar.
To see the metrics as charts, click the **Charts** tab:

From the **Records** tab, you can double-click a session to view the **Session Details** dialog.
Locate DOM Elements

On this page:
- Methods for Locating Elements
- Best Practices for Locating Elements

Related pages:
- Write Your First Script
- Wait for DOM Elements
- Work with Screenshots
- Add Logs to Troubleshoot
- Verify Program Correctness

Your synthetic script simulates end-user interactions, so you'll need to be able to locate the DOM elements of your web pages. For example, your script might click on buttons, links, or enter text into text fields: this requires your script being able to locate and then select relevant HTML elements.

Methods for Locating Elements

The Selenium WebDriver library provides CSS selectors and XPath statements for selecting HTML elements. See 4. Locating Elements for the list of the library methods and usage examples.

You can also use the AppDynamics WebDriver Scripting Assistant, a Chrome extension, to help you create the selector statements.

Best Practices for Locating Elements

When locating elements, you are recommended to do the following. Think of it as a checklist.

- Understand the following about your application:
  - the DOM structure
  - which pages are dynamically and statically loaded
  - which elements of a page are loaded and visible.
- Use unique IDs for elements and selectors that are as short as possible: Selectors break all the time. Long hierarchical selectors break more easily than shorter ones, and using short selectors will reduce script maintenance over time.
- Click on the user-visible element instead, or send a Return key instead of submitting forms using the method submit().
- Pay attention to element visibility. The specification is complex, and the results are not always what users expect. See the WebDriver specification for more information.
Your script might be accessing a UI component that is not available in the DOM. The cause could be network latency, as shown in the diagram below, or the DOM simply isn’t ready to access.

You can avoid this issue by using either an explicit or implicit wait. The following section discusses the different types of waits and when to use each.

**Explicit Versus Implicit**

The WebDriver function `driver.implicitly_wait(n)` configures WebDriver for the remainder of a session. If you try to access an element that is not available, WebDriver will retry for up to `n` seconds to find that element in the DOM. This simple approach is often sufficient to make your scripts work reliably.

In the diagram below, the WebDriver retries to click **Button 2** until it's available.
An explicit wait is code that requires a certain condition to occur before continuing to execute code. Although you may be tempted to use `time.sleep()` to wait for an element to be available, you should avoid doing this. Instead, use the WebDriver API methods given in 5.1. Explicit Waits of the Selenium Python Bindings documentation that enable you to wait for expected conditions.

In the diagram below, the explicit wait stipulates to click Button 2 (action) only when the button is available (the condition).
Work with Screenshots

On this page:

- Where to Find Screenshots
- When to Use Screenshots to Debug
- How to Take Manual Screenshots

Related pages:

- Write Your First Script
- Locate DOM Elements
- Wait for DOM Elements
- Work with Screenshots
- Add Logs to Troubleshoot
- Verify Program Correctness

Synthetic jobs will automatically take screenshots for you, but you might want to manually take screenshots to debug issues.

Where to Find Screenshots

The Session Details dialog shown below displays screenshots for each page, the last screenshot taken in the session, and a screenshot icon in the waterfall showing when the screenshot was taken. You can also view the larger version of the page screenshot from the Screenshots tab.

When to Use Screenshots to Debug

If interactions with a page are resulting in failures, you can use a screenshot to detect whether the UI component is on the page. For example, a screenshot can be used to determine if you are trying to click a button that isn’t on the page or that is being overlapped by another HTML component. You may also want to manually take a screenshot if your job is timing out to diagnose an issue with the page because screenshots are not be automatically taken for timed out jobs.

How to Take Manual Screenshots

Although screenshots are taken by default, you can manually take screenshots at any point during the execution of your script using either of the following:
driver.get_screenshot_as_file("homepage.png")

driver.save_screenshot("myscreenshot.png")
Add Logs to Troubleshoot

You can use logs to track the progress of your scripts and to better understand what is happening while the script is executing.

Print Statements

The simplest way to log messages is with print statements. You can use them to inspect objects or to determine where an issue is located.

For example, you can determine whether an element was selected on a page by displaying the contents of the object:

```python
elem = driver.find_element_by_id("wiki-content")
print(elem)
```

If the script results are incomplete, you can use a series of print statements to pinpoint where the problem might have occurred:

```python
print("Getting the web page 'appdynamics.com'.")
driver.get("http://docs.appdynamics.com")

print("Getting the content for the container with the ID 'wiki-content'.")
elem = driver.find_element_by_id("wiki-content")

element = driver.find_element_by_xpath("//select[@name='name']")
all_options = element.find_elements_by_tag_name("option")
for option in all_options:
    print("Value is: %s" % option.get_attribute("value"))
    option.click()
```

Logging Package

You can import the logging package to log different types of error messages.

To import the logging package:
import logging

Once you have imported the `logging` package, you can generate different types of logs:

```python
print("This is a print statement.")
logging.info("This is an info message")
logging.warn("This is a warning message")
logging.debug("This is a debug message")
logging.error("This is an error message")
```

The different types of errors are then color-coded in the waterfall of the **Session Details** dialog.

The script output will label the different log messages by the severity as shown below.
<table>
<thead>
<tr>
<th>Time (sec)</th>
<th>Severity</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>INFO</td>
<td>Starting measurement [b44b4b60-c-1234-5678-9012-345678901234]</td>
</tr>
<tr>
<td>0.000</td>
<td>INFO</td>
<td>Executing WebDriver script</td>
</tr>
<tr>
<td>14.231</td>
<td>INFO</td>
<td>This is a print statement.</td>
</tr>
<tr>
<td>14.231</td>
<td>INFO</td>
<td>This is an info message</td>
</tr>
<tr>
<td>14.231</td>
<td>WARNING</td>
<td>This is a warning message</td>
</tr>
<tr>
<td>14.231</td>
<td>ERROR</td>
<td>This is an error message</td>
</tr>
<tr>
<td>14.231</td>
<td>INFO</td>
<td>WebDriver script completed successfully!</td>
</tr>
</tbody>
</table>
Import Python Packages in Synthetic Scripts

Synthetic scripts are executed in short-lived containers where you can use preinstalled Python packages. Your script is restricted to the container and the container's lifespan.

**Preinstalled Libraries**

The following modules are installed in containers and can be accessed by importing them in your synthetic scripts.

- `os`
- `selenium`
- `Requests`
- `pysftp`

**Environmental Variables**

Because the `os` module is available, you can access the environment variables through `os.environ`.

For example, to display the variables, you could use a script similar to the following:

```python
import os
for i, j in os.environ.items():
    print(i, j)
```

**Upload and Download Files with SFTP**

One of the packages installed on these containers is `pysftp`, which you can use to upload and download files with SFTP.

The following code sample shows the basic functionality of the package.
import pysftp

cnopts = pysftp.CnOpts()
cnopts.hostkeys = None
# The <hostname> can be in either of the following formats: ftp.domain.ca || http://ftp.domain.ca
with pysftp.Connection('<hostname>', username='xxxx', password='xxxxxx', cnopts=cnopts) as sftp:
    # Change to the directory 'public'
    with sftp.cd('public'):
        # Fetch the remote file
        sftp.get('<remote_file_name>')
        # Upload the same file to a remote location
        sftp.put('<local_file_name>')

---

**Make HTTP Requests**

You can use the HTTP library requests to make HTTP requests within your synthetic scripts. The following example makes a GET request to the public GitHub Events API. See *Requests: HTTP for Humans*™ to learn more about the library’s functionality and usage.

```python
import requests

r = requests.get('https://api.github.com/events?per_page=1')
print("Status Code: %s\n" %(r.status_code))
print("Headers: \n%s\n" %(r.headers))
print("Response: \n%s") %(r.text)
```
Verify Program Correctness

On this page:
- Use Assertions to Verify Expected Results
- Use the Session Status to Find Script Issues

Related pages:
- Write Your First Script
- Locate DOM Elements
- Wait for DOM Elements
- Work with Screenshots
- Add Logs to Troubleshoot

You can use assertion statements to determine that your script is receiving the expected results. You use the session status to determine session status whether your jobs have failed because of an uncaught exception or an assertion statement failed.

In the following sections, you’ll find an assertion example and learn how to use the session status to determine which scripts have issues.

Use Assertions to Verify Expected Results

Assert statements use a simple syntax to verify an expected result. You can also provide a message to display if the assertion fails. In the example below, the message "Title should contain AppDynamics" will be displayed if the assertion fails.

```python
# Get page and check the title
driver.get("http://https://ecommerce.com/view.html/ref=nav_cart")
assert "E-Commerce Shopping Cart" in driver.title, "Title should contain E-Commerce Shopping Cart"

# Click link and confirm URL is correct
driver.find_element_by_link_text("Place your order").click()
assert "ThankYouCart" in driver.current_url
```

You can also verify that a string is not in the result or the truth of a mathematical expression. Assert statements that are false will result in a session status of FAILED.

```python
# Confirm the page doesn't gives a 500 error.
assert "500 Internal Server Error" not in driver.page_source
# Confirm the footer is on the page.
footer = driver.find_elements_by_class_name('footer-body')
assert len(footer) > 0, "Footer couldn't be found."
```

Use the Session Status to Find Script Issues

The two session statuses BROKEN and FAILED indicate that the script had an issue. If the session status is BROKEN, it means that your script threw an uncaught exception. You’ll most likely need to review your code and look for errors. See Exceptions for a list of common exceptions.
If the session state is FAILED, an assert statement failed or if you checked “Fail on page load error” and the page failed to load. You'll need to determine why your assertion statement failed or why the page couldn’t be loaded.

You can view the results of assertion statements in the session status detail and script output. From the **Session Details** page, you can view failed assert statements in a red exception box at the top and from the **Waterfall** tab:

Failed assert statements will also appear in the **Script Output** dialog:

![Script Output](image-url)
Select Client Certificates

On this page:

- Install the pyautogui Library
- Select the Only Client Certificate
- Select Certificate from List

This page is solely intended for use with the Private Synthetic Agent. You will not be able to run some of the synthetic scripts below with the Hosted Synthetic Agent.

When using client certificates, you may need to select the certificate from a dialog. To do this, you will need to install the Python pyautogui library on the host machine of your Private Synthetic Agent. The pyautogui library enables you to navigate the UI to select a client certificate.

Install the pyautogui Library

To install the PyAutoGUI Library:

1. Log in to the host machine for the Private Synthetic Agent as the agent_user.
2. Open a PowerShell console.
3. Use pip to install the library: > pip install pyautogui

Select the Only Client Certificate

When you manually navigate to a site that uses client certification, IE10 may prompt you with an alert dialog to select one client certificates as seen below:

To select the client certificate from a synthetic script, your code could focus on the alert dialog and accept the given client certificate as performed by the synthetic script below.
Select Certificate from List

In the case where your synthetic job needs to select one client certificate from a list presented in the alert dialog as shown below, you can use the PyAutoGUI Library to select and accept the client certificate that you want.

In the example below, you use `switch_to_alert` to focus on the alert dialog, and then use the `pyautogui` library to navigate to the correct certificate (tab), select it (enter), and then navigate to the OK button to select it (tab, enter).

```python
import pyautogui
pyautogui.FAILSAFE = False
sslUrl = "https://yourdomain.com/"
driver.get(sslUrl)
driver.switch_to_alert()
pyautogui.press(['tab', 'enter', 'tab', 'enter'])
```

You may have many more client certificates to choose from, so you will have to figure out the best way to use the PyAutoGUI APIs to navigate around the alert dialogue to select and accept the correct client certificate. See Keyboard Control Functions in the PyAutoGUI documentation for more information.
Synthetic Agent Locations

On this page:

- Security Considerations for Scripting
- How This Affects Your Synthetic Jobs
- Mapping of Execution Containers and Browser Locations

The primary purpose of Browser Synthetic Monitoring is to perform browser-based testing. To help you test browsers in the locations of your users, Browser Synthetic provides agents in locations around the world. Your scripts, however, are run in a separate location for the security reasons discussed in the next section.

**Security Considerations for Scripting**

The browsers configured in synthetic jobs are run on shared Windows machines. Because synthetic scripts are written in native Python, they are not run on the shared Windows machines. Instead, each synthetic script is executed in isolation within a Docker container in a location mapped to the configured browser location. This isolation of the agents and scripts guarantees the safety of the data handled by the scripts and has no effect on the performance of the agent.

**How This Affects Your Synthetic Jobs**

You should understand how the locations of the agent/browser and where the script is executed may affect your synthetic job. For example, if the location of the execution container and the configured browser are not in the same region, you may expect to see some network latency. Also, if you are using the Python Requests library to make HTTP requests, you should know where the request is being made from.

The following sections provide some insight as to how your job may be affected.

**URL Measurements**

URL Measurements are used to test site availability and will not be affected because no script is being executed.

**Synthetic Scripts**

Your script can use different libraries to make HTTP requests. The WebDriver API is used to mimic the browser making HTTP requests. To make cURL-like requests to test REST APIs or to retrieve data, you can use an HTTP library such as the Requests library. The tables below describe how the script is executed and the types of problems you may encounter.

<table>
<thead>
<tr>
<th>WebDriver HTTP Requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>**Script Execution Flow</td>
</tr>
</tbody>
</table>


1. You add your script to a synthetic job and configure the browser location in the **Controller UI**.

2. The synthetic job is transmitted to the Synthetic servers.

3. When the synthetic job is ready to run:
   a. A Docker container is created.
   b. The Docker container communicates with agents in selected regions.
   c. The script is executed within the container.

4. The script commands are run in a browser using the Remote WebDriver.

5. After the script finishes running, the agent collects browser metrics and sends them to the Controller UI.

6. The **Controller UI** receives and displays the metrics from your synthetic job.

The **WebDriver API** is used to test the functionality of web pages. You may experience network latency between the execution of the script and the results if the locations of the Docker container and the browser are different.

Because the script is running in a different location, network latency may increase the time to complete the execution of the script. The browser test results collected through this process, however, are **not** affected by this latency, and thus, the results will be reliable.

### Other HTTP Libraries

<table>
<thead>
<tr>
<th>Script Execution Flow</th>
<th>Potential Issues</th>
</tr>
</thead>
</table>
1. You add your script to a synthetic job and configure the browser location in the **Controller UI**.

2. The synthetic job is transmitted to the Synthetic servers.

3. When the synthetic job is ready to run:
   a. A Docker container is created.
   b. The Docker container communicates with agents in selected regions.
   c. The script is executed within the container.

4. After the script finishes running, the agent sends the script output to the Controller UI.

5. The **Controller UI** receives and displays the output from your synthetic job in the script console.

If you are using something other than WebDriver to make HTTP requests, such as the **Requests library**, you should know where the HTTP request is being made from.

You should also understand that the Controller UI only captures browser metrics, so you may want to capture output from your requests in the script console.

### Mapping of Execution Containers and Browser Locations

Script jobs configured to run in the browser locations listed below will be executed in the container location given in the same row.

<table>
<thead>
<tr>
<th>Browser Locations</th>
<th>Container Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ashburn, Virginia</td>
<td>US East (N. Virginia)</td>
</tr>
<tr>
<td>Houston, Texas</td>
<td></td>
</tr>
<tr>
<td>Montreal, Québec</td>
<td></td>
</tr>
<tr>
<td>Toronto, Ontario</td>
<td></td>
</tr>
<tr>
<td>Amsterdam, Netherlands</td>
<td>EU (Ireland)</td>
</tr>
<tr>
<td>Dublin, Ireland</td>
<td></td>
</tr>
<tr>
<td>Frankfurt, Germany</td>
<td></td>
</tr>
<tr>
<td>London, United Kingdom</td>
<td></td>
</tr>
<tr>
<td>Milan, Italy</td>
<td></td>
</tr>
<tr>
<td>Paris, France</td>
<td></td>
</tr>
<tr>
<td>Hong Kong, China</td>
<td>Asia Pacific (Tokyo)</td>
</tr>
<tr>
<td>Seoul, South Korea</td>
<td></td>
</tr>
<tr>
<td>Tokyo, Japan</td>
<td></td>
</tr>
<tr>
<td>Location</td>
<td>Region</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Chennai, India</td>
<td>Asia Pacific (Singapore)</td>
</tr>
<tr>
<td>Mumbai, India</td>
<td></td>
</tr>
<tr>
<td>Singapore</td>
<td></td>
</tr>
<tr>
<td>Melbourne, Australia</td>
<td>Asia Pacific (Sydney)</td>
</tr>
<tr>
<td>Sydney, Australia</td>
<td></td>
</tr>
<tr>
<td>Boardman, Oregon</td>
<td>US West (Oregon)</td>
</tr>
<tr>
<td>Dallas, Texas</td>
<td></td>
</tr>
<tr>
<td>San Francisco, California</td>
<td></td>
</tr>
<tr>
<td>Sao Paulo, Brazil</td>
<td></td>
</tr>
<tr>
<td>Querétaro, Mexico</td>
<td></td>
</tr>
</tbody>
</table>
The Synthetic Credential Vault securely stores credentials used for synthetic jobs. When you create a synthetic job with a synthetic script, you can retrieve a stored key/value credential pair, like a username and password, and inject the credential into the synthetic script. Then when the synthetic job runs, the synthetic script executes and retrieves the credentials stored in the Synthetic Credential Vault.

To access the Synthetic Credential Vault in the Controller, go to the gear icon (⚙️) > Tools > Manage Synthetic Credentials.

**Support**

This table describes the requirements and limitations of the Synthetic Credential Vault.

<table>
<thead>
<tr>
<th>Type</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deployment</td>
<td>Only available for SaaS customers.</td>
</tr>
<tr>
<td>Accounts</td>
<td>Limited to 1000 credentials per EUM account.</td>
</tr>
<tr>
<td>Synthetic jobs</td>
<td>Limited to 100 credentials per synthetic job.</td>
</tr>
<tr>
<td>Credential</td>
<td>Past versions of credentials are not stored. Once you update a credential, you cannot retrieve the previous version.</td>
</tr>
<tr>
<td>User level access</td>
<td>All users can view keys, users, timestamps, and associated applications.</td>
</tr>
<tr>
<td></td>
<td>Only admin-level users can:</td>
</tr>
<tr>
<td></td>
<td>• toggle hide/show values in the Controller UI.</td>
</tr>
<tr>
<td></td>
<td>• add, edit, and delete credentials.</td>
</tr>
</tbody>
</table>

**How to Use**

You can use the Synthetic Credential Vault in the following way:

1. Add a credential in key-value pair to the Synthetic Credential Vault.
2. Create a synthetic job that is using a synthetic script.
3. Inject the credential key into the appropriate place in your synthetic script. See the example from Use Synthetic Credentials in a Synthetic Script.
4. The synthetic job will run and retrieve the credential value associated with the stored credential key.

**Syntax Rules**

This table describes credential syntax rules for the Synthetic Credential Vault and synthetic scripts.
Create Credentials

In the Synthetic Credential Vault, you can add a single or multiple credentials.

**Add a single credential**

These instructions demonstrate how to add a single credential for a sample email and password.

1. Navigate to the gear icon (⚙️) > Tools > Manage Synthetic Credential.
2. Click Add.
3. Enter a key and value. See Syntax Rules.
4. (Optional) Select an associated application. When you search for credentials in the synthetic script, associated credentials are suggested first.

   ![Add/Edit Dialog](image)

   **Key**
   - email_key

   **Value**
   - user123@email.com

   **Application**
   - Optional

   ![Add/Cancel Buttons](image)

**Add multiple credentials**

You can add multiple credentials at once using the Import button. Make sure the imported credentials follow Syntax Rules.

These instructions demonstrate how to add a single credential for a sample email and password.

1. Navigate to the gear icon (⚙️) > Tools > Manage Synthetic Credentials.
2. Click Import.
3. Copy and paste credentials or type them line by line. The maximum number of credentials that can be imported at once is 250.

   ![Import Dialog](image)

   **Application**
   - Optional

   ![Cancel/OK Buttons](image)

Note: The associated application option is not available for bulk import. You must edit each credential individually to specify the associated application.
**Manage Credentials**

In the Synthetic Credential Vault, you can edit, delete, sort and search credentials.

**Edit Credentials**

With admin-level access, you can edit the associated application for existing and imported credentials. However, you can only edit one credential at a time.

**Delete Credentials**

With admin-level access, you can remove one or more credentials at a time.

**Sort Credentials**

You can sort credentials alphabetically by clicking on the key name, user, and timestamp columns.

**Search Credentials**

You can search credentials by key name, user, and application.

**Use Credentials in a Synthetic Script**

You can retrieve credentials stored in the Synthetic Credential Vault with a synthetic script. Credentials associated to a specific application appear first when you start typing the key.

1. To create a synthetic job in the Controller UI, go to **User Experience > Jobs > Add**.
2. Add your synthetic script. See **Write Your First Script** for more information.
3. Locate the DOM element.
4. Enter the corresponding credential using the syntax "<%key%>".

**Example**

```plaintext
email_key=user123@email.com
password_key=Password123
```

**Synthetic Script**

```python
driver.find_element_by_id("email").send_keys("<%email_key%>")
driver.find_element_by_id("pass").send_keys("<%password_key%>")
```
Alerts for Browser Synthetic Monitoring

On this page:
- Synthetic Event Types
- Synthetic Policies
- Site Availability
- Performance Issues

Related pages:
- Policies
- Health Rules
- Actions

You can set policies and alerts based on synthetic events, or you can set thresholds to trigger alerts when you create or edit a synthetic job.

Synthetic Event Types

The following two types of synthetic events are supported and can be used to trigger policies and actions:
- Synthetic Availability
- Synthetic Performance

You can set policies with alerts for these synthetic events, selecting either or both warning or critical conditions.

Synthetic Policies

You can use Synthetic Availability and Performance events to trigger policies, which can, for example, send an alerting email. See Configure Policies for more information.

Create Policy for Synthetic Availability

You can select warning and/or error events for policies for Synthetic Availability.

1. From the Browser App menu, click Alert & Respond.
2. Click Policies.
3. From the Policies page, click Create Policy Manually.
4. From the Create Policy dialog, enter a name for your policy.
5. Check the Synthetic Availability checkbox to select all the Synthetic Availability events or check one or more specific events.
5. Click Next.
6. Create an action to be executed when any of the Synthetic Availability events are triggered. See Actions for more information.

Site Availability

Availability is based on the session status:

- Single URL jobs
  - Warning = 4xx response, (session status is either WARNING or BROKEN)
  - Error = 5xx response, timeout, cannot contact application (FAILED)
- Scripted jobs
  - Warning = your script threw an uncaught exception (BROKEN) or the session status is WARNING.
  - Error = your script had an assertion failure or timed out (FAILED)

Performance Issues

Performance events are based on the criteria you set up when you set up scheduled jobs.

**Create Policy for Synthetic Performance Events**

You can select warning and/or error events for policies for Synthetic Performance.

1. From the Browser App menu, click Alert & Respond.
2. Click Policies.
3. From the Policies page, click Create Policy Manually.
4. From the Create Policy dialog, enter a name for your policy.
5. Check the Synthetic Performance check box to select all the Synthetic Performance events or check one or more specific events.
6. Click Next.
7. Create an action to be executed when any of the Synthetic Performance events are triggered. See Actions for more information.

**Configure Performance Thresholds**
You can also configure performance thresholds that will trigger events when the thresholds are exceeded.

1. Click + Add to add a new job.
2. In the New Job popup, scroll down to Configure Performance Thresholds.
3. From either Trigger a warning event when or Trigger a critical event when:
   - Click + Add Threshold to add a performance threshold for triggering a warning/critical event.
   - Select a performance threshold, enter a value that will trigger a warning event.
   - Select when the event should fire—immediately or after 2, 3, 5, or 10 consecutive failures.

4. You can select multiple performance thresholds by clicking + Add Threshold multiple times.

**Performance Thresholds**

The following are the supported performance thresholds. Follow the links for descriptions of the performance thresholds.

- End User Response Time
- Fully Loaded Time
- Visually Complete Time
- First Byte Time
- Start Render Time
- Bytes per Pageview
Synthetic Sessions

On this page:
- Session Status
- Session UI
- Analyze Session Results

Related pages:
- Browser Synthetic Metrics

You use the Session page to analyze synthetic sessions. A synthetic session is a record of the complete sequence of events that make up one synthetic test. A session can cover one page or, in the case of scripted jobs, can extend to multiple pages. All session timestamps are in local time, regardless of the configured scheduled job's timezone.

A session begins at the first navigation start in the sequence and continues until the last page is fully loaded. In the case of single-page sessions, the session begins and ends when the page is fully loaded (the session duration represented by the metric Visually Complete). Like Browser RUM Analyze and Network Request Analyze, the Synthetic Sessions view enables you to analyze results from all synthetic sessions.

For single-page applications (SPAs), sessions will include the base page and its virtual pages as one page: the base page.

Session Status

Sessions can have one of four statuses:

- **OK**: The test ran successfully and returned data.
- **WARNING**: The test ran successfully, but there were inaccessible resources.
- **BROKEN**: Your job has an error. You can find the error in the script console output.
  - In single URL jobs, the test received a 4xx response.
  - In scripted jobs, your script threw an uncaught exception. This can, for example, happen if you try to perform an action, like a click, on an element which is not on the page you are testing.
- **FAILED**: The synthetic test has detected a problem with your site.
  - In single URL jobs, one of the following occurred:
    - the browser couldn't reach the URL
    - the browser received a 5xx response
    - the session timed out
  - In scripted jobs, your script had an assertion failure or timed out.

Session UI

Records

The Records tab enables you to scan individual requests and allows you to filter and sort to get exactly the data set in which you are interested.
Session Details

Select a specific session and click View Details to see detailed information, including a page load waterfall and tabular details for resources. In multi-page sessions, the left navigation allows you to select each page from the sequence.

Business Transactions

When a business and a browser application have correlated business transactions, you can also create synthetic jobs from that browser application that makes requests to the business application to get business transactions in synthetic sessions. You can view business transactions in the Waterfall and the Business Transactions tabs of the Session Details dialog.
If you see View Snapshot next to resource in the waterfall, that indicates that there was a correlated business transaction. Clicking it will take you to the business transaction snapshot from the backend.

You can also click the Business Transactions tab to see a list of the server-side business transactions this request initiated. Click through to the related Business Transaction dashboards.

Script Output

If you used a script, you can click through to see any script output.
Session Screenshots

Synthetic captures screenshots of the browser as your job runs. By default, it takes one screenshot per page, when the page is finished rendering. Screenshots are retained for 30 days.

In scripted jobs, you can call an API from your script to take a maximum of 10 screenshots per page. The screenshot will be of the visibly complete content in the viewport (won’t include the browser chrome).

The Screenshots tab shown below displays the screenshot as a 200 x 150 image. The resolution of the screenshot will be the same as the device where the screenshot was taken.

The Waterfall tab will contain the visually complete screenshot and any custom screenshots (e.g., screenshot2 in the screenshot below) triggered by the script.


**Charts**

The **Charts** tab provides you with a set of widgets that offer predefined visualizations of the data set you have created, allowing you to create a dashboard.

You can delete, re-add, resize, and drag-and-drop to move all of the widgets.

**Analyze Session Results**

You understand the results of your synthetic jobs through sessions. From session details, you can determine if there were errors, what type of errors, and diagnose what factors are affecting performance. The following sections show you how to analyze synthetic session results for errors and performance issues.

**Errors**

**Determine Type of Error**

At this time, you should be able to determine the type of error by the **Status** and **Error Type** fields. The **session status** indicates whether your jobs are passing, failing (there is a problem with the system being tested), or broken (there is a problem with the job itself). The **Error Type** field gives more information about the cause of the error.

**Understand the Error**

1. From the **Sessions > Records** tab, select a session with an error and click **View Details**.
2. From the **Session Details** dialog, you can view when the error occurred and the returned error message in the **Waterfall** tab.
3. If your job used a custom script, you can click **Show Script Output** to view the script results. You can download or copy the results as well.
4. You can also view the screenshot to see how the error might be manifested to end users. You might want to display the errors in a more user-friendly way and possibly omit error details.

**Performance**

One of the best ways to see if your job is having performance issues is to view widgets representing aggregated metrics in the **Charts** tab. With a quick glance, you can see if there are site availability issues, where end users might be having bad user experiences, how long sessions are lasting, and how long it's taking for pages to be visually complete.

**Network Latency**
The Waterfall tab shows you a timeline of when resources are loaded and plots the events First Byte, DOM Ready, Load Event, and Visually Complete that mark significant and discreet moments from the time the browser receives data until the user can see a complete page.

To understand session performance, you can compare the end user response time (the same as the onload time) of a page with the session duration. This gives you an idea of which pages are taking too long to load and how that might be affecting the session.

You can also click one of the bars to determine if a resource is taking a long time to load. For example, if an image is taking too long to load, you can take actions to optimize the loading time such as caching the image, reducing its size, confirm that the image is the size that you’re displaying it, or not use the image at all.

**Problematic Resources**

The Session Details dialog shows you the final session state and the session state set by each page. The final session state is indicated in the top-left corner, whereas, the session state set by each page is shown in the Page Views in Session sidebar. In the screenshot below, the icon indicates that the first page set the session status to “Warning”. From the Waterfall tab, you can also discover which resources didn’t load, either marked in yellow for the “Warning” state or in red for the “Failed” state.

The Resources tab shows you the number of resources by category, the domains requested, and key performance metrics that you can use to sort the results. You can quickly identify resources that are slowing the page load and potential DNS problems.
You can also configure your synthetic jobs to check the availability of resources. You can set the session status to **Failed** or **Warning** when resources are unavailable. See [Configure Availability Rules](#) for configuration instructions. When the session status is set to **Failed** or **Warning**, Browser Synthetic Monitoring triggers corresponding events that can be used to create alerts. This enables you to be alerted when resources are missing. See [Alerts for Browser Synthetic Monitoring](#) to learn how to create availability policies.

### View the Effects of Performance Issue

Screenshots will show you what your user is seeing. From the screenshot, you can determine whether the content has completely loaded (text, images, containers, UI components) and the page is styled correctly (CSS).

From the **Waterfall** tab, you can determine when the screenshot (visually complete) and view a small thumbnail by mousing over the camera icon.

By navigating to the **Screenshots** tab and double-clicking the screenshot, you can view a larger screenshot to see details such as the styling, UI components, and content.
Synthetic Pages

Related pages:
- Browser RUM Pages & Ajax Requests
- Configure Page Identification and Naming

You use the Pages view to see an aggregated view of how each page visited in your jobs is performing over time.

The metrics shown in the Pages view are aggregated across all jobs, so if multiple jobs visit the same page, their metrics will be aggregated.

Pages Details

Just as in the Browser RUM view you can filter the dataset and manage which pages are displayed and which metrics are shown. If you select an individual page and click Details, you are shown a Page dashboard very similar to the Browser RUM Pages Dashboard.

The Synthetic Pages Dashboard, however, includes some additional metrics. To understand these added synthetic metrics, you can hover over the metric name and a tooltip explaining the metric popup. For more information, see Browser Synthetic Metrics.
Request Synthetic Snapshots On-Demand

Browser Synthetic On-Demand allows you to take quick snapshots of single pages whenever you wish.

Create an On-Demand Snapshot

To create an on-demand session, click the + Generate a Snapshot button at the top of the On-Demand list page.

After the Generate a Synthetic Snapshot popup appears:

1. Provide the URL of the page you wish to test.
2. From the dropdown, select where the request should originate and which browser should be used.
3. Click Generate Snapshot.

View On-Demand Snapshot Results

When the test completes, it is marked as Done in the Snapshot list. To see the detailed results, click the test. The Summary tab provides a waterfall of the page load and additional request details.
The Resource Details tab provides visualizations of Resources by Type and Domains Requested and a waterfall displaying the download of every resource. Click a waterfall bubble to see connection details. Domains Requested is also presented as a table.
Browser Synthetic Metrics

Many of the metrics collected by Browser Synthetic are identical to those collected by Browser RUM. Additional synthetic-only metrics are defined in Browser Synthetic Metrics Defined.

Key Browser RUM metrics are displayed on the Geo, Sessions, and Pages dashboards. They can also be seen in the Metric Browser, just like RUM metrics, giving you the ability to view these metrics in the context of Ajax requests, iframes, applications, base pages, browsers, devices, and geographic locations. Metrics will only appear in the Metric Browser if there is synthetic data. For example, if your synthetic jobs do not have failed sessions, the metric Failed Sessions will not appear.
There are also metrics for specific tests and, under the **Synthetic Jobs** heading, for Synthetic job health overall.
## Browser Synthetic Metrics Defined

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Availability</strong></td>
<td>The ratio of <strong>OKAY</strong> sessions (successful) to all sessions, expressed per-million: 0 (for all failed) to 1,000,000 (for all succeeded). The ratio is calculated using the following: (# of successful sessions/# of total sessions) * 1,000,000. Availability is represented in per-million instead of as a percentage because Controller metrics are given as integers, making expressing the availability as a percentage less precise. For example, although the availability percentage in actuality might be 99.99%, it would only be able to be expressed as an integer such as 100% or 99%. To convert per-million to a percentage, just divide by Availability (ppm) by 10,000. For example, if Availability (ppm) is 999,800, then Availability (%) is 99.98%. Most Controller UI components display metrics as percentages, but the Metric Browser, which shows &quot;raw data&quot;, shows the value per-million.</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Billable Time</td>
<td>The seconds that were billed against your Browser Synthetic license.</td>
</tr>
<tr>
<td>Broken Sessions</td>
<td>The number of sessions that completed with the session status <strong>BROKEN</strong>, indicating that the job has an error. You can find the error in the script console output.</td>
</tr>
<tr>
<td>Data Downloaded Before Fully Loaded</td>
<td>The bytes of data downloaded by the browser between the navigation start of a page and the navigation start of the next page or the end of the session. In particular, includes any resources loaded after the page load event.</td>
</tr>
<tr>
<td>Data Downloaded Before Onload</td>
<td>The bytes of data downloaded by the browser between the navigation start for a page and the load event for that page.</td>
</tr>
<tr>
<td>DOM Elements per Pageview</td>
<td>The number of DOM elements present on a page at the time of the page's load event. Includes sub-documents in frames.</td>
</tr>
<tr>
<td>Failed Sessions</td>
<td>The number of sessions that completed with the session status <strong>FAILED</strong>, indicating the synthetic test has detected a problem with your site. See Session Status for more information.</td>
</tr>
<tr>
<td>Fully Loaded Time</td>
<td>The time from the start of the initial navigation until 2 seconds of no network activity has passed after Document Complete. This will usually include any activity that is triggered by JavaScript after the main page loads. The Fully Loaded Time is given in milliseconds and percentiles.</td>
</tr>
<tr>
<td>Job Executions</td>
<td>Total number of sessions (per browser, per location) for a given job.</td>
</tr>
<tr>
<td>Number of Resources Loaded Before Fully Loaded</td>
<td>See Data Downloaded Before Fully Loaded above.</td>
</tr>
<tr>
<td>Number of Resources Loaded Before Onload</td>
<td>See Data Downloaded Before Onload above.</td>
</tr>
<tr>
<td>Ok Sessions</td>
<td>The number of sessions that completed with the session status <strong>OK</strong>, indicating the synthetic job ran successfully and returned data.</td>
</tr>
<tr>
<td>Pages per Session</td>
<td>The number of pages per session.</td>
</tr>
<tr>
<td>Session Duration</td>
<td>For scripted jobs: the time needed to execute the script from the first line to the last line. For URL jobs: the time from when the browser initiated the first HTTP request until the last resource was loaded.</td>
</tr>
<tr>
<td>Sessions with Resource Errors</td>
<td>The number of sessions that had missing or inaccessible resources.</td>
</tr>
<tr>
<td>Session Visual Time (SVT)</td>
<td>This metric captures the time the browser spent loading pages. Relative to Session Duration, this excludes the script process time and the time your script was interacting with content on a given page.</td>
</tr>
<tr>
<td>Speed Index</td>
<td>This metric captures whether a page renders incrementally or all at once. For example, if two pages both take five seconds to finish rendering, but one is mostly rendered after a second, while the other is blank until the last moment. Then the first page will have a Speed Index close to 1000, while the second will have a Speed Index close to 5000 (the Visually Complete time in milliseconds). More information on how it is calculated is available in the WebPageTest documentation.</td>
</tr>
<tr>
<td>Start Render Time</td>
<td>The time from the start of the initial navigation until the first non-white content is rendered in the browser display in milliseconds.</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Visually Complete</td>
<td>The time when an end-user would determine that the page is visually complete in the viewport. This time is calculated by using the last visual change to the page and is in milliseconds. The page represents the viewable content (the viewport) in a browser window. The default size for a browser window is 1024 x 768.</td>
</tr>
<tr>
<td>Warning Sessions</td>
<td>The number of sessions that completed with the session status WARNING, indicating the test ran successfully, but there were inaccessible resources.</td>
</tr>
</tbody>
</table>
Browser Synthetic Licenses

On this page:
- Synthetic Agent Licenses / Editions
- Billing Cycle
- License Key
- Check Usage

This topic describes how to interpret the details of your Browser Synthetic licensing information. It assumes that you have acquired your Browser Synthetic license from your AppDynamics sales representative.

Synthetic Agent Licenses / Editions

There are two Synthetic Agent licenses: Synthetic Hosted Agent and Synthetic Private Agent.

The table below provides an overview of the two licenses and the available editions:

<table>
<thead>
<tr>
<th>Synthetic Agent License</th>
<th>Available Editions</th>
<th>License Unit</th>
<th>Description</th>
</tr>
</thead>
</table>
| Synthetic Hosted Agent  | Pro                | Synthetic Time Blocks (STBs) | Synthetic Hosted Agent licensing is based on units known as Synthetic Time Blocks (STBs). For license details, see the following entry on the License Entitlements and Restrictions page:  
  - Browser Synthetic Monitoring - Hosted Agent (SaaS) |
|                         | Lite Edition       | Synthetic Time Blocks (STBs) | A Lite Edition license gives you 4,000 Synthetic Time Blocks (STBs) per month and only allows you to create on-demand jobs. |
| Synthetic Private Agent | Pro                | Pages × Locations            | The Synthetic Private Agent licensing is based on page counts and locations (Pro) or just pages (Unlimited Locations). One Synthetic Private Agent license can only have either the Pro or the Unlimited Locations edition, not both. |
|                         | Unlimited Locations | Pages                    | Locations are user-defined. Add locations to get visibility from different offices or facilities. You can add multiple servers ("agents") to a location to increase capacity: This doesn’t impact your license.  
  For the Pro Edition, license units are calculated by multiplying the number of visited pages by the number of locations (round-robin or not). For example, a synthetic job visiting two pages in three locations would use six license units.  
  For the Unlimited Locations Edition, the license units are simply the number of visited pages. Thus, a synthetic job visiting two pages in three locations would simply use two license units.  
  For license details, see the following entries on the License Entitlements and Restrictions page:  
  - Browser Synthetic Monitoring - Private Agent (SaaS)  
  - Browser Synthetic Monitoring - Private Agent (Unlimited Locations) |

Additional License Details

To help you optimize the use of your Synthetic Agent license(s), keep the following in mind:

- Scheduled jobs are measured in terms of the time it takes to actually run the test in our systems. Because it is not possible to know in advance how long a given test may take, you can set a timeout value when you create the job, to place an upper limit on the time to be used for a test. If the test times out, you receive partial results.
- Once you have used up your monthly allotment, all your scheduled jobs are paused until the following month.
- If there is a problem accessing a site (receiving, for example, a 5xx error), the system automatically re-runs the test. You are charged for these retests.
- If you see a session with the status INTERNAL_ERROR, this indicates a problem with AppDynamics’ infrastructure, and you will not be charged—even if the session contains partial results.
- The browsers, frequency, duration, and other configuration of the synthetic job do not impact the license.
Billing Cycle

Browser Synthetic licenses are managed per month, and allotments are reset on the first of every month.

License Key

This is the license key associated with your EUM Account. It is used to associate your data with your account.

Check Usage

As your jobs run, your usage is tracked and the STBs you have already consumed and the STBs you are projected to consume in the current usage period are calculated. To see the current status of your account, check the Percent Quota Used line at the top of the Jobs window.

Jobs

Hosted Agent license used: 12%. Private Agent license used: 5%.  

To see more details, click the line. The Synthetic Usage popup appears.

Hosted Agent Usage:

- Usage Period 09/01/18 11:00:00 AM - 10/01/18 12:00:00 PM
- Blocks Allocated 520,000
- Blocks Consumed 62,594 (12%)
- Blocks Projected Projection unavailable

Private Agent Usage:

- Units Allocated 20
- Units Utilized 1 (5%)
Mobile Real User Monitoring (Mobile RUM) allows you to understand your native iOS, Android, Xamarin, Cordova-based, or React Native mobile application as your end users actually use it.

It provides you with visibility into the functioning of the application itself and the application's interactions with the network it uses and any server-side applications it may talk to. This page introduces Mobile RUM.

Benefits of Mobile RUM

With Mobile RUM, you can:

- **Understand and improve your mobile application’s performance**
  - Know when your application is slow because of networking problems. See Network Requests List.
  - Determine whether a request is slow because of your servers or because of the network connection. See Network Request Dashboard.
  - Trace an individual request from the initial user action in the mobile application through the associated business transaction(s) on the application server(s). See Network Request Snapshots.
  - Estimate the network performance you can expect for different requests, carriers, devices, and geographies by viewing current metrics. See Usage Stats in Mobile App Dashboard.
  - Detect slow UI issues known as “application not responding” (ANR). You can track ANRs in the Code Issues Dashboard, identify sessions where ANRs occurred, and configure ANR thresholds to define severity levels.

- **Reduce crashes**
  - Observe when your application crashes and what caused each crash. See Crash Analyze and Crash Snapshots.
  - Learn which environments experience the most crashes. See Crash Dashboard.

- **Reduce errors**
  - Observe how many network errors occur and which requests caused them. See Network Requests List and Network Request Dashboard.
  - Capture errors and exceptions from your mobile applications, so that they are displayed in the Code Issues Dashboard and the session records and details. To learn how to use the SDKs to report errors, see Report and Exceptions (iOS/Android).

- **Learn about your users**
  - Learn which devices and technologies most of your users are running. See Usage Stats in Mobile App Dashboard and Crash Dashboard.
  - View where your users are located in the world. See Geo Dashboard in Mobile App Dashboard.
  - Understand how users are navigating through your application and what actions they are taking. See Mobile Sessions.
  - Capture user interactions with your mobile application such as button clicks and when text is edited. See "Enable User Interaction Capture Mode" (iOS/Android) to learn how.

- **Customize the data that Mobile RUM returns**
  - Create new metrics and timers. See Customize the iOS Instrumentation, Customize the Android Instrumentation, Customize the Xamarin Instrumentation, and Customize the Cordova Instrumentation.
  - Collect diagnostic data from your executing code via information points. See Customize the iOS Instrumentation, Customize the Android Instrumentation, Customize the Xamarin Instrumentation, and Customize the Cordova Instrumentation.
How It Works

To use Mobile RUM, you add a small piece of highly performant code, the Mobile Agent, to the source of your mobile application. This process is called **instrumenting** and is described in [Instrument iOS Applications](#), [Instrument Android Applications](#), [Instrument Xamarin Applications](#), [Instrument Cordova Applications](#), and [Instrument React Native Applications](#). As your end users interact with your application, the agent collects metrics and any error information on the application's performance and sends that information to the EUM Server, where it is processed. The EUM Server then makes the data available to the AppDynamics controller UI, where it is displayed in a series of dashboards and charts.

If your application crashes, the agent also creates a crash snapshot with information to help you analyze what happened, including the crashed function, the source file containing the crashed function, the line number, if available, and a stack trace of the application at the time of the crash, along with various other identifying data. And if the server-side application with which your mobile app interacts is also instrumented, you can get correlated metric information for the entire round-trip request life cycle.

Data Collection

When the agent detects that something of interest has occurred in the application, it collects data describing that event and stores it in a local buffer. Typically, this data consists of approximately 200-400 bytes per event. The agent periodically flushes this local buffer by sending the events in the buffer to the EUM Server in a request called a **beacon**. To minimize power consumption, the agent schedules these flushes, if possible, when the application is using the network.

If a network connection to the EUM Server is not available, the agent deletes older events to prevent unbounded growth in the buffer. The maximum size of the buffer is 200 events.

License and Enable Mobile RUM

Mobile RUM requires a separate license from that for your application server and for Browser Real User and Synthetic Monitoring and must be enabled before it is available for use.

For information about licensing, including a description of the two types of licenses, Lite and Pro, see [Mobile RUM Licenses](#).

For information on enabling or disabling Mobile RUM monitoring, see [Enable Mobile RUM on Your Controller](#).

AppDynamics Mobile RUM allows you to investigate the performance of your mobile application. [AppDynamics Mobile App](#) allows you to monitor your Controller from your mobile device.
Hybrid Application Support

On this page:
- What Are Hybrid Applications?
- Hybrid Use Cases
- Instrumentation Steps
- How it Works
- Mobile Agent and JavaScript Agent Compatibility
- Hybrid Support Limitations

AppDynamics supports monitoring for hybrid applications based on the iOS/Android native SDK or any Cordova-based framework.

What Are Hybrid Applications?

Hybrid applications are web applications running inside an Android or iOS native shell. The web application can be in the form of an Android WebView, an iOS WKWebView, or a Cordova web application that consists of HTML, CSS, JavaScript, and other resources just like a conventional web application.

Hybrid Use Cases

Mobile RUM supports the following two use cases.

- Native Mobile applications that run web applications or web views. Examples of this are Android WebViews and iOS WKWebView.
- Applications written in Cordova-based frameworks, such as Ionic, that provide a native shell for running a web application and an interface for native mobile APIs.

Instrumentation Steps

To monitor your hybrid applications, you will need to set up and access Mobile RUM and follow the steps below based on your hybrid use case:

<table>
<thead>
<tr>
<th>Native iOS Application</th>
<th>Native Android Application</th>
<th>Cordova-Based Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Instrument an iOS Application</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Configure Hybrid Support for the iOS Application</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Configure the Android Build</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Instrument an Android Application</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Configure Hybrid Support for Android Applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Instrument a Cordova Application</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Customize the Cordova Instrumentation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How it Works

For native Android/iOS applications, the Android Agent or iOS Agent will inject the JavaScript Agent into an Android WebView or iOS WKWebView. For Cordova-based applications, the AppDynamics plugin will inject the JavaScript Agent into the web application.

The diagrams below provide an overview of how Mobile RUM is used to monitor mobile native applications and Cordova-based applications.
You instrument your mobile application with either the Android/iOS Agent.

When your mobile application instantiates a WebView or WKWebView instance, the Mobile Agent injects the JavaScript Agent into it.

The JavaScript Agent monitors the performance and WebView usage.

The JavaScript Agent reports the application activity by sending beacons to the Mobile Agent.

The Mobile Agent repackages the browser beacons into mobile beacons and sends the beacons to the EUM Server.

Cordova-Based Application Architecture

The AppDynamics Cordova plugin injects the JavaScript Agent into the Cordova WebView.

The Cordova-based application uses the plugin's JavaScript binding to configure the Mobile Agent.

The JavaScript Agent injected into the WebView monitors the application activity and performance.

The JavaScript Agent sends browser beacons to the Mobile Agent.

The Mobile Agent parses and converts the browser beacons into mobile beacons.

The Mobile Agent sends the mobile beacons to the EUM Server.

Mobile Agent and JavaScript Agent Compatibility
When hybrid app support is enabled, the Mobile Agents use an embedded JavaScript Agent to monitor mobile web pages. If you are already monitoring the same mobile web pages with a browser application, the JavaScript Agent version used to instrument the browser application may be different than the JavaScript Agent version used by the Mobile Agent. If the two versions of the JavaScript Agent are incompatible, the Mobile Agent will not be able to instrument web views or access event notifications from web views.

**How to Ensure Compatibility**

To ensure compatibility, browser applications should use JavaScript Agent version 4.5.4+ and mobile applications should use Mobile Agent versions 4.5.5+. When using compatible versions of the JavaScript Agents, the JavaScript Agent of the browser app will report beacons for a web page, and the embedded JavaScript Agent of the mobile app will be ignored and not report beacons because a web page cannot be instrumented with the JavaScript Agent multiple times.

If you have disabled the Mobile Agent's hybrid app support, there will be no compatibility issues regardless of the JavaScript Agent version used by the browser application.

**Troubleshooting Incompatible JavaScript Agent Versions**

If you are using incompatible versions of the JavaScript Agents, do one of the following to correct the problem:

- Upgrade the JavaScript Agent that you are using to instrument the browser application.
- In the mobile instrumentation, disable the hybrid support (iOS/Android), so that the mobile web pages are only instrumented by the browser application instrumented with the older version of the JavaScript Agent.

**Hybrid Support Limitations**

- For mobile native applications, the JavaScript Agent is injected into the top-level WebView or WKWebView instance, not in embedded web pages (iFrames). You have to manually instrument the iFrames with the JavaScript Agent.
- For Cordova-based applications, you have to manually report events using the plugin's JavaScript API.
- Network calls made in the context of the WebView for Cordova-based applications show up as Ajax calls from the JavaScript Agent.
Set Up and Access Mobile RUM

You set up and configure Mobile RUM in two areas:

- Your AppDynamics Controller
- Your iOS, Android, Xamarin, or Cordova application

To prepare Mobile RUM you need to make changes in both these areas in a particular order.

You need to do the following:

1. Check your prerequisites
2. Review Controller capacity
3. Evaluate your mobile application
4. Complete the Getting Started Wizard
5. Instrument and verify your app
6. Configure network request naming and thresholds

Check Your Prerequisites

Before you can use AppDynamics to monitor your mobile application, you need to make sure you have the following prerequisites:

- An AppDynamics account, with access to an AppDynamics Controller
- A Mobile RUM License. See Mobile RUM Licenses.
- Access to your mobile application source code

Review Your Controller Capacity

If you use an on-premises Controller and plan to monitor mobile applications, assess your Controller’s capacity to accommodate the increase in the number of metrics that Mobile RUM will generate. The number of individual metric data points generated depends on the level of activity of your mobile applications. As a rough guide, the use of Mobile RUM can increase the number of metric data points by as much as 15 to 25K per instrumented application, if your applications are heavily accessed by mobile users. The actual number depends on how many network requests your applications receive.

The number of separate RUM metric names saved in the Controller database can be larger than the kinds of individual data points saved. For example, a metric name for a metric for iOS 5 might still be in the database even if all your users have migrated away from iOS 5. So, the metric name would no longer have an impact on resource utilization, but it would count against the default limit in the Controller for metric names per application. The default limit for names is 200,000 for Browser RUM and 100,000 for Mobile RUM.

For more information about Controller sizing, see Hardware Requirements per Performance Profile.

Evaluate Your Mobile Application

By default, Mobile RUM can monitor your application in two ways: network requests and crash reporting.

If you want to collect information on network requests, your application must make HTTP calls using specific classes:
An iOS application must use `NSURLConnection` or `NSURLSession` to generate network requests.

An Android application must itself (i.e., not via an external framework) use `HttpURLConnection`, `HttpsURLConnection`, `HttpClient`, `OkHttp`, or `ch.boye.httpclientandroidlib` to generate network requests.

Using the SDK, custom HTTP libraries can also be monitored and used by the agent itself. See Customize the iOS Instrumentation and Customize the Android Instrumentation for more information.

Even if your application does not generate network requests, or if you do not want to monitor that activity, you can use AppDynamics purely to monitor crashes. You can also extend the Mobile Agents using the SDK to collect other data, like how your application is functioning internally.

**Complete the Getting Started Wizard**

Use the Getting Started Wizard to create a mobile app group, get an EUM App Key, and guide you through the instrumentation process. Once you have successfully completed the steps in the wizard, Mobile RUM will automatically be enabled for the created mobile app group.

Your Controller must have a Mobile RUM license before you can enable the Mobile RUM functionality.

**Disable Mobile RUM**

To later disable/enable Mobile RUM:

1. Open your mobile application.
2. From the left-hand navigation bar, click Configuration.
3. Click Container Configuration >.
4. Toggle the Mobile Monitoring switch to OFF/ON.

**Get Your Application Key**

To find your App Key after you’ve completed the Getting Started Wizard:

1. Open your mobile application.
2. From the left-hand navigation bar, click Configuration.
3. From the Configuration page, you can view the EUM App Key in the Mobile App Group Configuration section as shown below.
Instrument and Verify Your App

The method you follow to instrument your app depends on your platform:

- Instrument iOS Applications
- Instrument Android Applications
- Instrument Xamarin Applications
- Instrument Cordova Applications

Configure Network Request Naming and Thresholds

You can customize how you want the Controller to name your application’s requests and what performance thresholds you want to be in effect. See Configure Mobile Network Request Naming and Thresholds.

External Access

Mobile RUM is made up of several components, any or all of which can either be located on the Internet or hosted inside your own data center/network. On-premises access points are configured at installation or through the UI. If your installation requires access to any of these components on the Internet, see Access the SaaS EUM Server for more information.
Instrument iOS Applications

Before you can monitor your iOS application, you will need to instrument your application to enable the iOS Agent to collect mobile metrics.

After you have set up and accessed Mobile RUM, follow the instructions below:

1. Install the iOS SDK
2. Instrument an iOS Application
3. Customize the iOS Instrumentation (Optional)
4. Troubleshoot the iOS Instrumentation (Optional)
Install the iOS SDK

On this page:
- iOS SDK Install Requirements
- CocoaPods Install
- Manual Install
- Upgrade the iOS SDK

Related pages:
- Instrument an iOS Application
- Customize the iOS Instrumentation
- Troubleshoot the iOS Instrumentation

You can install the iOS SDK using CocoaPods or by manually downloading and installing it. We recommended that you use CocoaPods to install the iOS SDK because it handles the dependencies, the build settings, and simplifies upgrading.

Follow the instructions for your preferred installation method:
- CocoaPods Install
- Manual Install

iOS SDK Install Requirements

To install the iOS SDK, you must be running Xcode 7 or higher.

CocoaPods Install

1. Add the line below to a target block in your Podfile:

   ```ruby
   pod 'AppDynamicsAgent'
   ```

   For example:

   ```ruby
   platform :ios, '8.0'
   use_frameworks!
   
   target 'YourApp' do
   pod 'AppDynamicsAgent'
   end
   ```

2. In your project directory, run this command:
Manual Install

To manually install the iOS SDK, follow the steps below:

1. Download the iOS SDK
2. Add the Framework to the App
3. Add Required Libraries
4. Set the -ObjC Flag

Download the iOS SDK

1. Navigate to the AppDynamics Download page.
2. From the APP AGENT list, check the Mobile RUM Agent - iOS checkbox.
3. Click Download for the latest iOS Agent that will appear in the Releases results. This downloads a file named iOSAgent-<version>.zip.

Add the Framework to the App

Uncompress the .zip file you downloaded: there are two images in the framework file:

ADEUMInstrumentation.framework/Versions/A/ADEUMInstrumentation for Xcode 7 and above
ADEUMInstrumentation.framework/Versions/Xcode6/ADEUMInstrumentation for Xcode 6 and lower.

Add the appropriate framework to the project.

By default the symlinks are set up to use the Xcode 7 library (using Apple conventions):
1. ADEUMInstrumentation.framework/ADEUMInstrumentation -> Versions/Current/ADEUMInstrumentation
2. ADEUMInstrumentation.framework/Versions/Current -> A

Bitcode is part of Apple's App Thinning initiative.

Add Required Libraries

The AppDynamics iOS Agent requires the following libraries:

- SystemConfiguration.framework
- CoreTelephony.framework
- libz.dylib or .tbd
- libsqlite3.dylib or .tbd

To add the libraries

1. Select the target that builds your app in Xcode.
2. Select the **Build Phases** tab.
3. Expand the **Link Binary With Libraries** section.
4. If any of the above libraries are not listed:
   - Click the + button.
   - Locate the missing library in the list.
   - Click **Add**.

Repeat this step for each missing library.

**Set the -ObjC Flag**

You also need to add the `-ObjC` flag to Other Linker Flags.

1. Select your project in the **Project Navigator**.
2. In the target list, select the target that builds your application.
3. Select the **Build Settings** tab.
4. Scroll to **Linking** and open.
5. Go to **Other Linker Flags** and double-click to open the popup.
6. If the `-ObjC` flag is not in your list, click + and add it.

The `-ObjC` flag is necessary because the iOS Agent defines categories with methods that can be called at runtime, and by default, these methods are not loaded by the linker. As a result, you'll get an "unrecognized selector" runtime exception. The use of `-ObjC` ensures the methods will be loaded.

**Upgrade the iOS SDK**

**Upgrade with CocoaPods**

1. In your project directory, run this command: `pod update`
2. Rebuild.

**Upgrade Manually**

To pick up new features or to get crucial bug fixes you want to upgrade the iOS SDK in your app.

1. Download the updated SDK.
2. Replace the `.framework` file using the updated SDK.
3. Rebuild your app.
Instrument an iOS Application

Follow the steps below to get your EUM App Key and instrument your iOS apps.

1. Get Your Application Key
2. Initialize the Agent
3. Generate a dSYM File
4. Monitor Crashes with the dSYM File
5. Customize the Instrumentation (Optional)
6. Configure the iOS Agent for On-Prem Deployments (Optional)

Get Your Application Key

After you completed the Getting Started Wizard, you were given an EUM App Key. You will need this key when you modify the source code. In some cases, multiple mobile applications can share the same key.

If you have completed the Getting Started Wizard, but don't have your EUM App Key, see Get Your Application Key.

Initialize the Agent

Objective-C

Edit your app's AppDelegate file to initialize the Mobile Agent as soon as the app launches. This registers your application and only needs to be done once in your code.

1. In your application's AppDelegate.m file, add this import statement:

   #import <ADEUMInstrumentation/ADEumInstrumentation.h>

2. In the method didFinishLaunchingWithOptions, create an ADEumAgentConfiguration object with the EUM App Key that you received when completing the Getting Started Wizard to initialize the iOS Agent:

   // Example EUM App Key: "AAA-AAB-AUM"
   ADEumAgentConfiguration *config = [[ADEumAgentConfiguration alloc] initWithAppKey:<#EUM_APP_KEY#>];

Your code should look something like the following:
2. Configure the iOS Agent to report metrics and screenshots to the SaaS EUM Server in your region and initialize the agent by passing the ADEumAgentConfiguration object to the method initWithConfiguration. (If you are using an on-premises EUM Server, see Configure the iOS Agent for On-Prem Deployments (Optional) for implementation details.)

3. Configure the iOS Agent to report metrics and screenshots to the SaaS EUM Server in your region and initialize the agent by passing the ADEumAgentConfiguration object to the method initWithConfiguration. (If you are using an on-premises EUM Server, see Configure the iOS Agent for On-Prem Deployments (Optional) for implementation details.)

```swift
#import <ADEUMInstrumentation/ADEumInstrumentation.h>
#import "AppDelegate.h"

// ...
-(BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Example EUM App Key: "AAA-AAB-AUM"
    ADEumAgentConfiguration *config =
        [[ADEumAgentConfiguration alloc]
            initWithAppKey:<#EUM_APP_KEY#>];
    // other tasks
    return YES;
}
```
```swift
#import <ADEUMInstrumentation/ADEumInstrumentation.h>
#import "AppDelegate.h"

// ...
-(BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    ADEumAgentConfiguration *config =
    [[ADEumAgentConfiguration alloc]
    initWithAppKey:<#EUM_APP_KEY#>];
    // The default SaaS EUM Server and
    Screenshot Service are in the Americas,
    // so you can omit the following settings
    if you are in the Americas.
    config.collectorURL =
    @"https://col.eum-appdynamics.com";
    config.screenshotURL =
    @"https://image.eum-appdynamics.com/";
    [ADEumInstrumentation
    initWithConfiguration: config];
    // other tasks
    return YES;
}
```
#import <ADEUMInstrumentation/ADEumInstrumentation.h>
#import "AppDelegate.h"

// ...
-(BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    ADEumAgentConfiguration *config =
        [[ADEumAgentConfiguration alloc]
         initWithAppKey:<#EUM_APP_KEY#>];
    // Configure the iOS Agent to report the
    // metrics and screenshots to the SaaS EUM Server in
    // EMEA
    config.collectorURL = @"https://fra-col.eum-appdynamics.com";
    config.screenshotURL = @"https://fra-image.eum-appdynamics.com/";
    [ADEumInstrumentation
     initWithConfiguration: config];
    // other tasks
    return YES;
}
4. (Optional) If your application is using another tool to report crashes, the iOS Agent will warn you with the following message:

```
Agent has detected a third party crash reporting tool. You may wish to disable AppDynamics Crash Reporting by setting the crashReportingEnabled configuration flag to NO.
```

You are recommended to use only one crash reporting tool for better results. See Disable Crash Reporting for instructions on how to disable the iOS Agent's crash reporting.

5. Save the file.
Swift 4.2

The iOS Agent is compatible with applications created using the Swift programming language.

1. In your application's AppDelegate.swift file, add this import statement:

```swift
import ADEUMInstrumentation
```

2. In your AppDelegate's didFinishLaunchingWithOptions, create an ADeumAgentConfiguration object with the EUM App Key that you received when completing the Getting Started Wizard to initialize the iOS Agent:

```swift
// Example EUM App Key: "AAA-AAB-AUM"
let config = ADEumAgentConfiguration(appKey: <#EUM_APP_KEY#>)
```

3. Configure the iOS Agent to report metrics and screenshots to the SaaS EUM Server in your region and initialize the agent by passing the ADeumAgentConfiguration object to the method initWithConfiguration. (If you are using an on-premises EUM Server, see Configure the iOS Agent for On-Prem Deployments (Optional) for implementation details.)
#import <ADEUMInstrumentation/ADEumInstrumentation.h>
#import "AppDelegate.h"

    // ...
    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        let config = ADEumAgentConfiguration(appKey: <#EUM_APP_KEY#>)
            // The default SaaS EUM Server and Screenshot Service are in
            // the Americas, so you can omit the following settings
            // if you are in the Americas.
            config.collectorURL = "https://col.eum-appdynamics.com"
            config.screenshotUrl = "https://image.eum-appdynamics.com"
        ADEumInstrumentation.initWith(config)
    // other tasks
        return true
    }
#import <ADEUMInstrumentation/ADEumInstrumentation.h>
#import "AppDelegate.h"

    // ...
    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        let config = ADEumAgentConfiguration(appKey: "#EUM_APP_KEY#")
            // Configure the iOS Agent to report the metrics and screenshots to the SaaS EUM Server in EMEA
        config.collectorURL = "https://fra-col.eum-appdynamics.com"
        config.screenshotUrl = "https://fra-image.eum-appdynamics.com"
        ADEumInstrumentation.initWith(config)
            // other tasks
        return true
    }
#import <ADEUMInstrumentation/ADEumInstrumentation.h>
#import "AppDelegate.h"

// ...
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    let config = ADEumAgentConfiguration(appKey: <#EUM_APP_KEY#>)
    // Configure the iOS Agent to report the metrics and screenshots
    // to the SaaS EUM Server in APAC
    configcollectorURL = "https://syd-col.eum-appdynamics.com"
    config.screenshotUrl = "https://syd-image.eum-appdynamics.com"
    ADEumInstrumentation.initWith(config)
    // other tasks
    return true
}

4. (Optional) If your application is using another tool to report crashes, the iOS Agent will warn you with the following message:

Agent has detected a third party crash reporting tool. You may wish to disable AppDynamics Crash Reporting by setting the crashReportingEnabled configuration flag to NO

You are recommended to use only one crash reporting tool for better results. See Disable Crash Reporting for instructions on how to disable the iOS Agent’s crash reporting.

5. Save the file.

Apple Watch Extensions

Apps written for watchOS 1 contain a WatchKit extension that runs on the user’s iPhone, but watchOS 2 also supports a new architecture, where the WatchKit extension runs on the Apple Watch itself. AppDynamics supports the watchOS 1 architecture, but
not the new watchOS 2 architecture. Note that apps using the watchOS 1 architecture can run on both watchOS 1 and 2, so if your application is designed for watchOS 1, you can use AppDynamics on both versions of watchOS.

Because watchOS 1 apps are functionally launched in response to an interaction with the Watch UI, the SDK initialization code should be called at the point of that interaction in the iPhone app, which is usually not at the extension's AppDelegate.m call. The syntax remains the same.

Generate a dSYM File

To enable the agent to provide human-readable information in the crash snapshots that are produced if the application crashes, compile with the DWARF with dSYM file option to generate a debug symbols (dSYM) file for the application. For more details about why you would want to do this, see Get Human-Readable Crash Snapshots.

1. In Xcode, select your project in the Project Navigator.
2. In the target list, select the target that builds your application.
3. Select the Build Settings tab.
4. In the Build Options section, make sure that the Debug Information Format is set to DWARF with dSYM File.
5. Rebuild the Xcode project.

Monitor Crashes with the dSYM File

This step is optional but highly recommended if you plan to monitor crashes. AppDynamics needs the dSYM file for the application to produce human-readable stack traces for crash snapshots.

For instructions, see Upload the dSYM File.

Customize the Instrumentation (Optional)

The ADEUMInstrumentation class has additional methods to allow you to extend the kinds of data you can collect and aggregate using AppDynamics. There are five basic kinds of extensions that you can create:

- Custom timers: any arbitrary sequence of events within your code timed, even spanning multiple methods
- Custom metrics: any integer-based data you wish to collect
- User data: any string key/value pair you think might be useful
- Information points: how often a single method is invoked, and how long it takes to run
- Breadcrumbs: context for a crash

For more information, see Customize the iOS Instrumentation.

Configure the iOS Agent for On-Prem Deployments (Optional)

By default, the agent is configured to send its beacons to the EUM Cloud, which is an instance of the EUM Server running on AWS. If you wish to instrument your app in an environment that is using an on-prem version of the EUM Server, you need to modify the URL to which the agent sends its beacons. You do this using the AdeumAgentConfiguration object to set the Collector URL and the Screenshot Service URL to your on-premises EUM Server URL.

The iOS Agent knows which path to use to make calls to different services (Collector/Screenshot Service). For example, if the EUM Server URL is https://myEUMServerURL.com:7001, the iOS Agent will know to use https://myEUMServerURL.com:7001/eumcollector to make requests to the EUM Collector. By specifying the Collector URL, you will not be able to use the SaaS deployment of the EUM Cloud for the Screenshot Service.

To get the EUM Server URL:

1. Open the Administration Console.
2. From the left navigation bar, click Controller Settings.
3. In the search field, enter eum.beacon.host or eum.beacon.https.host if you're using HTTPS.
4. Copy the value for the configuration. This is your EUM Server URL.
The code examples below show how to set the Collector URL and Screenshot Service URL using Objective-C and Swift.

### Objective-C

```objective-c
ADEumAgentConfiguration *adeumAgentConfig =
    [[ADEumAgentConfiguration alloc]
    initWithAppKey:<#EUM_APP_KEY#>];

// Set the Collector URL and Screenshot Service URL to the EUM Server URL.
adeumAgentConfig.collectorURL = <#COLLECTOR_URL:PORT#>;
adeumAgentConfig.screenshotURL = adeumAgentConfig.collectorURL;

[ADEumInstrumentation initWithConfiguration:adeumAgentConfig];
```

### Swift 4.2

```swift
let configuration = ADEumAgentConfiguration(appKey: <#EUM_APP_KEY#>)

// Set the Collector URL and Screenshot Service URL to the EUM Server URL.
configuration.collectorURL = <#COLLECTOR_URL:PORT#>
configuration.screenshotUrl = configuration.collectorURL

ADEumInstrumentation.initWith(configuration)
```

---

**Enable HTTP to Send Beacons to On-Premises EUM Servers**

If you use an on-premises EUM Server and you wish to use HTTP to dispatch your beacons to the EUM Server, starting with iOS 9 you need to set a flag in your app's `info.plist` file to allow it to use the unsecured connection. By default, HTTPS is enforced in all iOS 9 applications by App Transport Security (ATS), and the iOS Agent complies with ATS when used with the EUM Cloud.
Customize the iOS Instrumentation

On this page:

- Collect Additional Types of Data
- Add a Crash Reporting Callback
- Report Errors and Exceptions
- Configure Application-Not-Responding (ANR) Detection
- Disable Crash Reporting
- Configure Hybrid Application Support
- Programmatically Control Sessions
- Start and End Session Frames
- Configure the Agent for Custom App Names
- Configure the Agent for Ignoring Some HTTP Requests
- Use the Agent with a Custom HTTP Library
- Capture User Interactions
- Configure and Take Screenshots
- Transform URLs for Network Requests
- Enable Logging and Set Logging Level

Related pages:

- Install the iOS SDK
- Instrument an iOS Application
- iOS SDK Documentation

Once you have instrumented your iOS application with the Mobile iOS SDK, you can also use the APIs exposed by the SDK to customize the data for your app that appears in the Controller UI.

Because the agent stores data about events in a local buffer before reporting the information, you are recommended to use the APIs with discretion.

Collect Additional Types of Data

You can use methods available in the ADEUMInstrumentation class to collect six additional types of data:

<table>
<thead>
<tr>
<th>Type of Data</th>
<th>Description</th>
<th>Specifications</th>
<th>Where Data is Displayed</th>
</tr>
</thead>
</table>
| Info points  | How often a method is invoked, and how long it takes to run. | Data is numeric  
Names must consist of alphanumeric characters and/or spaces | Metric Browser  
Custom Data  
Network Request Snapshots  
Mobile Sessions  
Network Request Analyze |
| Custom timers| Any arbitrary sequence of events within your code timed, even spanning multiple methods. | Data is numeric  
Metric names must consist of alphanumeric characters and/or spaces | Metric Browser  
Custom Data |
| Custom metrics| Any integer-based data you wish to collect. | Data is numeric  
Metric names must consist of alphanumeric characters and/or spaces | Metric Browser  
Custom Data |
| User data    | Any string key/value pair you think might be useful. | Data can be any type  
Metric names have no restrictions | Network Request Snapshots  
Mobile Sessions  
Network Request Analyze |
Breadcrumbs

The context for a crash.

- Data can be any data type
- Metric names have no restrictions

Network Request
Snapshots
Mobile Sessions
Network Request
Analyze

User interaction

Capture when users press buttons, click on lists, and select text.

- Data can be any data type
- Metric names have no restrictions

Network Request
Snapshots
Mobile Sessions
Network Request
Analyze

When you have set up additional data types, the Mobile Agent packages that data in a mobile beacon. Normally, the beacon is transmitted when the instrumented app sends an HTTP request or when the app is restarted following a crash, but if custom data has been collected and neither of those events has occurred for at least five minutes, the custom data is sent at that time.

**Info Points**

Information points allow you to track how your own code is running. You can see how often a method is invoked, and how long it takes to run, by using `beginCall` and `endCall`, something like the following:

**Objective-C**

```objective-c
- (void)myMethod
{
    id tracker = [ADEumInstrumentation beginCall:self selector:_cmd];

    // Implementation of method here ...

    [ADEumInstrumentation endCall:tracker];
}
```

**Swift 4.2**

```swift
@objc (void) myMethod
{
    let tracker = try! ADEumInstrumentation.beginCall(self, selector: _cmd)

    // Implementation of method here ...

    try! ADEumInstrumentation.endCall(tracker)
}
```
func myMethod() {
    let tracker = ADEumInstrumentation.beginCall(self, selector: #function)
    // Implementation of method here ...
    ADEumInstrumentation.endCall(tracker)
}

If an exception is thrown, it is also reported. This information appears in the **Custom Data** view in the Controller UI.

**Custom Timers**

Custom timers allow you to time any arbitrary sequence of events within your code, even spanning multiple methods, by using `startTimer` and `stopTimer`. For example, to track the time a user spends viewing a screen, the instrumentation could look like this:

### Objective-C

```objective-c
- (void)viewDidAppear:(BOOL)animated {
    [super viewDidAppear:animated];
    [ADEumInstrumentation startTimerWithName:@"View Lifetime"];
}

- (void)viewDidDisappear:(BOOL)animated {
    [super viewDidDisappear:animated];
    [ADEumInstrumentation stopTimerWithName:@"View Lifetime"];
}
```

### Swift 4.2

```swift
func myMethod() {
    let tracker = ADEumInstrumentation.beginCall(self, selector: #function)
    // Implementation of method here ...
    ADEumInstrumentation.endCall(tracker)
}
```
func viewDidAppear(_ animated: Bool) {
    super.viewDidAppear(animated)
    ADEumInstrumentation.startTimer(withName: "View Lifetime")
}
func viewDidDisappear(_ animated: Bool) {
    super.viewDidDisappear(animated)
    ADEumInstrumentation.stopTimer(withName: "View Lifetime")
}

This information appears in the **Custom Data** view of the Controller UI.

Calling `startTimerWithName` again with the same name value resets a named timer.

**Custom Metrics**

Any integer-based data can be passed to the agent. The first parameter to the `reportMetricWithName` call is the name you want the metric to appear under in the Controller UI. The metric name should only contain alphanumeric characters and spaces. Illegal characters are replaced by their ASCII hex value.

Reporting a metric called “My custom metric”, for example, would look something like this:

```objective-c
[ADEumInstrumentation reportMetricWithName:@"My custom metric"
 value:<#VALUE HERE#>];
```

This information appears in the **Custom Data** view of the Controller UI.

**User Data**

You can set any string key/value pair you think might be useful. The first parameter to the `setUserData` call is the key you want to use, which must be unique across your application. The second is the value that you want to be assigned to the key.

For example:

Objective-C
Swift 4.2

```swift
func onUserLogged(in userid: String?) {
    ADEumInstrumentation.setUserData("User ID", value: userid)
}
```

This information is available in Network Request Analyze and is added to any crash snapshots that may be taken. Keys and values are limited to 2048 characters each.

You can also set user data with values of other types (Long, Boolean, Double, Date) using the following methods:

- `setUserDataLong:value`
- `setUserDataBoolean:value:
- `setUserDataDouble:value:
- `setUserDataDate:value:

**Breadcrumbs**

Breadcrumbs allow you to situate a crash in the context of your user's experience. Set a breadcrumb when something interesting happens. If your application crashes at some point in the future, the breadcrumb will be displayed along with the crash report.

There are two ways of leaving breadcrumbs:

- Crash Reports Only
- Modal

Using this method means that breadcrumbs are reported in crash reports only.

```swift
+ (void)leaveBreadcrumb:(NSString *)breadcrumb
```

Using this method lets you fine tune where the breadcrumbs are reported, either only in crash reports or in crash reports and sessions.
+ (void)leaveBreadcrumb:(NSString *)breadcrumb
  mode:(ADEumBreadcrumbVisibility)mode

Where mode is either:
  • ADEumBreadcrumbVisibilityCrashesOnly
  • ADEumBreadcrumbVisibilityCrashesAndSessions

If the breadcrumb is over 2048 characters, it is truncated. If it is empty or nil, no breadcrumb is recorded. Each crash report displays the most recent 99 breadcrumbs.

Add a Crash Reporting Callback

You may want to make crash report information that Mobile RUM collects available to other parts of your code, for example, to Google Analytics, if you are using it. To enable you to pass on summary crash information, you can set up a crash report runtime callback. To get a callback when the iOS Agent detects and then reports a crash, you need to implement the following protocol in your code:

@protocol ADEumCrashReportCallback <NSObject>
- (void)onCrashesReported:(NSArray<ADEumCrashReportSummary *> *)crashReportSummaries;
@end

This callback is invoked on your app's UI thread, so any significant work should be done on a separate work thread.

Each ADEumCrashReportSummary passed in has the following properties:
@interface ADEumCrashReportSummary : NSObject
/** Uniquely defines the crash, can be used as key to find full crash
report. */
@property (nonatomic, readonly) NSString *crashId;
/** The exception name, may be `nil` if no `NSException` occured. */
@property (nonatomic, readonly) NSString * ADEUM_NULLABLE exceptionName;
/** The exception reason, may be `nil` if no `NSException` occured. */
@property (nonatomic, readonly) NSString * ADEUM_NULLABLE
exceptionReason;
/** The Mach exception signal name */
@property (nonatomic, readonly) NSString *signalName;
/** The Mach exception signal code */
@property (nonatomic, readonly) NSString *signalCode;
@end

If you are sending the information to another analytics tool, such as Google Analytics, it is best to include all five properties:
exceptionName and exceptionReason are optional and useful for a quick identification of what the crash is. These are only
present if the crash cause occurred within an exception reporting runtime, such as Objective-C.
signalName and signalCode are useful for quick identification of the crash. These are from the system and are independent
of the runtime.
For additional information, crashId can be used to look up the crash in the AppDynamics Controller UI.
For example, to print the crash information to iOS's logger, you could implement an ADEumCrashReportCallback class like this:

// assumes the containing object has "adopted" the protocol
- (void)onCrashesReported:(NSArray<ADEumCrashReportSummary *> *)summaries
{
for (ADEumCrashReportSummary *summary in summaries) {
NSLog(@"Crash ID: %@", summary.crashId);
NSLog(@"Signal: %@ (%@)", summary.signalName,
summary.signalCode);
NSLog(@"Exception Name:\n%@", summary.exceptionName);
NSLog(@"Exception Reason:\n%@", summary.exceptionReason);
}
}

You set the object that implements the ADEumCrashReportCallback protocol during agent configuration:

Copyright © AppDynamics 2012-2019

Page 452


ADEumAgentConfiguration *config = [ADEumAgentConfiguration new];
config.crashReportCallback = myCrashReportCallback;

Your callback is invoked, on the main/UI thread, if a crash from a previous run is detected and collected. See the latest iOS SDK documentation for more information.

Report Errors and Exceptions

You can report exceptions using the method `reportError` from the `ADEumInstrumentation` class. Reported exceptions will appear in session details.

The method can have the following two signatures:

<table>
<thead>
<tr>
<th>Objective-C Function Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(void)reportError:(NSError *)error withSeverity:(ADEumErrorSeverityLevel)severity;</td>
<td>Use this signature to report errors, set the severity level of the issue, and send the stack trace.</td>
</tr>
<tr>
<td>(void)reportError:(NSError *)error withSeverity:(ADEumErrorSeverityLevel)severity andStackTrace:(BOOL)stacktrace;</td>
<td>Use this signature to report errors, set the severity level of the issue, and explicitly specify whether the stack trace should be included.</td>
</tr>
<tr>
<td></td>
<td>If you include the stack trace with the reported error by setting <code>stacktrace</code> to <code>YES</code>, you can view the stack trace in the Code Issues Details dialog.</td>
</tr>
<tr>
<td></td>
<td>To report the error without the stack trace, set <code>stacktrace</code> to <code>NO</code>.</td>
</tr>
</tbody>
</table>

**Severity Levels**

You can also set one of the following severity levels for an issue. With the severity level, you can filter errors in the Code Issues Dashboard or Code Issues Analyze.

- `ADEumErrorSeverityLevelInfo`
- `ADEumErrorSeverityLevelWarning`
- `ADEumErrorSeverityLevelCritical`

**Examples of Reporting Errors**

The example below uses the API to report possible exceptions and set the severity level to `ADEumErrorSeverityLevelCritical` for a failed attempt to perform a file operation.

Objective-C
NSError *err = nil;
[[NSFileManager defaultManager] contentsOfDirectoryAtPath:@"pathToFile" error:&err];
if (err) {
    [ADEumInstrumentation reportError:err
     withSeverity:ADEumErrorSeverityLevelCritical, andStackTrace: NO];
} else {
    ...
}

Swift 4.2

var err: Error? = nil
try? FileManager.default.contentsOfDirectory(atPath: "pathToFile")
if err != nil {
    ADEumInstrumentation.reportError(err, withSeverity: ADEumErrorSeverityLevelCritical, andStackTrace: false)
} else {
    ...
}

You can also create and report custom errors with the following. Note that because reportError is not passed the argument andStackTrace, by default, the stack trace is automatically included with the error.

Objective-C
Swift 4.2

```swift
var domain = "com.YourCompany.AddUsers.ErrorDomain"
var desc = NSLocalizedString("Unable to add user.", comment: "")
var userInfo = [NSLocalizedDescriptionKey: desc]
var error = NSError(domain: domain, code: -101, userInfo: userInfo)
ADEumInstrumentation.reportError(error, withSeverity:
    ADEumErrorSeverityLevelWarning)
```

**Configure Application-Not-Responding (ANR) Detection**

By default, the iOS Agent does not detect ANR issues, and when ANR detection is enabled, the ANR issues are reported without stack traces. You must manually enable ANR detection and set a flag to include stack traces through the iOS Agent configuration. For more information about ANR monitoring, see [Code Issues](#). To specify thresholds for ANR issues, see [Configure Application Not Responding Thresholds](#).

**Enable ANR Detection**

You enable the detection of ANR issues by configuring the instrumentation with the `anrDetectionEnabled` property as shown below.

**Objective-C**

```objective-c
NSString *domain = @"com.YourCompany.AddUsers.ErrorDomain";
NSString *desc = NSLocalizedString(@"Unable to add user.", @"");
NSDictionary *userInfo = @[ NSLocalizedDescriptionKey : desc ];
NSError *error = [NSError errorWithDomain:domain
    code:-101
    userInfo:userInfo];
ADEumInstrumentation reportError:error withSeverity:
    ADEumErrorSeverityLevelWarning];
```
ADEumAgentConfiguration *adeumAgentConfig =
[[ADEumAgentConfiguration alloc] initWithAppKey:
<#EUM_APP_KEY#>];
// Enable ANR detection
adeumAgentConfig.anrDetectionEnabled = YES;
[ADEumInstrumentation initWithConfiguration:adeumAgentConfig];

Swift 4.2

let config = ADEumAgentConfiguration(appKey: <#EUM_APP_KEY#>);
// Enable ANR detection
config.anrDetectionEnabled = true;
ADEumInstrumentation.initWith(config);

Report Stack Traces with ANRs

In addition to enabling ANR detection, you set the property anrStackTraceEnabled to YES (Objective-C) or true (Swift) to report stack traces with the ANRs.

Objective-C

ADEumAgentConfiguration *adeumAgentConfig =
[[ADEumAgentConfiguration alloc] initWithAppKey:
<#EUM_APP_KEY#>];
// Enable ANR detection
adeumAgentConfig.anrDetectionEnabled = YES;
// Set the flag to include stack traces with ANRs
adeumAgentConfig.anrStackTraceEnabled = YES;
[ADEumInstrumentation initWithConfiguration:adeumAgentConfig];
Swift 4.2

```swift
let config = ADEumAgentConfiguration(appKey: "#EUM_APP_KEY#")
// Enable ANR detection
config.anrDetectionEnabled = true
// Set the flag to include stack traces with ANRs
config.anrStackTraceEnabled = true
ADEumInstrumentation.initWith(config)
```

Disable Crash Reporting

Crash reporting is enabled by default, but you can manually disable crash reporting through the instrumentation configuration. If you are using other crash reporting tools, you might disable crash reporting to minimize conflicts and optimize the crash report results.

You can disable crash reporting by configuring the instrumentation with the `crashReportingEnabled` property as shown in the following code example.

Objective-C

```objective-c
ADEumAgentConfiguration *config = [[ADEumAgentConfiguration alloc] initWithAppKey:appKey];
config.crashReportingEnabled = No
[ADEumInstrumentation initWithConfiguration:config];
```
Configure Hybrid Application Support

By default, the iOS Agent instruments iOS WKWebViews, but does not collect and report Ajax calls. See Hybrid Application Support for an overview and an explanation of how it works.

You can configure the static or runtime configuration to disable hybrid application support or modify its behavior. The sections below show you how to change the defaults for hybrid support through either runtime or static configuration.

Runtime Configuration for Hybrid Application Support

The code example below disables the injection of the JavaScript Agent. By disabling the injection, the WKWebViews in your application will not be instrumented and Ajax calls will not be reported.

```swift
ADEumAgentConfiguration *adeumAgentConfig = [[ADEumAgentConfiguration alloc] initWithAppKey: <#EUM_APP_KEY#>];
// Disable the JavaScript Agent Injection
adeumAgentConfig.jsAgentEnabled = NO;
[ADEumInstrumentation initWithConfiguration:adeumAgentConfig];
```

The JavaScript Agent injection is enabled by default. To also enable the collection and reporting of Ajax calls:

```swift
ADEumAgentConfiguration *adeumAgentConfig = [[ADEumAgentConfiguration alloc] initWithAppKey: <#EUM_APP_KEY#>];
// Enable the collection and reporting of Ajax calls
adeumAgentConfig.jsAgentAjaxEnabled = YES;
[ADEumInstrumentation initWithConfiguration:adeumAgentConfig];
```

Static Configuration for Hybrid Application Support

You should use static configuration for the following reasons:

- force the instrumentation of WKWebViews and/or Ajax calls (override the runtime configuration)
- disable hybrid support and override the runtime configuration
- set the URL to your self-hosted JavaScript Extension file

The table below describes the supported properties and provides the default value for the info.plist file.
<table>
<thead>
<tr>
<th>Property</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serverJsAgentEnabled</td>
<td>true</td>
<td>If the client receives a <code>false</code> for this flag, then the JavaScript Agent will be disabled. Thus, the WKWebViews and Ajax requests will not be monitored. The injection occurs during the creation of a new WKWebView. So, if a WKWebView is created when this flag is set to <code>false</code>, that particular WKWebView won’t be instrumented even if the flag is subsequently set to <code>true</code>.</td>
</tr>
<tr>
<td>ForceWebviewInstrumentation</td>
<td>false</td>
<td>When set to <code>true</code>, the iOS Agent will inject the JavaScript Agent into the WKWebViews regardless of the runtime configuration.</td>
</tr>
<tr>
<td>ForceAjaxInstrumentation</td>
<td>true</td>
<td>When set to <code>true</code>, Ajax operations will always be collected and reported regardless of the runtime configuration.</td>
</tr>
<tr>
<td>ADRUMExtUrlHttp</td>
<td><a href="http://cdn.appdynamics.com">http://cdn.appdynamics.com</a></td>
<td>The JavaScript Agent consists of two components: the base JavaScript Agent and the JavaScript Agent extension. The base JavaScript Agent is built into the Mobile Agent binary and injected according to the rules above.</td>
</tr>
<tr>
<td>ADRUMExtUrlHttps</td>
<td><a href="https://cdn.appdynamics.com">https://cdn.appdynamics.com</a></td>
<td>After initialization, the JavaScript Agent fetches the JavaScript Agent extension from the URLs specified by these properties.</td>
</tr>
</tbody>
</table>

**Example Configuration**

The example `info.plist` below forces the instrumentation of WKWebViews (overriding the runtime configuration), but does not force the collection and reporting of Ajax requests. The configuration also sets the URL where the JavaScript Extension file is obtained.

```xml
<plist>
 <dict>
  ...
  <key>ADEUM_Settings</key>
  <dict>
   <key>ForceWebviewInstrumentation</key>
   <true/>
   <key>ForceAjaxInstrumentation</key>
   <false/>
   <key>ADRUMExtUrlHttp</key>
   <string>http://<your-domain>/adrum.cdn</string>
   <key>ADRUMExtUrlHttps</key>
   <string>https://<your-domain>/adrum.cdn</string>
  </dict>
  ...
 </dict>
</plist>
```

**Programmatically Control Sessions**

By default, a mobile session ends after a period of user inactivity. For example, when a user opens your application, the session begins and only ends after the user stops using the app for a set period of time. When the user begins to use the application again, a new session begins.

Instead of having a period of inactivity to define the duration of a session, however, you can use the following API to programmatically
control when sessions begin and end:

```objective-c
- (void)startNextSession
```

When you call the method `startNextSession` from the `ADEumInstrumentation` class, the current session ends and a new session begins. The API enables you to define and frame your sessions so that they align more closely with business goals and expected user flows. For example, you could use the API to define a session that tracks a purchase of a product or registers a new user.

Excessive use of this API will cause sessions to be throttled (excessive use is >10 calls per minute per iOS Agent, but is subject to change). When not using the API, sessions will fall back to the default of ending after a period of user inactivity.

**Example of a Programmatically Controlled Session**

In the example below, the current session ends and a new one begins when the check out is made.
-(void) checkout {
    AppDelegate *appDelegate = (AppDelegate *) [[UIApplication sharedApplication] delegate];
    NSString *checkoutUrl = [appDelegate.url stringByAppendingString:@"rest/cart/co/"];
    NSURL *url = [NSURL URLWithString:checkoutUrl];
    NSMutableURLRequest *request = [[NSMutableURLRequest alloc] initWithURL:url cachePolicy:NSURLRequestUseProtocolCachePolicy timeoutInterval:60.0];
    NSURLResponse *response = nil;
    NSError *error = nil;
    NSData *body = [NSURLConnection sendSynchronousRequest:request returningResponse:&response error:&error];
    const char *responseBytes = [body bytes];
    if (responseBytes == nil)
        checkoutResponse = [NSString stringWithUTF8String:"Could not connect to the server"];
    else {
        checkoutResponse = [NSString stringWithUTF8String:responseBytes];
        [ADEumInstrumentation startNextSession];
    }
}

Swift 4.2
Start and End Session Frames

You can use the SessionFrame API to create session frames that will appear in the session activity. Session frames provide context for what the user is doing during a session. With the API, you can improve the names of user screens and chronicle user flows within a business context.

Use Cases

The following are common use cases for the SessionFrame API:

- One ViewController performs multiple functions and you want more granular tracking of the individual functions.
- A user flow spans multiple ViewController or user interactions. For example, you could use the API to create the session frames “Login”, “Product Selection”, and “Purchase” to chronicle the user flow for purchases.
- You want to capture dynamic information based on user interactions to name session frames, such as an order ID.

SessionFrame API

The table below lists the three methods you can use with session frames. In short, you start a session frame with `startSessionFram...
and then use the returned `ADEumSessionFrame` object to rename and end the session frame.

### Objective-C

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ADEumInstrumentation</code></td>
<td>+ (ADEumSessionFrame *)startSessionFrame:(NSString *)name</td>
<td>Use this to start and name your session frame. Naming session frames enable you to easily identify and track the frames in the Sessions Dialog.</td>
</tr>
<tr>
<td><code>ADEumSessionFrame</code></td>
<td>- (void)updateName:(NSString *)name</td>
<td>Rename the session frame name. You call this method from the object returned from <code>startSessionFrame</code>.</td>
</tr>
<tr>
<td><code>ADEumSessionFrame</code></td>
<td>- (void)end</td>
<td>End the session frame. You call this method from the object returned from <code>startSessionFrame</code>.</td>
</tr>
</tbody>
</table>

### Swift 4.2

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ADEumInstrumentation</code></td>
<td>startSessionFrame(_ name: String?) -&gt; ADEumSessionFrame</td>
<td>Use this to start and name your session frame. Naming session frames enable you to easily identify and track the frames in the Sessions Dialog.</td>
</tr>
<tr>
<td><code>ADEumSessionFrame</code></td>
<td>updateName(_ name: String?)</td>
<td>Rename the session frame name. You call this method from the object returned from <code>startSessionFrame</code>.</td>
</tr>
<tr>
<td><code>ADEumSessionFrame</code></td>
<td>end()</td>
<td>End the session frame. Returned from <code>startSessionFrame</code>.</td>
</tr>
</tbody>
</table>
Session Frame Example

In the following example, the `SessionFrame` API is used to track user activity during the checkout process.

Objective-C
#import "ADEumSessionFrame.h"
...
@property (nonatomic, strong) ADEumSessionFrame *checkoutSessionFrame;

-(IBAction)checkoutCartButtonClicked:(id)sender {
    // The user starting to check out starts when the user
    // clicks the checkout button
    // this may be after they have updated quantities of items
    // in their cart, etc.
    checkoutSessionFrame = [ADEumInstrumentation
    startSessionFrame:@"Checkout"];
}

-(IBAction)confirmOrderButtonClicked:(id)sender {
    // Once they have confirmed payment info and shipping
    // information, and they
    // are clicking the "Confirm" button to start the backend
    // process of checking out
    // we may know more information about the order itself, such
    // as an Order ID.
    NSString *newSessionName = [NSString
    stringWithFormat:@"Checkout: Order ID %",orderId];
    [checkoutSessionFrame updateName:newSessionName];
}

-(void)processOrderCompleted {
    // Once the order is processed, the user is done "checking
    // out" so we end
    // the session frame
    [checkoutSessionFrame end];
    checkoutSessionFrame = nil;
}

-(void)checkoutCancelled {
    // If they cancel or go back, you'll want to end the session
    // frame also, or else
    // it will be left open and appear to have never ended.
    [checkoutSessionFrame end];
    checkoutSessionFrame = nil;
}
import ADEumSessionFrame
...

var checkoutSessionFrame: ADEumSessionFrame?
@IBAction func checkoutCartButtonClicked(_ sender: UIButton) {
    // The check out starts when the user clicks the checkout button.
    // This may be after they have updated quantities of items in their cart, etc.
    checkoutSessionFrame = ADEumInstrumentation.startSessionFrame("Checkout")
}

@IBAction func confirmOrderButtonClicked(_ sender: UIButton) {
    // Once users have confirmed payment info and shipping information, and they
    // are clicking the "Confirm" button to start the backend process of checking out,
    // we may know more information about the order itself, such as an order ID.
    let newSessionName = "Checkout: Order ID \(orderId)"
    checkoutSessionFrame.updateName(newSessionName)
}

func processOrderCompleted() {
    // Once the order is processed, the user is done "checking out", so we end the session frame.
    checkoutSessionFrame.end()
    checkoutSessionFrame = nil
}

func checkoutCancelled() {
    // If they cancel or go back, you'll want to end the session frame also, or else it will be
    // left open and appear to have never ended.
    checkoutSessionFrame.end()
    checkoutSessionFrame = nil
}

Configure the Agent for Custom App Names

By default, AppDynamics automatically detects the name of your application. The application name is a string form of the bundle ID.
Thus, if the bundle ID is `com.example.appdynamics.HelloWorld`, the application name will be "com.example.appdynamics.HelloWorld".

There may be cases, however, where you deploy essentially the same app binary with different bundle IDs to various regional app stores. To make sure all the data belonging to one app is collected and displayed together, despite varying bundle IDs, you can set a common name by giving the apps a custom name. To do this, set the application name property in the `ADEumAgentConfiguration` instance that you use to set up `ADEumInstrumentation`. See the latest iOS SDK documentation for more information.

```objc
@property (nonatomic, strong) NSString *applicationName;
```

### Configure the Agent for Ignoring Some HTTP Requests

In some cases, HTTP requests using NSURL are used for internal purposes in an application and do not represent actual network requests. Metrics created based on these requests are not normally useful in tracking down issues, so preventing data on them from being collected can be useful. To ignore specific NSURL requests, set the excluded URL patterns property in the `ADEumAgentConfiguration` instance that you use to set up `ADEumInstrumentation`. Use the simplest regex possible. See the latest iOS SDK documentation for more information.

```objc
@property (nonatomic, strong) NSSet *excludedUrlPatterns;
```

### Use the Agent with a Custom HTTP Library

The iOS Agent automatically detects network requests when the underlying implementation is handled by either by the `NSURLConnection` or the `NSURLSession` classes. This covers the great majority of iOS network requests. In some cases, however, mobile applications use custom HTTP libraries.

- To have the iOS Agent detect requests from a custom library, add request tracking code to your application manually, using the `ADEumHTTPRequestTracker` class.
- To set headers to allow correlation with server-side processing, use the `ADEumServerCorrelationHeaders` class.
- To configure the agent to use your custom library to deliver its beacons over HTTP, use the `ADEumCollectorChannel` protocol and the `ADEumAgentConfiguration` class.

### Add Request Tracking

To add request tracking manually, you tell the agent when the request begins and when it ends. You also set properties to tell the agent the status of the response.

#### Start and complete tracking a request

To begin tracking an HTTP request, call the following method immediately before sending the request.

```objc
You must initialize the agent using one of the ADEumInstrumentation's initWithKey methods before using this method.
```
@interface ADEumHTTPRequestTracker : NSObject
...
+ (ADEumHTTPRequestTracker *)requestTrackerWithURL:(NSURL *)url;

Where `url` is the URL being requested. This parameter must not be `nil`.

To complete tracking an HTTP request, immediately after receiving a response or an error, set the appropriate properties on the tracker object and call the following method to report the outcome of the request back to the agent. You should not continue to use this object after calling this method. To track another request, call `requestTrackerWithURL` again.

- (void)reportDone;

Properties to be set

The following properties should be set on the `requestTrackerWithURL` object to describe to the agent the results of the call.

@property (copy, nonatomic) NSError *error;

Indicates the failure to receive a response, if this occurred. If the request was successful, this should be `nil`.

@property (copy, nonatomic) NSNumber *statusCode;

Reports the HTTP status code of the response, if one was received.
- If a response was received, this should be an integer.
- If an error occurred and a response was not received, this should be `nil`.

@property (copy, nonatomic) NSDictionary *allHeaderFields;

Provides a dictionary representing the keys and values from the server’s response header. The format of this dictionary should be identical to the `allHTTPHeaderFields` property of `NSURLRequest`. The dictionary elements consist of key/value pairs, where the key is the header key name and the value is the header value.
If an error occurred and a response was not received, this should be `nil`.

Example:

Given a request snippet like this:

```objective-c
- (NSData *)sendRequest:(NSURL *)url error:(NSError **)error {
    // implementation omitted
    NSData *result = nil;
    if (errorOccurred) {
        *error = theError;
    } else {
        result = responseBody;
    }
    return result;
}
```

Adding the tracker could look something like this:

```objective-c
- (NSData *)sendRequest:(NSURL *)url error:(NSError **)error {
    ADEumHTTPRequestTracker *tracker = [ADEumHTTPRequestTracker requestTrackerWithURL:url];
    // implementation omitted
    NSData *result = nil;
    if (errorOccurred) {
        *error = theError;
        tracker.error = theError;
    } else {
        tracker.statusCode = theStatusCode;
        tracker.allHeaderFields = theResponseHeaders;
        result = responseBody;
    }
    [tracker reportDone];
    return result;
}
```

Enable Server-Side Correlation

To enable correlation between your request and server-side processing, add specific headers to outgoing requests that the server-side agent can detect and return the headers obtained from the server-side agent in the response to make them available to the iOS Agent.

This is done automatically for standard HTTP libraries.
You must:

1. Call the `generate` method and set the generated headers before sending a request to the backend.
2. Report back the response headers, using the `allHeaderFields` property shown above.

**Configure Agent to Use Custom HTTP Library**

The iOS Agent uses HTTP to deliver its beacons. To have the agent use your custom HTTP library for this purpose, do the following.

1. Implement a class that conforms to this protocol:

```objc
/**
 * Protocol for customizing the connection between the agent SDK and the collector.
 */
@protocol ADEumCollectorChannel <NSObject>
/**
 * Sends a request synchronously and returns the response received, or an error.
 * The semantics of this method are exactly equivalent to NSURLConnection's
 * @param request The URL request to load.
 * @param response Out parameter for the URL response returned by the server.
 * @param error Out parameter used if an error occurs while processing the request. May be NULL.
 */
- (NSData *)sendSynchronousRequest:(NSURLRequest *)request
returningResponse:(NSURLResponse **)response
error:(NSError **)error;
@end
```

2. Set the `collectorChannel` property in `ADEumAgentConfiguration` before initializing `ADEumInstrumentation`, passing in an instance of your class that implements `ADEumCollectorChannel`. See the latest iOS SDK documentation for more information.
Capture User Interactions

You can enable the iOS Agent to track certain UI events triggered by user interactions. Once user interactions have been captured, you can sort sessions by UI event and view UI events in the timeline of the session waterfall.

You can capture when users do one or all of the following:

- press buttons
- select table cells
- select text fields
- select text views

Security and Privacy Concerns

The interaction capture mode is disabled by default for security and privacy reasons as user interactions may contain sensitive information. Moreover, this potential security and privacy issue may be compounded if you enable both the capturing of UI interactions and screenshots.

Enable User Interaction Capture Mode

To enable user interaction capture mode, you assign the capture mode to the `interactionCaptureMode` of the `ADEumAgentConfiguration` object. The instrumentation code example below configures the iOS Agent to capture all the supported types of user interactions.

```objective-c
ADEumAgentConfiguration *config = [[ADEumAgentConfiguration alloc] initWithAppKey: <#EUM_APP_KEY#>];
config.interactionCaptureMode = ADEumInteractionCaptureModeAll;
[ADEumInstrumentation initWithConfiguration:config];
```

You can also configure the iOS Agent to only capture one type of user interaction:

```objective-c
ADEumAgentConfiguration *config = [[ADEumAgentConfiguration alloc] initWithAppKey: <#EUM_APP_KEY#>];
config.interactionCaptureMode = ADEumInteractionCaptureModeButtonPressed;
[ADEumInstrumentation initWithConfiguration:config];
```

Configure and Take Screenshots
Mobile screenshots are enabled by default. You can configure the Controller UI to automatically take screenshots or use the iOS SDK to manually take a screenshot as shown below:

**Objective-C**

```
[ADEumInstrumentation takeScreenshot];
```

**Swift 4.2**

```
ADEumInstrumentation.takeScreenshot()
```

**Disable Screenshots**

You can disable screenshots from the Controller UI or with the iOS SDK. To disable screenshots with the iOS SDK, set the property `screenshotsEnabled` of the `ADEumAgentConfiguration` object to `NO` for Objective-C and `false` for Swift as shown below.

**Objective-C**

```
ADEumAgentConfiguration *config = [[ADEumAgentConfiguration alloc] initWithAppKey: <#EUM_APP_KEY#>];
config.screenshotsEnabled = NO;
[ADEumInstrumentation initWithConfiguration:config];
```

**Swift 4.2**

```
```
let config = ADEumAgentConfiguration(appKey: "<#EUM_APP_KEY#>");
config.screenshotsEnabled = false;
ADEumInstrumentation.initWith(config);

**Block/Unblock Screenshots**

You can also use the iOS SDK to block screenshots from being taken during the execution of a code block. This just temporarily blocks screenshots from being taken until you unblock screenshots. This enables you to stop taking screenshots in situations where users are entering personal data, such as on login and account screens.

The `ADEumInstrumentation` class provides the methods `blockScreenshots` and `unblockScreenshots` to block and unblock screenshots. If screenshots are disabled through the property `screenshotsEnabled` of the `ADEumAgentConfiguration` object or through the Controller UI, these methods have no effect. You can also call `screenshotsBlocked` to check if screenshots are being blocked.

**Transform URLs for Network Requests**

When your application makes network requests, you may not want to report URLs containing sensitive information to the EUM Server. You can instead transform the network request URL before reporting it or ignore it altogether.

To do so:

1. Implement a network request callback that modifies or ignores specific URLs.
2. Register the network request callback in the initialization code.

**Implement the Network Request Callback**

The callback that modifies or ignore specific URLs is an implementation of the protocol below. The callback method `networkRequestCallback` is synchronous, so it is recommended that you return from the function quickly.

```objective-c
- (BOOL)networkRequestCallback:(ADEumHTTPRequestTracker *)networkRequest
```

**Transforming URLs**

The `networkRequestCallback` method, in general, should follow the steps below to transform URLs:

1. Identify specific URLs using techniques such as regex or pattern matching.
2. Modify the `url` property of the `ADEumHTTPRequestTracker` object. (Modifying other properties of the `ADEumHTTPRequestTracker` object will be ignored.)
3. Assign a valid URL to the `url` property.
4. Return `YES` (Objective-C) or `true` (Swift).

The first step is optional as you could choose to transform the URLs of all network requests.
In general, however, you would want to identify and transform URLs that contain sensitive information as implied in the example below.
Ignoring URLs

If the `networkRequestCallback` method returns `false`, the beacon is dropped. The general process for ignoring beacons is as follows:

1. Identify specific URLs using techniques such as regex or pattern matching.
2. Return `false`.

You could theoretically ignore all network requests by having the callback `networkRequestCallback` always return `NO` (Objective-C) or `false` (Swift):
In general, though, you would identify network requests that you didn’t want to monitor and return **NO** (Objective-C) or **false** (Swift) to ignore the network request as implied by this example.
- (BOOL)networkRequestCallback:(ADEumHTTPRequestTracker *)networkRequest
{
    NSString *urlString = networkRequest.url.absoluteString;
    BOOL returnBeacon = YES;
    if (!([urlString rangeOfString:@"avatar"].location ==
            NSNotFound)) {
        returnBeacon = NO;
    }
    return returnBeacon;
}

Swift 4.2

func networkRequestCallback(_ networkRequest: ADEumHTTPRequestTracker?) -> Bool {
    let urlString = networkRequest?.url.absoluteString
    var returnBeacon = true
    if !(Int((urlString as NSString?)?.range(of:
            "avatar").location ?? 0) == NSNotFound) {
        returnBeacon = false
    }
    return returnBeacon
}

Register the Callback

After implementing the callback, you register the object implementing the protocol method in the initialization code as shown below. When the iOS Agent is ready to create a network request beacon, it will first call the callback with an ADEumHTTPRequestTracker object.
Enable Logging and Set Logging Level

You use the method `loggingLevel` to enable and set the logging level. You can set logging to one of the following levels:

- `ADEumLoggingLevelOff`
- `ADEumLoggingLevelAll`
- `ADEumLoggingLevelVerbose`
- `ADEumLoggingLevelDebug`
- `ADEumLoggingLevelInfo`
- `ADEumLoggingLevelWarn`
- `ADEumLoggingLevelError`

Use verbose, all, and debug levels of logging only for troubleshooting and be sure to turn off for production.

Examples:

Objective-C

```objc
ADEumAgentConfiguration *config = [[ADEumAgentConfiguration alloc] initWithAppKey: <#EUM_APP_KEY#>];
config.networkRequestCallback = self;
[ADEumInstrumentation initWithConfiguration:config];
```
-(BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // appKey should be assigned your EUM app key
    ADEumAgentConfiguration *config = [[ADEumAgentConfiguration
alloc] initWithAppKey: <#EUM_APP_KEY#>];
    config.loggingLevel = ADEumLoggingLevelAll;
    [ADEumInstrumentation initWithConfiguration:config];
    ...
}

Swift 4.2

func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool {
{
    // appKey should be assigned your EUM app key
    let config = ADEumAgentConfiguration(appKey:
<#EUM_APP_KEY#>)
    config.loggingLevel = .all
    ADEumInstrumentation.initWithConfiguration(config)
    ...
    return true
}

iOS SDK Documentation

See the latest iOS SDK documentation or the previous versions listed below:

- https://docs.appdynamics.com/javadocs/ios-sdk/4.5/4.5.0/html/
- https://docs.appdynamics.com/javadocs/ios-sdk/4.5/4.5.5/html/
- https://docs.appdynamics.com/javadocs/ios-sdk/50/50.0/html/
After 4.5.6, the iOS Agent started using a version number different from that of the Controller and the other AppDynamics platform components. See Mobile Agent Version and Deployment Support Matrix for the minimum version of the Controller and the EUM Server required for complete support of all the iOS Agent features.
Upload the dSYM File

On this page:
- Enable the dSYM File
- Upload the dSYM File to AppDynamics Automatically with Each Build
- Upload the dSYM File to AppDynamics Manually

Related pages:
- Manage the dSYM Files with Bitcode Enabled
- Troubleshoot the iOS Instrumentation

AppDynamics needs the dSYM file for the application to produce human-readable stack traces for crash snapshots. For details about why you should do this, see Get Human-Readable Crash Snapshots.

If you update the application, you need to provide the new dSYM file for the new application version. The dSYM file contains a UUID that links it to a specific Xcode build, so AppDynamics can unambiguously match the correct dSYM file with an incoming crash report with no additional information.

To upload the dSYM file, you do the following:
- enable the dSYM file
- set up your environment to upload the file automatically each time you build or upload the file manually

Enable the dSYM File

To enable the agent to provide human-readable information in the crash snapshots that are produced if the application crashes, compile with the `DWARF with dSYM` file option to create a debug symbols file for the application.

To enable dSYM

1. In Xcode, select your project in the Project Navigator.
2. In the target list, select the target that builds your application.
3. Select the Build Settings tab.
4. In the Build Options section, make sure that the Debugging Information Format is set to `DWARF with dSYM File`.

Upload the dSYM File to AppDynamics Automatically with Each Build

Automating the upload of your dSYM file reduces the number of manual steps required for each build and ensures that all builds have appropriate dSYM files available for AppDynamics to use.

1. In Xcode, select your project from the Project Navigator.
2. Click on the application target.
3. Select the Build Phase tab in the Settings editor.
4. Click the + icon in the upper left corner of the main panel.
5. Select New Run Script Phase from the dropdown.
6. In the script box, add the following lines:
export ADRUM_ACCOUNT_NAME="<Account_Name_HERE>" // From the View
License - End User Monitoring section of the License Page
export ADRUM_LICENSE_KEY="<License_Key_HERE>" // From the View
License - End User Monitoring section of the License Page
SCRIPT=$(/usr/bin/find "${SRCROOT}" -name
xcode_build_dsym_upload.sh | head -n 1)
/bin/sh "${SCRIPT}"

7. There are also some optional parameters you can set if desired. To set them, add the following line(s) after the second export
statement above. Set to 1 to enable.

export ADRUM_UPLOAD_WHEN_BUILT_FOR_SIMULATOR=0
export ADRUM_TREAT_UPLOAD_FAILURES_AS_ERRORS=0
export ADRUM_EUM_PROCESSOR="<EUM_SERVER_URL>"

The last statement should be used to set the URL for an on-prem version of the EUM Server.

Upload the dSYM File to AppDynamics Manually
There are three steps to upload the file manually.
1. Get the dSYM file from Xcode
2. Upload the dSYM file to AppDynamics using the UI
or
Upload the dSYM File to AppDynamics Using the API
3. Check Uploaded dSYMs Using the REST API

Get the dSYM file from Xcode
1. In Xcode, run the Xcode build: Product > Build.
2. View the log navigator: View > Navigators > Show Report Navigator.
Older versions of Xcode used Show Log Navigator.

3. Click the log entry for the most recent build.
4. Near the end of the log, find and mouse over the log entry named Generate <Your_App_Name>.app.dSYM.
5. Click the button on the right side of the entry to expand it.
The end of the displayed command is the path to the dSYM file.
6. Navigate to this dSYM file in the Finder.
7. Right-click on the dSYM file and choose Compress.
8. Upload to AppDynamics the .zip file that Finder generates.

Upload the dSYM file to AppDynamics Using the UI
1.
2.
3.
4.

From the Mobile App menu, click Configuration.
Click Mobile App Configuration >.
From dSYM Mappings, click Upload dSYM package file for iOS crashes.
From the XCode dSYM package upload dialog, click Choose File.

Copyright © AppDynamics 2012-2019

Page 482


The uploader expects a file with a .zip extension.

5. In the file browser locate the zipped dSYM file for the application that you are instrumenting and click **Open**.

6. Click **Upload**.

**Upload the dSYM File to AppDynamics Using the REST API**

The API uses HTTP basic authentication. The username is your EUM account name and the password is your EUM license key.

**Set up your HTTP basic authentication credentials**

1. In the upper-right corner of the Controller UI, click the gear icon (⚙️) and choose **License**.
2. From the **Account Usage** tab, scroll down to the **User Experience** section.
3. Click **Show** next to License Key to display the EUM license key. This is your password for authentication.

   ![User Experience](image)

4. **URL-encode** the EUM account name and the EUM license key.

**Send the dSYM file**

Send the dSYM as a zip archive in the body of a **PUT** request to the following URI:

```
https://api.eum-appdynamics.com/v2/account/<EUM_Account_Name>/ios-dsym
```

You must set a Content-Type header, `-H Content-Type:application/octet-stream`, and your URL-encoded account name (the username) and license key (the password) to the call.

**Sample Request and Response**

This is a sample request and response using the REST API.

Upload Request

The following example uses **curl** to send a dSYM file named **UISampleApp.app.dSYM.zip**. The EUM account name is “Example account” and the password is “Example-License-Key-4e8ec2ae6cfe”, the EUM License Key. The plus signs replace spaces in the account name when the account name is URL-encoded.

```
curl -v -H Content-Type:application/octet-stream --upload-file
UISampleApp.app.dSYM.zip --user Example
account:Example-License-Key-4e8ec2ae6cfe
https://api.eum-appdynamics.com/v2/account/Example+account/ios-dsym
```

Upload Response

The successful output of the sample request looks like this:
Check Uploaded dSYMs Using the REST API

You can check to make sure that your dSYMs have successfully uploaded using two REST APIs.

1. Get a list of the UUIDs for the last 50 dSYMs you have uploaded
2. Check if a specific dSYM has been uploaded

List of the Last 50 dSYMs uploaded

The `dsymQuery` method allows you to retrieve a list of UUIDs for up to the last 50 dSYM files that have been uploaded to your account, along with the time they were uploaded. The response is displayed as JSON, by upload time, with the most recent first.

1. Set up your authentication as described in Upload the dSYM File to AppDynamics Using the REST API.
2. Create a GET request of the form:

   ```bash
   curl --user Example+account:Example-License-Key-4e8ec2ae6cfe
   https://<EUM_Cloud/Server_Host:Port>/v2/account/Example+account/crash-symbol-file-query/dsym
   ```

   where the value for `--user` is the authentication string you created in Upload the dSYM File to AppDynamics Using the REST API, `EUM_Cloud/Server` is `api.eum-appdynamics.com:443` for SaaS-based EUM Cloud accounts or as configured for on-premises EUM Servers.
Sample Request

```bash
curl --user Example+account:Example-License-Key-4e8ec2ae6cfe
```

Sample Response

```json
{"dSymFiles":[
{"uploadTime":"mm/dd/yyyy 14:15:32","UUID2":"my_uuid2"},
{"uploadTime":"mm/dd/yyyy 14:15:32","UUID":"my_uuid"}
]}
```

Check for specific dSYM by UUID

The `checkForDSymFile` method allows you to check if a specific dSYM by UUID has been uploaded. The upload time is returned in the response.

1. Set up your authentication as described in Upload the dSYM File to AppDynamics Using the REST API.
2. Create a GET request of the form:

   ```bash
curl --user Example+account:Example-License-Key-4e8ec2ae6cfe
   https://<EUM_Cloud/Server_Host:Port>/v2/account/Example+account/crash-symbol-file-query/dsym/uuid/<UUID_to_Check>
   ```

   where the value for `--user` is the authentication string you created in Upload the dSYM File to AppDynamics Using the REST API, `EUM_Cloud/Server` is api.eum-appdynamics.com:443 for SaaS-based EUM Cloud accounts or as configured for on-prem EUM Servers, and `UUID_to_Check` is the UUID of the dSYM in which you are interested.

Sample Request

```bash
curl --user Example+account:Example-License-Key-4e8ec2ae6cfe
https://<EUM_Cloud/Server_Host:Port>/v2/account/Example+account/crash-symbol-file-query/dsym/uuid/<UUID_to_Check>
```

Sample Response
{"uploadTime":"mm/dd/yyyy 14:15:32","UUID":"my_uuid"}
Manage the dSYM Files with Bitcode Enabled

On this page:

- Get and Upload the Correct dSYM File
- Upload Missing dSYM Files

Related pages:

- Upload the dSYM File

With iOS 9, Apple introduced a new mode of building and distributing iOS apps called App Thinning. As part of this initiative, apps containing bitcode are delivered.

If you choose to create your app containing bitcode, the process of acquiring your dSYM files for upload while you are in local development, debugging, and testing mode (Xcode on simulators or Devices) is exactly the same as in earlier versions.

It also works if you are using Ad Hoc Provisioning. But once you have used iTunes Connect to upload your application to Apple, for either TestFlight beta testing or application release, you have to take a few additional steps to get your dSYM files for use with AppDynamics.

Get and Upload the Correct dSYM File

As part of the process of preparing to upload your application using iTunes Connect, you create an archive. After the app is uploaded to iTunes Connect, it is processed by Apple. To have the correct version of the dSYM for AppDynamics, you must download this processed version to your local archive.

To download the processed dSYM file, in the Xcode Archives Organizer:

1. Make sure this version of the application is available for download (either TestFlight or the iTunes store).
2. In the Archives organizer, select the archive for this version.
3. Click the Download dSYMs button.

4. Right-click the .xcarchive file you receive and select Show in Finder. Right-click again and select Show Package Contents, and select the dSYMs folder.
For more information, see the Apple Developer docs.

Upload Missing dSYM Files

If you have crashes that AppDynamics could not symbolicate because it did not have a correct dSYM file, you get an error message in the Crashes Details dialog.

If you click Upload missing dSYM files in Instrumentation, a list of dSYM hashes appears. Match the missing files with the dSYMs in your archives, as shown above. If necessary, click Download dSYMs to update the archive. Upload the necessary files to AppDynamics.

Some of the missing dSYM files that AppDynamics reports may belong to simulation/debug builds that have crashed. In this case, the dSYMs may no longer be available.
The following sections provide instructions and tips for solving some common iOS instrumentation issues.

**Agent Not Detecting Requests**

First, confirm that you are using either the `NSURLConnection` or the `NSURLSession` class. By default, you are required to make network requests. If you are using a custom HTTP library, make sure you follow the instructions given in Use the Agent with a Custom HTTP Library.

If you're using `NSURLSession`, be sure to create an instance of `NSURLSession` after initializing the iOS agent. This is because the agent may not be aware of objects created before the instrumentation is initialized.

**Unrecognized Selector Error**

If you are instrumenting your app using the manual method and you see this error message:

```
+[NSURLConnection ADEumInsertHooks]: unrecognized selector sent to class <hex value>
```

you didn’t add `-ObjC` to your link flags. See Set the `-ObjC` Flag for instructions.
Instrument Android Applications

Before you can monitor your Android application, you need to build and instrument your application to enable the Android Agent to collect mobile metrics. The Android Agent supports monitoring for Android applications written in Java or any JVM-based language such as Kotlin.

After you have set up and accessed Mobile RUM, use one of the following ways to build and instrument your application:

- **Instrument an Application with the Android Agent Installer** - The Android Agent Installer is an Android Studio plugin that will configure the build and insert the instrumentation code for you.
- **Instrument an Android Application Manually** - If you are unable to use the Android Studio plugin, follow these instructions to configure your build and instrument your application.

For further customization, see [Customize the Android Build](#) and [Customize the Android Instrumentation](#).
Instrument an Application with the Android Agent Installer

1. Check the Prerequisites
2. Get the EUM App Key

The plugin is recommended for initial basic instrumentation. If your app requires custom initialization or more advanced configuration, see Customize the Android Build and Customize the Android Instrumentation.

Check the Prerequisites

To use the plugin, you need the following:

- an AppDynamics account - If you don't have an account yet, sign up for a free account.
- Android Studio versions 2.1+, 3.2.1+ (versions 3.0+ to 3.2.0+ are not supported)
- Java 7+

If you are running an on-premises deployment, you will also need the URL to your EUM Server.

Get the EUM App Key

1. From the top navigation bar of your Controller UI, go to User Experience > Mobile Apps.
2. Click Add App > Android to open the Getting Started Wizard - Android page.
2. Click Android Studio Plugin.
3. Select the Create a new Mobile App Group radio button, enter an application name, and click Continue.
4. Copy the displayed App Key and the Collector URL.
5. Follow the instructions in Install and Run the AppDynamics Plugin.
3 Install and Run the AppDynamics Plugin

From the main menu bar, open the Android Studio Preferences panel by clicking “Android Studio” > “Preferences”.

7. Keep the wizard open until the instrumentation has been verified in the wizard.

5 Verify Your Instrumentation

Generate network requests on your instrumented application.
Instrument an Android Application Manually

On this page:

- Build the Android Application
- Instrument the Android Application
- Upgrade the Android Mobile Agent

Related pages:

- Verify the Android Instrumentation
- Confirm the Mobile Agent Connected to the Controller
- Customize the Android Build
- Customize the Android Instrumentation

To instrument Android Applications, you need to first build the application for the platform you are using and then add the instrumentation code.

To get started, follow the instructions below:

1. Build the Android Application
2. Instrument the Android Application

Build the Android Application

To build your application, follow the instructions for your platform:

- Gradle/Android Studio
- Maven
- Ant

If you are fetching the AppDynamics Android SDK from the AppDynamics Download page, see Download Manually.

**Gradle/Android Studio**

Complete the following steps to configure the build for your Android application:

1. Confirm the compatibility of your Gradle, Android Tools with the AppDynamics plugin versions.
2. Install the Android Agent.
3. Activate the plugin.

**Install the Android Agent**

Use the native package system to install the Android Agent. In the app module build.gradle, add the class path of the AppDynamics Gradle Plugin to the build path dependencies clause. Use `com.appdynamics:appdynamics-gradle-plugin:4.5` unless you need to use another version of the AppDynamics plugin for compatibility between your Gradle and Android Tools.
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:1.1.0'
        classpath 'com.appdynamics:appdynamics-gradle-plugin:4.5.+' // this line added for AppDynamics
    }
}
allprojects {
    repositories {
        jcenter()
    }
}

Activate the Plugin

In your module-level build.gradle, add the adeum plugin immediately after the com.android.application plugin, so that it looks similar to the example below:

```gradle
apply plugin: 'com.android.application'
apply plugin: 'adeum' // this line added for AppDynamics
```

Apache Maven Project

If your application is a Maven Project:

1. Add the following code to the <dependencies> section:

```xml
<dependency>
    <groupId>com.appdynamics</groupId>
    <artifactId>appdynamics-runtime</artifactId>
    <version>1.0</version>
</dependency>
```

2. Add the following code to the <plugins> section:
<plugin>
  <groupId>com.appdynamics</groupId>
  <artifactId>appdynamics-maven-plugin</artifactId>
  <version>1.0</version>
  <executions>
    <execution>
      <phase>compile</phase>
      <goals>
        <goal>adinject</goal>
      </goals>
    </execution>
  </executions>
</plugin>

**Ant/Eclipse**

See the Knowledge Base article *Use Ant to Build Android Apps with the AppDynamics Android SDK* for instructions.

**Instrument the Android Application**

After you have completed building your application, follow the steps below:

1. Get the Application Key
2. Add the Required Permissions
3. Modify the Source
4. Run the Build

**Get Application Key**

After you completed the *Getting Started Wizard*, you were given an EUM App Key. You will need this key when you modify the source code. In some cases, multiple mobile applications can share the same key.

If you have completed the *Getting Started Wizard*, but don't have your EUM App Key, see *Get Your Application Key*.

**Add the Required Permissions**

Open your application's AndroidManifest.xml file and verify that it has these permissions:

```xml
<uses-permission
  android:name="android.permission.INTERNET"></uses-permission>
<uses-permission
  android:name="android.permission.ACCESS_NETWORK_STATE"></uses-permission>
```

If both of these permissions are not present, add them.

**Modify the Source**
1. In the source file that defines your application's primary Activity, add the following import:

```java
import com.appdynamics.eumagent.runtime.Instrumentation;
```

2. In your primary Activity's `onCreate()` method, add the following lines, passing in the EUM App Key from step 2 above:

```java
Instrumentation.start(<EUM_APP_KEY>, getApplicationContext());
```

3. Save the file.
Your code should look something like this.

```java
import com.appdynamics.eumagent.runtime.Instrumentation;
...
@Override public void onCreate(Bundle savedInstanceState) {
    Instrumentation.start(<EUM_APP_KEY>, getApplicationContext());
    ...
}
```

**Verify the Instrumentation**

See Verify the Android Instrumentation for build and verification instructions.

**Upgrade the Android Mobile Agent**

As new features are added to the agent, you will need to upgrade the Android SDK in your app.

To upgrade, you simply update the build file for your platform:

- Gradle/Android Studio
- Maven

The process of installing and updating the latest Android SDK is the same.
Verify the Android Instrumentation

On this page:
- Run the Build
- Verify the Instrumentation

To verify your Android instrumentation:

1. Run the build.
2. Verify the instrumentation.
3. Confirm the agent connected to the Controller.

Run the Build

Build your Android app following the instructions for your build system:
- Android Studio
- Gradle
- Maven

In the console, you should see something like this:

```
[injector] /=========================================/
[injector] | AppDynamics BCI Instrumentation summary |
[injector] \=========================================/
[injector] - Total number of classes visited (#720 classes)
[injector] - Total number of classes instrumented (#1 classes)
[injector] - Total number of classes failed to instrument (#2 classes)
[injector] - Total number of features discovered (#3)
```

Verify the Instrumentation

Based on the build system you used, verify that the instrumentation was successful:
- Gradle/Android Agent Installer Plugin
- Maven
- Ant

**Gradle/Android Agent Installer Plugin**

If you didn't use the `-i` flag, check to make sure there is a line in your console output that contains "inject". If you don't see this information printed in your console, either your project is incorrectly configured or the injector failed to run completely. There is a very detailed log of this process either at `<project>/target/appdynamics_eum_android_bci.log` or `<module>/target/appdynamics_eum_android_bci.log`.

**Maven**

If you don't see this information printed in your console, either your project is incorrectly configured or the injector failed to run.
completely. There is a very detailed log of this process either at `<project>/target/appdynamics_eum_android_bci.log` or `<module>/target/appdynamics_eum_android_bci.log`.

**Ant**

If you don't see this information printed in your console, either your project is incorrectly configured or the injector failed to run completely. There is a very detailed log of this process either at `<project>/target/appdynamics_eum_android_bci.log` or `<module>/target/appdynamics_eum_android_bci.log`. 
Customize the Android Build

Related pages:
- Instrument an Android Application Manually
- Customize the Android Instrumentation

Build the Android Application showed you how to use the bare minimum configuration to build your application.

The following pages show you how to add additional configuration to customize your build:

- Automatically Upload Mapping Files
- Configure ProGuard to Prevent Obfuscation and Class Removal
- Exclude Classes from Being Instrumented
- Enable/Disable Instrumentation for Build Types
- Enable/Disable Native Crash Reporting
Automatically Upload Mapping Files

On this page:
- Uploading to the SaaS EUM Server
- Uploading to the On-Prem EUM Server
- Modifying the Default Upload Behavior
- Configure Proguard to Prevent Obfuscation and Class Removal

Related pages:
- Configure Proguard to Prevent Obfuscation and Class Removal

You can configure the build to add your EUM account information so that you can automatically upload your ProGuard or DexGuard mapping files for crash reports with each build.

This is the recommended way of managing these types of mapping files, although manual modes are available for uploading mapping files. If you do not use ProGuard or DexGuard to obfuscate your files you can ignore this section.

DexGuard Limitation
DexGuard mapping files must be in the default location. This means that their dexguard configurations should not specify --printMapping, which changes where to print the mapping file.

Uploading to the SaaS EUM Server

If you are using the SaaS EUM Server, you just need to provide your account name and license key to automate uploading as shown below.

```java
adeum {
  // Add this information to allow the upload of ProGuard or DexGuard mapping files.
  // If you are using the EUM Cloud (SaaS), this automatically uploads your files with each build
  account {
    name "The EUM Account Name from the License screen"
    licenseKey "The EUM License Key from the License screen"
  }
}
```

Uploading to the On-Prem EUM Server

If you are using an on-prem deployment, in addition to supplying your account name and license, you must also assign the URL to your on-prem EUM Server to the url property as shown below.
adeum{
    // The account information is also needed for on-prem deployments.
    account {
        name "The EUM Account Name from the License screen"
        licenseKey "The EUM License Key from the License screen"
    }
    ...
    // Add this information to point to the on-prem EUM Server.
    url "https://<your-on-prem-eum-server>:7001"
}

Modifying the Default Upload Behavior

You can also modify the default upload behavior for both on-prem and SaaS deployments. For on-prem deployments, you will need to provide the URL to your on-prem EUM Server with the `url` property.

The configuration example below sets `failBuildOnUploadFailure` to `true`, so that the build will fail if the upload to the URL specified by `url` is unreachable. The object `proguardMappingFileUpload` is used for both Proguard and DexGuard mapping files. If you don't want your build to fail because the ProGuard or DexGuard mapping file couldn't be uploaded, do not modify the default setting.

adeum{
    ...
    // Add this information to point to the on-prem EUM Server.
    url "https://<your-on-prem-eum-server>:7001"

    // Add this information if you want to modify upload behavior.
    proguardMappingFileUpload {
        failBuildOnUploadFailure true // If true, will fail build. Defaults to false.
        enabled true // Enables automatic uploads. Defaults to true
    }
}

Configure Proguard to Prevent Obfuscation and Class Removal

If you use ProGuard to verify or optimize your code, add the following lines to the ProGuard configuration file, the default is `ProGuard.cfg`, to prevent ProGuard from obfuscating or removing classes needed for proper instrumentation.
-keep class com.appdynamics.eumagent.runtime.DontObfuscate
-keep @com.appdynamics.eumagent.runtime.DontObfuscate class * { *; }
Configure ProGuard to Prevent Obfuscation and Class Removal

Related pages:
- Automatically Upload Mapping Files

If you use ProGuard to verify or optimize your code, add the following lines to the ProGuard configuration file, the default is `ProGuard.cfg`, to prevent ProGuard from obfuscating or removing classes needed for proper instrumentation.

```
-keep class com.appdynamics.eumagent.runtime.DontObfuscate
-keep @com.appdynamics.eumagent.runtime.DontObfuscate class * { *; }
```
Exclude Classes from Being Instrumented

You can exclude one or more class names from being instrumented, using the "adeum" excludeClasses snippet.

Class names can contain wildcards:

- '?' for a single character
- '*' for any number of characters, but not the package separator
- '"' for any number of (any) characters
- '$' prefix for an inner class name

To exclude named classes from instrumentation, add the class names to the excludeClasses array, using matching patterns where needed:

```plaintext
...
adeum {
    // Excludes all the classes in the android.support.multidex and okio packages.
    excludeClasses = ['android.support.multidex.*', 'okio.*']
}
...
```
Enable/Disable Instrumentation for Build Types

By default, the instrumentation is enabled for both release and debug builds. In certain cases, however, you might want to disable instrumentation for a certain build.

For example, if you are tracking down an issue in your code, you might want to disable instrumentation in the debug build and enable instrumentation in the release build. To do this, in the adeum object, set enabledForDebugBuilds to false and set enabledForReleaseBuilds to true as shown below.

```plaintext
... 
adeum {
    // other stuff, if it exists
    // Optional.
    // By default, instrumentation is enabled for both debug and release builds.
    // This controls instrumentation for "debuggable" build types.
    enabledForDebugBuilds = false
    // This controls for non-debuggable build types.
    enabledForReleaseBuilds = true
}
...
```

Verify the Instrumentation State at Runtime

If one or more of your builds have disabled instrumentation, you need to disable the instrumentation check at runtime. You can set boolean fields in the build configuration to disable the runtime verification of instrumentation. The runtime verification of instrumentation is on by default. You can turn it off using the method withAutoInstrument.

In the build configuration, you can set a boolean value for the CHECK_ENABLED field for build types. The table below shows the config value, the instrumentation state, and then describes the runtime behavior of your applications.

<table>
<thead>
<tr>
<th>Config Value</th>
<th>Instrumentation State</th>
<th>Runtime Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Enabled</td>
<td>The agent will verify that instrumentation has been enabled before initializing.</td>
</tr>
<tr>
<td></td>
<td>Disabled</td>
<td>The agent will verify that the instrumentation has not been enabled and then thrown an IllegalArgumentException exception.</td>
</tr>
<tr>
<td>false</td>
<td>Enabled</td>
<td>The agent will initialize without checking whether instrumentation has been enabled.</td>
</tr>
<tr>
<td></td>
<td>Disabled</td>
<td></td>
</tr>
</tbody>
</table>

For example, in the build configuration below, the field CHECK_ENABLED is set to true for the release build and false for the debug build.

The agent for the debug build will not verify if instrumentation has been enabled before executing instrumentation code, whereas, the application code for the release build will check before executing the instrumentation code.
If the value of `CHECK_ENABLED` is `true`, the Android Agent will confirm that instrumentation has been enabled before executing the initialization code. If the value is `false`, the Android Agent executes the initialization code regardless of whether instrumentation has been enabled.

The following instrumentation initialization code shows you how to check the value of the build config `CHECK_ENABLED` with the method `withCompileTimeInstrumentationCheck`.

```java
android {
    // usual stuff
    buildTypes {
        // usual stuff
        release {//these lines added for AppDynamics
            //release based configuration
            // The release build by default is not "debuggable".
            // The build config "CHECK_ENABLED" will be accessible in the runtime environment.
            // This enables the Android Agent to verify that the instrumentation has been enabled before running the initialization code.
            // If instrumentation has not been enabled, an "IllegalState" exception is thrown.
            buildConfigField "boolean", "CHECK_ENABLED", "true"
        }
        debug {
            // Setting 'CHECKED_ENABLED' to "false" means the Android Agent will run the initialization code without confirming that instrumentation has been enabled. No exception will be thrown.
            buildConfigField "boolean", "CHECK_ENABLED", "false"
        }
    }
}
...
import com.appdynamics.eumagent.runtime.Instrumentation;
...
Instrumentation.start(
    AgentConfiguration config = AgentConfiguration.builder()
        .withAppKey("EUM_APP_KEY")
        .withContext(this)

        .withCompileTimeInstrumentationCheck(BuildConfig.CHECK_ENABLED)
        .build();
);
Enable/Disable Native Crash Reporting

On this page:

- How It Works
- Configuration for Native Crash Reporting
- View Source Code Information in the Stack Trace

You can report native crashes caused by native libraries. This feature is available for all architectures supporting the Android Native Development Kit (NDK).

How It Works

When instrumentation is initialized, the native crash handler loads the AppDynamics library libadeum. At entry points of your code, a signal handler is injected into the running process. After a signal is handled, libadeum restores the previous signal handler. (This enables you to use different methods to handle signals.)

The handler collects and writes the information about the native crash into a temporary file. When the application is restarted, the temporary file is read, processed, and the crash information is displayed in the Controller UI.

Configuration for Native Crash Reporting

By default, however, this feature is turned off, so you need to add the section nativeCrashHandling and set enabled to true. In addition, you point the library paths in the variantLibraryPaths array for each build variant to write the symbol files locally and transmit the symbol files to the EUM Server. You can also specify a build ID for a project build. After you build the project, the build ID can be found in build/appDynamics/ndkSym<VariantName>/<AppBuildID>. If not specified, the build ID of the most recently built project will be used.

```plaintext
adeum {
  // Other configuration if needed.
  nativeCrashHandling {
    enabled = true
    symbolUpload {
      buildId = "<your_custom_build_id>"
      variantLibraryPaths = ["release": "ndkLib/obj/local",
                             "debug": "ndkLib/obj/debug", ...]
    }
  }
}
```

View Source Code Information in the Stack Trace

You have two different ways to view symbol information in the stack trace:

1. From the Crash Details dialog, download the crash report and use a utility like ndk-stack to view the crash report with source code level symbolizations.
2. Upload the symbol files and then view the stack trace in the Crash Details dialog.

Download the Crash Report and Use ndk-stack

1. From the Crash Dashboard, double-click one of the crashes listed in the Unique Crashes widget.
2. From the Crash Details dialog, click Download to download the crash report.
3. Run `ndk-stack` (or a similar utility) on the downloaded crash report as the input file to generate a crash report with source code level symbolications.

**Upload the Symbol File and View the Stack Trace**

You are recommended to automatically generate and upload symbol files, but you can also manually generate and upload the symbol files, too. After you upload the symbol files, when the app with the same build ID as the UUID of the uploaded `ndkSYM` file crashes, you will see the file name and the line number next to the C/C++ function name for each frame of the stack trace shown in the Crash Details dialog.

**Automatically Generate and Upload Symbol Files**

You can run the following `gradle` command to generate and upload the symbol files to the EUM Server. Replace `<VariantName>` with the build variant names you defined in the configuration. For example, in the configuration above, the `<VariantName>` could be `release` or `debug`.

```
$ gradle appDynamicsUploadNDKSymbolFile<VariantName>
```

**Manually Generate and Upload Symbol Files**

To generate symbol files, run the following `gradle` command, where `<VariantName>` is the build variant names. The generated symbol file will be written to `build/appDynamics/ndkSym<VariantName>/<AppBuildID>/<AppBuildID>.ndkSYM.zip`.

```
$ gradle appDynamicsGenerateNDKSymbolFile<VariantName>
```

To manually upload the generated symbol file, use the following `cURL` command, replacing the `<ndkSymZipFile>` with your generated symbol file, `<Account Name>` with your EUM Account name, and `<License Key>` with your EUM License Key.

```
$ curl -v -H Content-Type:application/octet-stream --upload-file <ndkSymZipFile> --user <Account Name>:<License Key> https://api.eum-appdynamics.com/v2/account/<Account Name>/android-ndksym
```
Customize the Android Instrumentation

On this page:
- Collect Additional Types of Data
- Add a Crash Reporting Callback
- Report Errors and Exceptions
- Configure Hybrid Support
- Programmatically Control Sessions
- Start and End Session Frames
- Use a Custom HTTP Library
- Capture User Interactions
- Configure and Take Screenshots
- Transform URLs for Network Requests
- Enable Logging and Set Logging Level

Related pages:
- Instrument an Application with the Android Agent Installer
- Instrument an Android Application Manually
- Android SDK Documentation

Once you have instrumented your Android application with the Android SDK, you can also use the APIs exposed by the SDK to customize the data for your app that appears in the Controller UI.

Because the agent stores data about events in a local buffer before reporting the information, you are recommended to use the APIs with discretion.

Collect Additional Types of Data

The Instrumentation class has additional methods to allow you to extend the kinds of application data you can collect and aggregate using Mobile RUM. There are six basic kinds of extensions that you can create:

<table>
<thead>
<tr>
<th>Type of Data</th>
<th>Description</th>
<th>Specifications</th>
<th>Where Data is Displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Info points</td>
<td>How often a method is invoked, and how long it takes to run.</td>
<td>• Data is numeric&lt;br&gt;• Names must consist of alphanumeric characters and/or spaces</td>
<td>• Metric Browser&lt;br&gt;• Custom Data&lt;br&gt;• Network Request&lt;br&gt;• Snapshots&lt;br&gt;• Mobile Sessions&lt;br&gt;• Network Request&lt;br&gt;• Analyze</td>
</tr>
<tr>
<td>Custom timers</td>
<td>Any arbitrary sequence of events within your code timed, even spanning multiple methods.</td>
<td>• Data is numeric&lt;br&gt;• Metric names must consist of alphanumeric characters and/or spaces</td>
<td>• Metric Browser&lt;br&gt;• Custom Data</td>
</tr>
<tr>
<td>Custom metrics</td>
<td>Any integer-based data you wish to collect.</td>
<td>• Data is numeric&lt;br&gt;• Metric names must consist of alphanumeric characters and/or spaces</td>
<td>• Metric Browser&lt;br&gt;• Custom Data</td>
</tr>
<tr>
<td>User data</td>
<td>Any string key/value pair you think might be useful.</td>
<td>• Data can be any type&lt;br&gt;• Metric names have no restrictions</td>
<td>• Network Request Snapshots&lt;br&gt;• Mobile Sessions&lt;br&gt;• Network Request Analyze</td>
</tr>
<tr>
<td>Breadcrumbs</td>
<td>The context for a crash.</td>
<td>• Data can be any data type&lt;br&gt;• Metric names have no restrictions</td>
<td>• Network Request Snapshots&lt;br&gt;• Mobile Sessions&lt;br&gt;• Network Request Analyze</td>
</tr>
</tbody>
</table>
User interactions | Capture when users press buttons, click on lists, and select text. |
|------------------|-------------------------------------------------|

- Data can be any data type
- Metric names have no restrictions
- Network Request Snapshots
- Mobile Sessions
- Network Request Analyze

When you have set up info points, custom timers, custom metrics, and/or user data, the Mobile Agent packages that data in a mobile beacon. Normally, the beacon is transmitted when the instrumented app sends an HTTP request or when the app is restarted following a crash, but if custom data has been collected and neither of those events has occurred for at least 5 minutes, the custom data is sent on at that time.

**Info Points**

Information points allow you to track how your own code is running. You can see how often a method is invoked, how long it takes to run, and if an exception is thrown. The simplest way to set up an information point is to use the `@InfoPoint` annotation. For example:

```java
@InfoPoint
public void infoPointMethod(String arg1, int arg2, long value) {
    System.out.println("Executing infoPointMethod!");
}
```

You can also do this manually, using the `CallTracker` interface. For example, to collect information on your `downloadImage` method, you could use code similar to this:

```java
private void downloadImage(URL url) {
    CallTracker tracker =
    Instrumentation.beginCall("com.example.android.awesomeapp.ImageDownloader",
        "downloadImage")
        .withArguments(url);
    try {
        //download image.
        tracker.reportCallEnded();
    } catch(Exception e) {
        //handle exception thrown
        tracker.reportCallEndedWithException(e);
    }
}
```

This information appears in the **Custom Data** view of the Controller UI.

**Custom Timers**

Custom timers allow you to time any arbitrary sequence of events within your code, even spanning multiple methods, by using `startTimer` and `stopTimer`. 
public class MyActivity extends Activity {
    @Override
    protected void onStart(){
        Instrumentation.startTimer("Time Spent on MyActivity");
        //your code here.
    }
    
    @Override
    protected void onStop(){
        Instrumentation.stopTimer("Time Spent on MyActivity");
        //your code here.
    }
}

The methods startTimer(String) and stopTime(String) can be called from different threads. Calling startTimer again with the same name value resets a named timer.

This information appears in the Custom Data view of the Controller UI.

**Custom Metrics**

Any integer-based data can be passed to the agent. The first parameter to the reportMetric call is the name you want the metric to appear under in the Controller UI. The metric name should only contain alphanumeric characters and spaces. Illegal characters are replaced by their ASCII hex value.

For example, to track the number of times your users click the checkout button in your UI, you could use code similar to this.

```java
findViewById(R.id.checkout_button).setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View view){
        //run your checkout routine.
        Instrumentation.reportMetric("Checkout Count", 1);
    }
});
```

This information appears in the Custom Data view of the Controller UI.

**User Data**

You can set any string key/value pair you think might be useful. The first parameter to the setUserData call is the key you want to use, which must be unique across your application. The second is the value you want to be assigned to the key.

For example:
void onUserLoggedIn(String userid) {
    Instrumentation.setUserData("User ID", userid);
    ...
}

This information is available in Network Request Analyze and is added to any crash snapshots that may be taken. Keys and values are limited to 2048 characters each.

You can also set user data with values of other types (Long, Boolean, Double, Date) using the following methods:

- setUserDataLong
- setUserDataBoolean
- setUserDataDouble
- setUserDataDate

**Breadcrumbs**

Breadcrumbs allow you to situate a crash in the context of your user's experience. Set a breadcrumb when something interesting happens. If your application crashes at some point in the future, the breadcrumb will be displayed along with the crash report.

There are two ways of leaving breadcrumbs:

- Crash Reports Only
- Modal

Using this method means that breadcrumbs are reported in crash reports only.

```
public static void leaveBreadcrumb(java.lang.String breadcrumb)
```

Using this method lets you fine tune where the breadcrumbs are reported, either only in crash reports or in crash reports and sessions.

```
public static void leaveBreadcrumb(java.lang.String breadcrumb, int mode)
```

Where `mode` is either:

- CRASHES_ONLY
- CRASHES_AND_SESSIONS

If the `breadcrumb` is over 2048 characters, it is truncated. If it is empty, no breadcrumb is recorded. Each crash report displays the most recent 99 breadcrumbs.
Add a Crash Reporting Callback

You may want to make crash report information that Mobile RUM collects available to other parts of your code, for example, to Google Analytics, if you are using it. To enable passing on summary crash information, you can set up a crash report runtime callback. To get a callback when the Android Agent detects and then reports a crash, you need to implement the following interface in your code:

```java
public interface CrashReportCallback {
    void onCrashesReported(Collection<CrashReportSummary> summaries);
}
```

The method `onCrashesReported` is invoked during the next initialization of the agent after a crash has occurred.

We send a Collection instead of an individual callback because there could be more than one crash, even though there typically is only one.

This callback is invoked on your app's UI thread, so any work should be done on a separate work thread.

Each `CrashReportSummary` has the following properties:

```java
public class CrashReportSummary {
    public final String crashId;
    public final String exceptionClass;
    public final String exceptionMessage;
}
```

If you are sending the information to another analytics tool, such as Google Analytics, it is best to include all three properties: `exceptionClass` and `exceptionMessage` are useful for quick identification of the crash, but for more detailed information, `crashId` can be used to look up the crash in the AppDynamics Controller UI.

For example, to print the crash information to Android's logger, you could implement a `CrashReportCallback` class like this:
public static class MyCrashReportCallback implements CrashReportCallback {
    @Override
    public void onCrashesReported(Collection<CrashReportSummary> summaries) {
        for (CrashReportSummary crash : summaries) {
            Log.e("MyApp", "Crash Detected: " + crash.exceptionClass + " : " + crash.exceptionMessage + " (" + crash.crashId + ")");
        }
    }
}

You set your callback as using the AgentConfiguration object:

```java
final AgentConfiguration config = AgentConfiguration.builder()
    .withAppKey(appKey)
    .withContext(context)
    .withCrashCallback(new MyCrashReportCallback())
    .build();
```

Your callback is invoked after a crash, during the next initialization, on the main thread. For more information, see the latest JavaDocs or the complete Android SDK API.

**Report Errors and Exceptions**

You can report exceptions using the method `reportError` from the Instrumentation class. Reported exceptions will appear in session details.

You can also set one of the severity levels below for an issue. With the severity level, you can filter errors in the **Code Issues Dashboard** or **Code Issues Analyze**.

- ErrorSeverityLevel.INFO
- ErrorSeverityLevel.WARNING
- ErrorSeverityLevel.CRITICAL

The example below uses the API to report possible exceptions and sets the severity level to `ErrorSeverityLevel.CRITICAL` (critical) when writing to a file.
private void writeToFile(String filePath, String data) {
    try {
        OutputStream outputStream = new FileOutputStream(filePath);
        Writer outputStreamWriter = new OutputStreamWriter(outputStream);
        outputStreamWriter.write(data);
        outputStreamWriter.close();
    } catch (IOException e) {
        Log.e("Exception", "File write failed: " + e.toString());
        Instrumentation.reportError(e, ErrorSeverityLevel.CRITICAL);
    }
}

Configure Hybrid Support

By default, the Android Agent instruments Android WebViews. The Android Agent is able to do this by injecting the JavaScript Agent into WebViews. See Hybrid Application Support for an overview and an explanation of how it works.

Runtime Configuration for Hybrid Support

The code example below disables the injection of the JavaScript Agent. If the client receives a false for this flag, then the JavaScript Agent will be disabled. Thus, WebViews will not be instrumented, and Ajax requests will not be monitored.

Instrumentation.start(AgentConfiguration.builder()
    .withAppKey("<EUM_APP_KEY>")
    .withContext(getApplicationContext())
    .jsAgentInjectionEnabled(false)
    .build());

The injection occurs during the creation of a new WKWebView. So, if a WKWebView is created when this flag is set to false, that particular WKWebView won't be instrumented even if the flag is subsequently set to true.

The collection and reporting of Ajax calls are disabled by default. To enable the injection and the collection and reporting of Ajax calls, pass true to the method jsAgentEnabled in the instrumentation configuration as shown below.
Programmatically Control Sessions

By default, a mobile session ends after a period of user inactivity. For example, when a user opens your application, the session begins and only ends after the user stops using the app for a set period of time. When the user begins to use the application again, a new session begins.

Instead of having a period of inactivity to define the duration of a session, however, you can use the following API to programmatically control when sessions begin and end:

```java
void startNextSession()
```

When you call the method `startNextSession` from the `Instrumentation` class, the current session ends and a new session begins. The API enables you to define and frame your sessions so that they align more closely with business goals and expected user flows. For example, you could use the API to define a session that tracks a purchase of a product or registers a new user.

Excessive use of this API will cause sessions to be throttled (excessive use is >10 calls per minute per Android Agent, but is subject to change). When not using the API, sessions will fall back to the default of ending after a period of user inactivity.

**Example of a Programmatically Controlled Session**

In the code example below, the current session ends and a new one begins when the check out is made.

```java
public void checkoutCart(){
    if (currentCartItems!=null && currentCartItems.size()>0){
        CheckoutTask checkoutReq = new CheckoutTask();
        checkoutReq.execute(getEndpoint() + "cart/co");
        currentCartItemsMap.clear();
        convertItemsMapToList();
        Instrumentation.startNextSession();
    } else {
        displayToast("There are no items in the cart");
    }
}
```
Start and End Session Frames

You can use the SessionFrame API to create session frames that will appear in the session activity. Session frames provide context for what the user is doing during a session. With the API, you can improve the names of user screens and chronicle user flows within a business context.

Use Cases

The following are common use cases for using the SessionFrame API:

- One Activity performs multiple functions and you want more granular tracking of the individual functions.
- A user flow spans multiple activities or user interactions. For example, you could use the API to create the session frames "Login", "Product Selection", and "Purchase" to chronicle the user flow for purchases.
- You want to capture dynamic information based on user interactions to name session frames, such as an order ID.

SessionFrame API

The table below lists the three methods you can use with session frames.

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instrumentation</td>
<td>static SessionFrame startSessionFrame(String sessionFrameName)</td>
<td>Use this to start and name your session frame. Naming session frames enable you to easily identify and track the frames in the Sessions Dialog.</td>
</tr>
<tr>
<td>SessionFrame</td>
<td>static void updateName(String updatedSessionFrameName)</td>
<td>Use this to start and name your session frame. Naming session frames enable you to easily identify and track the frames in the Sessions Dialog.</td>
</tr>
<tr>
<td>SessionFrame</td>
<td>static void end()</td>
<td>End the session frame. The method is called from the SessionFrame object returned from startSessionFrame.</td>
</tr>
</tbody>
</table>

Session Frame Example

In the following example, the ShoppingCartActivity class uses the SessionFrame API to track user activity during the checkout process.
public class ShoppingCartActivity extends Activity {

    SessionFrame checkoutSessionFrame;

    public void onCheckoutCartButtonClicked() {
        // The user starts the checkout by clicking the checkout button.
        // This may be after they have updated the quantities of items in the
        // cart, etc.
        checkoutSessionFrame = Instrumentation.startSessionFrame("Checkout");
    }

    public void onConfirmOrderButtonClicked() {
        // Once they have confirmed payment info and shipping information,
        // and they
        // are clicking the "Confirm" button to start the backend process of
        // checking out,
        // we may know more information about the order itself, such as an
        // order ID.
        checkoutSessionFrame.updateName("Checkout: Order ID " + orderId);
    }

    public void onProcessOrderCompleted() {
        // Once the order is processed, the user is done "checking out", so
        // we end the session frame.
        checkoutSessionFrame.end();
        checkoutSessionFrame = null;
    }

    public void onCheckoutCanceled() {
        // If the user cancels or returns to the cart, you'll want to end the
        // session frame also, or else it will be
        // left open and appear to have never ended.
        checkoutSessionFrame.end();
        checkoutSessionFrame = null;
    }
}

Use a Custom HTTP Library

The Android Agent automatically detects network requests when the underlying implementation is handled by any one of the supported
network libraries. To have the Android Agent detect requests from a custom library, add request tracking code to your application
manually, using the HttpRequestTracker interface.

Supported Network Libraries

The libraries below cover the great majority of Android network requests. In some cases, however, mobile applications use custom
HTTP libraries.
To set headers to allow correlation with server-side processing, use the `ServerCorrelationHeaders` class.

### Add Request Tracking

To add request tracking manually, you use an `HttpRequestTracker` object to tell the agent when the request begins and when it ends and to report fields of the response to the agent.

### Tracking a request

To begin tracking an HTTP request, use an instance of the following interface.

```java
public interface HttpRequestTracker {
    public Exception getException();
    public HttpRequestTracker withException(Exception e);

    public String getError();
    public HttpRequestTracker withError(String error);

    public int getResponseCode();
    public HttpRequestTracker withResponseCode(int responseCode);

    public Map<String, List<String>> getResponseHeaderFields();
    public HttpRequestTracker withResponseHeaderFields(Map<String, List<String>> responseHeaderFields);

    /**
     * Stops tracking an HTTP request.
     * Immediately after receiving a response or an error, set the appropriate fields and call this method to report the outcome of the HTTP request. You should not continue to use this object after calling this method -- if you need to track another request, obtain a new instance.
     */
    public void reportDone();
}
```

**Example:**

You must initialize the agent using the `Instrumentation.start` method before using this interface.
Given a request snippet like this:

```java
public byte[] sendRequest(URL url) throws HttpException {
    try {
        // implementation omitted
        return responseBody;
    } catch (UnderlyingException e) {
        throw new HttpException(e);
    }
}
```

Adding the tracker could look something like this:

```java
public byte[] sendRequest(URL url) throws HttpException {
    HttpRequestTracker tracker = Instrumentation.beginHttpRequest(url);
    try {
        // implementation omitted
        tracker.withResponseCode(theResponseCode)
            .withResponseHeaderFields(theResponseHeaderFields)
            .reportDone();
        return responseBody;
    } catch (UnderlyingException e) {
        tracker.withException(e)
            .reportDone();
        throw new HttpException(e);
    }
}
```

**Enable Server-Side Correlation**

To enable correlation between your request and server-side processing, add specific headers to outgoing requests that the server-side agent can detect.

This is done automatically for standard HTTP libraries.

```java
public class ServerCorrelationHeaders {
    public static Map<String, List<String>> generate();
}
```
You must:

1. Call the `generate` method and set the generated headers before sending a request to the backend.
2. Report back the response headers, using data from the `withResponseHeaderFields` field.

**Override the Request/Response Content-Length**

You can generally obtain the content lengths of the network request and response by passing the headers with `HttpRequestTracker.withRequestHeaderFields()` and `HttpRequestTracker.withResponseHeaderFields()`.

If for some reason this does not work for your custom HTTP tracking—for example, the network library doesn't populate those fields until its being transmitted—then you can still report the request and response content lengths using `HttpRequestTracker.withRequestContentLength(Long length)` and `HttpRequestTracker.withResponseContentLength(Long length)`.

For example, suppose you want to track a request that has a byte array of content. You could report the request content length by passing the size of the byte array as shown below.

```java
byte[] requestContent;
HttpRequestTracker tracker;
tracker.withRequestContentLength(requestContent.size());
```

Use the `AgentConfiguration` Object to Customize the Agent

To customize the behavior of the agent itself, you pass the `AgentConfiguration` object to the `Instrumentation.start` method. The `AgentConfiguration` object allows you to do these things:

- Point to an on-premises EUM Server
- Enable logging
- Custom set the application name, useful if you deploy essentially the same app binary with different package names to different geographic areas. This ensures that all the data ends up being processed under the same name.
- Ignore HTTP requests internal to your application that are not used for network requests
- Configure the agent to use your custom HTTP library to send its beacons

The syntax looks like the following:
Instrumentation.start(AgentConfiguration.builder()
    .withAppKey("<EUM_APP_KEY>")
    .withContext(getApplicationContext())
    .withCollectorURL(collectorURL*) // The URL of the EUM Server(on-prem)
    .withCompileTimeInstrumentationCheck(true) // Set to false if you are using features of the SDK only, like custom HTTP support, but not to instrument your app.
    .withLoggingEnabled(true)//set default INFO logging. Tagged "AppDynamics".
    .withApplicationName(applicationName)//set a custom app name
    .withExcludedUrlPatterns(excludedUrlPatterns) // Set excluded url regex patterns for http tracking
    .withCollectorChannelFactory(collectorChannelFactory()) // The custom HTTP implementation to use
    .build());

* The default URL of the EUM server is https://mobile.eum-appdynamics.com:443. For a list of URLs of EUM servers, see External Access Locations.

See the latest JavaDocs for more information.

**Configure the Agent to Use Custom HTTP Library**

The Android Agent uses HTTP to deliver its beacons. To have the agent use your custom HTTP library for this purpose, do the following.

1. Implement a class that extends the following abstract class:

```java
public abstract class CollectorChannel {
    private URL url;
    private int connectTimeout;
    private int readTimeout;
    private Map<String, List<String>> requestProperties = new HashMap<String, List<String>>(){}
    private String requestMethod;

    public void setURL(URL url) {
        this.url = url;
    }

    public URL getURL() {
        return url;
    }

    public void setConnectTimeout(int connectTimeout) {
        this.connectTimeout = connectTimeout;
    }
```
public int getConnectTimeout() {
    return connectTimeout;
}

public void setReadTimeout(int readTimeout) {
    this.readTimeout = readTimeout;
}

public int getReadTimeout() {
    return readTimeout;
}

public void addRequestProperty(String property, String value) {
    if (!requestProperties.containsKey(property)) {
        requestProperties.put(property, new ArrayList<String>());
    }
    requestProperties.get(property).add(value);
}

public Map<String, List<String>> getRequestProperties() {
    return Collections.unmodifiableMap(requestProperties);
}

public void setRequestMethod(String requestMethod) {
    this.requestMethod = requestMethod;
}

public String getRequestMethod() {
    return requestMethod;
}

public abstract OutputStream getOutputStream() throws IOException;

public abstract InputStream getInputStream() throws IOException;

public abstract int getResponseCode() throws IOException;
public abstract Map<String, List<String>> getHeaderFields() throws IOException;
}

This interface is loosely based on HttpURLConnection.

2. Implement a version of the CollectorChannelFactory interface, which looks like this:

```java
public interface CollectorChannelFactory {
    /**
     * Returns a new instance of CollectorChannel.
     * If you want to supply a custom CollectorChannel, implement this interface, and return
     * an instance of your concrete implementation of CollectorChannel from this method.
     */
    public CollectorChannel newCollectorChannel();
}
```

The implementation of `newCollectorChannel` should return a new instance of your implementation of `CollectorChannel`.

3. Pass the `CollectorChannelFactory` to the `AgentConfiguration` object.

Capture User Interactions

You can enable the Android Agent to track certain user interactions. Once user interactions have been captured, you can sort sessions by UI event and view the UI event in the timeline of the session waterfall.

You can capture when users do one or all of the following:

- press buttons
- select a text field
- click on a list item

Security and Privacy Concerns

The interaction capture mode is disabled by default for security and privacy reasons as user interactions may contain sensitive information. Moreover, this potential security and privacy issue may be compounded if you enable both the capturing of UI interactions and screenshots.

Enable User Interaction Capture Mode

To enable user interaction capture mode, pass the capture mode to the method `withInteractionCaptureMode()` from an `AgentConfiguration` object. The instrumentation code example below configures the Android Agent to capture all the supported types of user interactions.
You can also configure the Android Agent to only capture one type of user interaction:

```java
Instrumentation.start(AgentConfiguration.builder()
    .withAppKey("<EUM_APP_KEY>")
    .withContext(getApplicationContext())
    .withInteractionCaptureMode(InteractionCaptureMode.ButtonPressed)
    .build());
```

**Configure and Take Screenshots**

Mobile screenshots are enabled by default in the Android Agent. You can configure the Controller UI to automatically take screenshots or use the Android SDK to manually take a screenshot as shown below:

```java
Instrumentation.takeScreenshot();
```

For example, you might want to take a screenshot after you load a UI element to view how it's displayed to customers:

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_spinner_with_toast);
    spinner = (Spinner) findViewById(R.id.spnOptions);
    btnSpinnerVal = (Button) findViewById(R.id.btnSpinnerValue);
    loadSpinner();
    Instrumentation.takeScreenshot();
}
```

**Disable Screenshots**
You can disable screenshots from the Controller UI or with the Android SDK. To disable screenshots with the Android SDK, use the method `withScreenshotsEnabled(false)` from the `AgentConfiguration` class as shown below.

```java
Instrumentation.start(AgentConfiguration.builder()
    .withAppKey("<EUM_APP_KEY>")
    .withContext(getApplicationContext())
    .withScreenshotsEnabled(false)
    .build());
```

### Block/Unblock Screenshots

You can also use the Android SDK to block screenshots from being taken during the execution of code block. This just temporarily blocks screenshots from being taken until you unblock screenshots. This enables you to stop taking screenshots in situations where users are entering personal data, such as on login and account screens.

You use the methods `Instrumentation.blockScreenshots()` and `Instrumentation.unblockScreenshots()` to block and unblock screenshots. If screenshots are disabled through `AgentConfiguration.Builder.withScreenshotsEnabled(true)` or through the Controller UI, these methods have no effect. You can call `Instrumentation.screenshotsBlocked()` to check if screenshots are being blocked.

### Transform URLs for Network Requests

When your application makes network requests, you may not want to report URLs containing sensitive information to the EUM Server. You can instead transform the network request URL before reporting it or ignore it altogether.

To do so:

1. Implement a network request callback that modifies or ignores specific URLs.
2. Register the network request callback in the initialization code.

### Implement the Network Request Callback

The callback that modifies or ignore specific URLs is an implementation of the interface below. The method `onNetworkRequest` is synchronous, so it is recommended that you return from the function quickly.

```java
public interface com.appdynamics.eumagent.runtime.NetworkRequestCallback
{
    boolean onNetworkRequest(HttpRequestTracker httpRequestTracker);
}
```

### Transforming URLs

The `onNetworkRequest` method, in general, should follow the steps below to transform URLs:

1. Identify specific URLs using techniques such as regex or pattern matching.
2. Modify the `url` property of the `HttpRequestTracker` object.
3. Assign a valid URL to the `url` property. (Modifying other properties of the `HttpRequestTracker` object will be ignored.)
4. Return `true`. 
The first step is optional as you could choose to transform the URLs of all network requests.

```java
private static class myNetworkRequestCallback implements com.appdynamics.eumagent.runtime.NetworkRequestCallback {
    @Override
    public boolean onNetworkRequest(HttpRequestTracker httpRequestTracker) {
        URL urlMask = new URL("http://networkrequest-mask.com");
        httpRequestTracker.withURL(urlMask);
        return true;
    }
}
```

In general, however, you would want to identify and transform URLs that contain sensitive information as implied in the example below.

```java
private static class myNetworkRequestCallback implements com.appdynamics.eumagent.runtime.NetworkRequestCallback {
    @Override
    public boolean onNetworkRequest(HttpRequestTracker httpRequestTracker) {
        String urlString = httpRequestTracker.getURL().toString();
        try {
            URL url = new URL("http://customer-account.com");
            if (urlString.contains("accountInfo")) {
                // Change the URL for calls to Facebook
                httpRequestTracker.withURL(url);
                return true;
            }
        } catch (MalformedURLException e) {
            return false;
        }
        return true;
    }
}
```

Ignoring URLs

If the `onNetworkRequest` method returns `false`, the beacon is dropped. The general process for ignoring beacons is as follows:

1. Identify specific URLs using techniques such as regex or pattern matching.
2. Return `false`.

You could theoretically ignore all network requests with the following implementation of `onNetworkRequest`.
In general, you would identify network requests that you didn't want to monitor and return `false` to ignore the network request as implied by this example.

```
private static class myNetworkRequestCallback implements com.appdynamics.eumagent.runtime.NetworkRequestCallback {
    @Override
    public boolean onNetworkRequest(HttpRequestTracker httpRequestTracker) {
        return false;
    }
}
```

---

**Register the Network Request Callback**

After implementing the callback, you register it in the initialization code as shown below. When the Android Agent is ready to create a network request beacon, it will first call the callback with an `HttpRequestTracker` object.

```
private static class myNetworkRequestCallback implements com.appdynamics.eumagent.runtime.NetworkRequestCallback {
    @Override
    public boolean onNetworkRequest(HttpRequestTracker httpRequestTracker) {
        String urlString = httpRequestTracker.getURL().toString();
        try {
            URL url = new URL("http://socialnetworksite.com");
            if (urlString.contains("avatar")) {
                // Ignore calls for avatars
                return false;
            }
        } catch (MalformedURLException e) {
            return false;
        }
        return true;
    }
}
```
Enable Logging and Set Logging Level

You use the method `withLoggingLevel` of the class `AgentConfiguration` to enable logging and set the logging level. You can set logging to one of the following levels:

- `LOGGING_LEVEL_NONE`
- `LOGGING_LEVEL_INFO`
- `LOGGING_LEVEL_VERBOSE`

Use verbose logging only for troubleshooting and be sure to disable for production.

Example:

```java
AgentConfiguration config = AgentConfiguration.builder()
    .withAppKey(appKey)
    .withContext(context)
    .withLoggingLevel(Instrumentation.LOGGING_LEVEL_VERBOSE)
    .build();
Instrumentation.start(config);
```

Android SDK Documentation

For the complete SDK API documentation, see the latest JavaDocs or the previous versions listed below:

- https://docs.appdynamics.com/javadocs/android-sdk/4.5/4.5.0/
- https://docs.appdynamics.com/javadocs/android-sdk/4.5/4.5.1/
- https://docs.appdynamics.com/javadocs/android-sdk/4.5/4.5.2/
- https://docs.appdynamics.com/javadocs/android-sdk/4.5/4.5.4/
- https://docs.appdynamics.com/javadocs/android-sdk/4.5/4.5.5/
- https://docs.appdynamics.com/javadocs/android-sdk/5/5.0/
- https://docs.appdynamics.com/javadocs/android-sdk/5/5.1/
- https://docs.appdynamics.com/javadocs/android-sdk/5/5.2/

After 4.5.6, the Android Agent started using a version number different from that of the Controller and the other AppDynamics platform components. See Mobile Agent Version and Deployment Support Matrix for the minimum version of the Controller and the EUM Server required for complete support of all the Android Agent features.
Manually Upload Mapping Files

On this page:
- Requirements
- Upload Methods
- Upload with the Controller UI
- Upload with the API

Related pages:
- Automatically Upload Mapping Files

If you have obfuscated your code and want to monitor crashes, you will need to upload the ProGuard or DexGuard mapping files. AppDynamics needs the mapping file for the application to produce human-readable stack traces for crash snapshots. For details about why you should do this, see Get Human-Readable Crash Snapshots.

DexGuard is built on ProGuard, so the Controller UI and some of the REST API for uploading mapping files will refer to ProGuard for both ProGuard and DexGuard files.

The sections below discuss the requirements, the available methods, and instructions for manually uploading the mapping files.

It is highly recommended to set up automatic uploading of the ProGuard or DexGuard mapping files using Gradle and your build.gradle file.

Requirements

To manually upload the mapping files, you need to associate the mapping file with the correct version of the application by providing the following:
- the package name of the Android package for the application
- the version code for that application specified in either the AndroidManifest.xml file or the build.gradle file

Upload Methods

You can then either upload the mapping file using the Upload missing ProGuard Mappings dialog in the Controller UI or use a special REST API. Perform the upload separately for each ProGuard mapping file that you are providing. You can also check your uploads with a separate REST API.

If you update your application, you need to upload the new version of the mapping file.

Upload with the Controller UI

1. From the Mobile App menu, click Configuration.
2. Click ProGuard Mappings >.
3. From the Upload missing ProGuard Mapping dialog:
   a. Enter the version code (a number) for the package. This is the versionCode property specified in either the AndroidManifest.xml file or the build.gradle file of the application for which this mapping file was generated.
   b. Click Choose File.
      The uploader expects a file with the .txt extension. The file is named mapping.txt.
   c. In the file browser, locate and select the mapping file and click Open.
   d. Click Upload.

Upload with the API

The API uses HTTP basic authentication. The username is your AppDynamics EUM account name and the password is your EUM license key.
Send the mapping file

Send the mapping file as a text file in the body of the PUT request to the following URL:

https://api.eum-appdynamics.com/v2/account/<MyAccountName>/<packageName>/<versionCode>/mapping-file

These parameters are required:

- **MyAccountName**: the URL-encoded version of your account name.
- **packageName**: the name of the Android package for which this mapping file was generated.
- **versionCode**: the string representation of the "versionCode" property specified in either the AndroidManifest.xml file or the build.gradle file of the application for which this mapping file was generated.

The request body contains the mapping file. You need to add a `Content-Type` header, `-H Content-Type:text/plain`, and your account name and license key/password to the call.

Sample Request and Response Using the REST API

This is a sample request and response using the REST API.

**Upload Request**

The following example uses curl to send a ProGuard mapping file. You would use a similar command to send a DexGuard mapping file. The account name is "Example account" and the license key/password is "Example-License-Key-4e8ec2ae6cfe". The plus signs replace spaces in the account name when the account name is URL-encoded. The package name of the Android application is "com.example.networklogger". The mapping file corresponds to the version with `versionCode`.

```
curl -v -H Content-Type:text/plain --upload-file mapping.txt --user Example+account:Example-License-Key-4e8ec2ae6cfe
https://api.eum-appdynamics.com/v2/account/Example+account/com.example.networklogger/1/mapping-file
```

**Upload Response**

The successful output of the example request looks like this:
* About to connect() to api.eum-appdynamics.com port 443 (#0)
* Trying ::1...
* connected
* Connected to api.eum-appdynamics.com (::1) port 443 (#0)
* Server auth using Basic with user 'Example+account'
> PUT /v2/account/Example+account/com.example.networklogger/1/proguard-mapping HTTP/1.1
> Authorization: Basic SW50ZXJuYWxrGVzdCthY2NvdW50OlRlc3RBY2N0LTFlMzktNDVkMy05MzAzLTRlOGVjMmF1NmNnZQ==
> User-Agent: curl/7.24.0 (x86_64-apple-darwin12.0) libcurl/7.24.0 OpenSSL/0.9.8y zlib/1.2.5
> Host: api.eum-appdynamics.com
> Accept: */*
> Expect: 100-continue
>
< HTTP/1.1 100 Continue
* We are completely uploaded and fine
< HTTP/1.1 200 OK
< Content-Length: 0
< Server: Jetty(8.1.4.v20120524)
<
* Connection #0 to host api.eum-appdynamics.com left intact
* Closing connection #0

**Check Uploaded Mapping Files Using the REST API**

You can check to make sure that your mapping files have successfully uploaded using two REST APIs.

1. Get a list of the GUIDs for the last 50 mapping files you have uploaded.
2. Check if a specific mapping file has been uploaded.

**List of the last 50 mapping files uploaded**

The `proguardQuery` method allows you to retrieve a list of GUIDs for up to the last 50 mapping files (ProGuard or DexGuard) that have been uploaded to your account, along with the time they were uploaded. The response is displayed as JSON, by upload time, with the most recent first.

1. Set up your authentication as described in Upload with the API.
2. Create a GET request of the form:

```
curl --user Example+account:Example-License-Key-4e8ec2ae6cfe https://<EUM_Cloud/EUM_Server_Host:port>/v2/account/<EUM_Account_Name>/crash-symbol-file-query/mapping-file
```
where the value for --user is the authentication string you created in step 1, EUM_Cloud refers to api.eum-appdynamics.com:443 for SaaS-based EUM Cloud accounts or EUM_Server_Host, which refers to the URL where your on-premises EUM Server is hosted, and EUM_Account_Name is your EUM account name.

Sample Request

```
curl --user Example+account:Example-License-Key-4e8ec2ae6cfe
```

Sample Response

```
{"proguardFiles": [
{"packageName":"my_package_name1", "version":"1",
"uploadTime":"mm/dd/yyyy 16:09:23","GUID":"my_build_id1"},
{"packageName":"my_package_name2", "version":"1",
"uploadTime":"mm/dd/yyyy 16:09:23","GUID":"my_build_id2"}
]}
```

Check for specific mapping file by GUID

The checkForProguardFile method allows you to check if a specific ProGuard or DexGuard file by GUID has been uploaded. The upload time is returned in the response.

a. Set up your authentication as described in Upload with the REST API.
b. Create a GET request of the form:

```
curl --user
Example+account:Example-License-Key-4e8ec2ae6cfe
https://<EUM_Cloud/EUM_Server_Host:port>/v2/account/<EUM_Account_Name>/crash-symbol-file-query/mapping-file/guid/<GUID_To_Check>
```

where the value for --user is the authentication string you created in step 1, EUM_Cloud refers to api.eum-appdynamics.com:443 for SaaS-based EUM Cloud accounts or EUM_Server_Host refers to the URL pointing to your on-premises EUM Server, EUM_Account_Name is your EUM account name and GUID_To_Check is the GUID of the ProGuard file in which you are interested.

Sample Request
curl --user
Example+account:Example-License-Key-4e8ec2ae6cfe

Sample Response

{"packageName":"mypackagename","version":"1","uploadTime":"mm/dd/yyyy 16:09:23","GUID":"my_build_id1"}
Troubleshoot the Android Instrumentation

On this page:
- Dex Failure After Upgrading Agent
- Instrumentation Appears to Be Running After Being Disabled
- Crashes Are Not Reported After Using Custom Default Uncaught Exception Handler
- Network Requests Are Not Being Reported
- Exclude Classes from Being Instrumented
- Enforce a Different Runtime Version from the Plugin Version

Related pages:
- Instrument Android Applications
- Customize the Android Build
- Customize the Android Instrumentation

The following sections provide instructions and tips for solving some common Android instrumentation issues.

**Dex Failure After Upgrading Agent**

If you build your application with Gradle and you get a dex failure after upgrading the Mobile Agent, you likely need to clear the Gradle cache.

**Using Android Studio**

To clear the cache using Android Studio:

1. Select **Build > Clean Project.**
2. Select **File > Invalidate Caches.**

Sometimes a restart is also necessary.

**Using the Command Line**

To clear the cache using the command line:

1. Stop Gradle.

```
$ gradlew --stop
```

2. You may also need to refresh dependencies.

```
$ gradlew --refresh-dependencies
```

You can also try removing the AppDynamics cached images.

1. Find the images.
2. Delete them.

Instrumentation Appears to Be Running After Being Disabled

If you disabled the instrumentation in Android using the Gradle flag `enabledForDebugBuilds`, but the instrumentation task still runs, this is because of the Transform API.

Android Gradle Plugin 1.5.0 introduced the Transform API that the Android Agent uses to do the bytecode injection. Due to the limitations of the Transform API, even when instrumentation is disabled, the `transformClassesWithAppDynamicsForDebug` task will still appear in the Gradle task log. To ensure that it is not actually instrumenting, run `gradle --info` flag to show the info logs.

You should see the following log:

```
Instrumentation is disabled for this build variant. Just copying the input files to fulfill Transform contract.
```

Crashes Are Not Reported After Using Custom Default Uncaused Exception Handler

Install your exception handler before you start the Android Agent. When an uncaught exception is handled by the agent, it will first store the crash report, and then call your exception handler.

Network Requests Are Not Being Reported

Please see the list of supported networking libraries. If your library is not on the list, you can always report these network requests manually.

Exclude Classes from Being Instrumented

```
- excludeClasses
```

Enforce a Different Runtime Version from the Plugin Version

```
- dependencies.compile 'com.appdynamics:appdynamics-runtime:4.5.+'
  adeum.dependencyInjection.enabled = false
```
Instrument Xamarin Applications

On this page:

- Supported Platforms
- Limitations

Related pages:

- Instrument a Xamarin Instrumentation
- Customize the Xamarin Instrumentation

Before you can monitor your Xamarin application, you will need to instrument your application to enable the Xamarin Agent to collect mobile metrics.

After you have set up and accessed Mobile RUM, follow the instructions below:

1. Confirm that the Xamarin Agent supports your platform
2. Understand the limitations of the Xamarin Agent
3. Instrument a Xamarin Application
4. Customize the Xamarin Instrumentation (Optional)

Supported Platforms

The Xamarin Agent can only be used with iOS and Android platforms. All other platforms will build and run without errors, but no monitoring will occur.

Limitations

The Xamarin Agent has the following limitations:

- Automatic instrumentation is not supported, so you will need to manually report events and metrics.
- Symbolication is not supported, although the Xamarin Agent does report uncaught exceptions and native application crashes.
Instrument a Xamarin Application

Follow the steps below to manually instrument your Xamarin iOS, Android, and Forms apps.

1. Add the Xamarin Agent Package
2. Get Your Application Key
3. Add the Required Permissions (Android Deployments Only)
4. Link the AppDynamics Agent Assembly
5. Initialize the Agent
6. Build the Application
7. Customize Your Instrumentation (Optional)
8. Point to an On-Premises EUM Server (Optional)

Add the Xamarin Agent Package

You obtain the Xamarin Agent from the NuGet Gallery. Follow the instructions given in Adding a Package to add the package AppDynamics Xamarin Agent from nuget.org.

Get Your Application Key

Complete the Getting Started Wizard to get an EUM App Key. You will need this key when you modify the source code. In some cases, multiple mobile applications can share the same key.

Because there is no Xamarin platform option, you will need to choose either Android or iOS. For Android, you will need to select Manual.

If you have completed the Getting Started Wizard, but don’t have your EUM App Key, see Get Your Application Key.

Add the Required Permissions (Android Deployments Only)

Open the file Properties/AndroidManifest.xml and verify that it has these permissions:

```xml
<uses-permission
android:name="android.permission.INTERNET"></uses-permission>
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"></uses-permission>
```

If these permissions are not present, add them.

Link the AppDynamics Agent Assembly

Forms Solution

Xamarin Forms is the unified development environment for running all platforms. If you try to run something on an unsupported platform, linking our agent won’t allow you to monitor the application, but it also will not cause any errors.

To use the Xamarin Agent for your iOS/Android apps, add the using directive at the top of the App.xaml.cs file:
using Xamarin.Forms;
using AppDynamics.Agent;

namespace <AppName>
{
    public partial class App : Application
    {
        ...
    }
}

**iOS Solution**

To use the Xamarin Agent in iOS apps, add the using directive at the top of the AppDelegate.cs file:

using Foundation;
using UIKit;
using AppDynamics.Agent;

public class AppDelegate : UIApplicationDelegate
{
    ...
}

**Android Solution**

To use the Xamarin Agent in Android apps, add the using directive at the top of the MainActivity.cs file:
using Android.App;
using Android.Widget;
using Android.OS;
using System;
using Android.Content;
using AppDynamics.Agent;

namespace <AppName>
{
    [Activity(Label = "Phoneword", MainLauncher = true, Icon = 
    "@mipmap/icon")]
    public class MainActivity : Activity
    {
        ...
    }
    ...
}

Initialize the Agent

To initialize the Xamarin Agent, you use the code below for iOS and Android. Use the EUM app key (enter as a string) that you received after completing step 2.

```csharp
var config = AppDynamics.Agent.AgentConfiguration.Create(<EUM_APP_KEY>);
AppDynamics.Agent.Instrumentation.InitWithConfiguration(config);
```

If you are running an on-premises EUM Server, you need to specify the URL to the EUM Server. See Point to an On-Premises EUM Server (Optional) to learn how.

Forms Solution

For Forms Solutions, you only need to place the initialize code in the constructor of the App.xaml.cs file for the Xamarin Agent to instrument both Android and iOS applications.
public App()
{
    InitializeComponent();
    // This initialization code is used by both iOS and Android apps.
    var config = AppDynamics.Agent.AgentConfiguration.Create(<EUM_APP_KEY>);
    AppDynamics.Agent.Instrumentation.InitWithConfiguration(config);
    MainPage = new FormsExamplePage();
}

If you have application code in the MainActivity.cs file for Android or AppDelegate.cs for iOS that you want to instrument, however, you should initialize the Xamarin Agent in those files as you would do for iOS Solutions and Android Solutions.

**iOS Solution**

For iOS apps, you place the initialize code in the AppDelegate.cs file in the method FinishedLaunching of the class AppDelegate as shown below.

```csharp
public class AppDelegate : UIApplicationDelegate
{
    // class-level declarations
    public override UIWindow Window
    {
        get;
        set;
    }
    public override bool FinishedLaunching(UIApplication application,
        NSDictionary launchOptions)
    {
        // The two lines below initialize the AppDynamics instrumentation.
        var config = AppDynamics.Agent.AgentConfiguration.Create(<EUM_APP_KEY>);
        AppDynamics.Agent.Instrumentation.InitWithConfiguration(config);
        ...
        return true;
    }
    ...
}
```

You may also consider placing it in the Main.cs int the method Main.

**Android Solution**

In the MainActivity.cs file, place the initialization code in the method OnCreate.
class MainActivity {
    protected override void OnCreate(Bundle savedInstanceState) {
        // The two lines below initialize the AppDynamics instrumentation.
        var config = AppDynamics.Agent.AgentConfiguration.Create(<EUM_APP_KEY>);
        AppDynamics.Agent.Instrumentation.InitWithConfiguration(config);
        ...
    }
}

Build the Application

Run and build your application from Visual Studio. From the Getting Started Wizard, you should see that the application has connected and the instrumentation has been verified.

Customize Your Instrumentation (Optional)

The Xamarin SDK has additional classes to allow you to extend the kinds of application data you can collect and aggregate using Mobile RUM. For more information, see Customize the Xamarin Instrumentation.

Point to an On-Premises EUM Server (Optional)

To use an on-premises EUM Server, you pass the URL to the on-premises EUM Server when you initialize the instrumentation with the EUM App Key from Get Your Application Key:

var config = AppDynamics.Agent.AgentConfiguration.Create(<EUM_APP_KEY>);
config.CollectorURL = <COLLECTOR_URL:PORT>;
AppDynamics.Agent.Instrumentation.InitWithConfiguration(config);

Upgrade the Xamarin Agent

As new features are added to the agent, you will need to upgrade the Xamarin Agent package in your app.

1. From Visual Studio, open the Xamarin application that has the AppDynamics Agent package.
2. From the Packages folder, select the AppDynamics Agent.
3. Right-click and click Update.
Customize the Xamarin Instrumentation

The following sections show you how to use the Xamarin SDK to customize your instrumentation.

**Track Calls**

You can instrument methods to see how often the instrumented method is invoked and how long it takes to run. To do this, add a call at the beginning and end of the method you’d like to instrument.

In the example below, the code executed in the constructor for the class `MyClass` will be tracked and reported. In your own code, start tracking calls by specifying the class and method in `BeginCall` and then complete the tracking and report the data by calling `ReportCallEnded`.

```csharp
using AppDynamics.Agent;
...
public class MyClass {
    public MyClass() {
        var tracker = Instrumentation.BeginCall("MyClass", "Constructor");
        // The code placed here will be tracked and reported.
        tracker.ReportCallEnded();
    }
}
```

**Timing Events**

Sometimes you want to time an event in your application that spans multiple methods. You can do this by calling `StartTimerWithName` when the event starts, and then `StopTimerWithName` when it ends. For example, to track the time a user spends viewing a screen, the instrumentation might look something like the following:
using AppDynamics.Agent;
...
async private void StartCapturePreview_Click(object sender,
RoutedEventArgs e) {
    capturePreview.Source = captureManager;
    Instrumentation.StartTimerWithName("CapturePreview");
    await captureManager.StartPreviewAsync();
}
async private void StopCapturePreview_Click(object sender,
RoutedEventArgs e) {
    await captureManager.StopPreviewAsync();
    Instrumentation.StopTimerWithName("CapturePreview");
}

Report Metrics

To report other types of data, you can use a metric. The metric name should only contain alphanumeric characters and spaces. Illegal characters are replaced by their ASCII hex value. The metric value must be a long integer.

The snippet below shows how you might report a metric.

using AppDynamics.Agent;
...
Instrumentation_ReportMetricWithName("Database Rows", 5123);

HTTP Requests

You can report a Network Request using the AppDynamics.Agent.HTTPRequestTracker class.

The following is an example of using HttpRequestTracker with the System.Net.Http.HttpClient class. The tracker object synchronously captures and reports the network request as well as any network errors.
using AppDynamics.Agent;
...
public async Task<string> Fetch(Uri uri) {
    var client = new HttpClient();
    // Create AppDynamics Tracker
    var tracker = HTTPRequestTracker.Create(uri);
    // Add AppDynamics Server Correlation Headers
    foreach (var header in ServerCorrelationHeaders.Generate) {
        // Each header could have multiple values
        foreach (var value in header.Value) {
            client.DefaultRequestHeaders.Add(header.Key, value);
        }
    }
    HttpResponseMessage response = null;
    try {
        response = await client.GetAsync(uri);
    } catch (Exception ex) {
        // Capture any network errors.
        tracker.Exception = ex;
        tracker.ReportDone();
        throw ex; //you decide to throw it or not
    }
    if (!response.Equals(null)) {
        // Capture request information such as the
        // status code, status message, and headers.
        tracker.StatusLine = response.ReasonPhrase;
        tracker.ResponseHeaderFields = response.Headers;
        tracker.ReportDone();
        return await response.Content.ReadAsStringAsync();
    }
    return null;
}

Leave Breadcrumbs

You can leave breadcrumbs to mark interesting events. For example, if your application crashes, the breadcrumbs you left will be displayed in the crash report and could provide context. You can also configure the breadcrumb to appear in sessions.

The following is the method signature for leaving breadcrumbs:
static void AppDynamics.Agent.Instrumentation.LeaveBreadcrumb(string breadcrumb, BreadcrumbVisibility mode)

You use the mode to set the visibility of the breadcrumb. The visibility defines where you will see the breadcrumb in the Controller UI. The value of mode can be one of the following:

- BreadcrumbVisibility.CrashesOnly – The breadcrumb will only appear in crash snapshots.

Thus, you would use the method below to set breadcrumbs that are only reported in crash reports:

```csharp
using AppDynamics.Agent;
...
Instrumentation.LeaveBreadcrumb("GetUserInfo", BreadcrumbVisibility.CrashesOnly);
```

If you would like to see the breadcrumb in crash reports and sessions:

```csharp
using AppDynamics.Agent;
...
Instrumentation.LeaveBreadcrumb("GetUserInfo", BreadcrumbVisibility.CrashesAndSessions);
```

**Report Errors and Exceptions**

You can report exceptions using the method reportError from the Instrumentation class. Reported exceptions will appear in session details.

You can also set one of the severity levels below for an issue. With the severity level, you can filter errors in the Code Issues Dashboard or Code Issues Analyze.

- ErrorSeverityLevel.INFO
- ErrorSeverityLevel.WARNING
- ErrorSeverityLevel.CRITICAL

The example below uses the API to report possible exceptions and sets the severity level to ErrorSeverityLevel.CRITICAL (critical) when writing to a file
using AppDynamics.Agent;
...
try {
    // possible exception //
}
catch (Exception e){
    Instrumentation.ReportError(exception, ErrorSeverityLevel.CRITICAL);
}

Programmatically Control Sessions

By default, a mobile session ends after a period of user inactivity. For example, when a user opens your application, the session begins and only ends after the user stops using the app for a set period of time. When the user begins to use the application again, a new session begins.

Instead of having a period of inactivity to define the duration of a session, however, you can use the following API to programmatically control when sessions begin and end:

```csharp
static void AppDynamics.Agent.Instrumentation.StartNextSession()
```

When you call the method `StartNextSession`, the current session ends and a new session begins. The API enables you to define and frame your sessions so that they align more closely with business goals and expected user flows. For example, you could use the API to define a session that tracks a purchase of a product or registers a new user.

Excessive use of this API will cause sessions to be throttled (excessive use is >10 calls per minute per Xamarin Agent, but is subject to change). When not using the API, sessions will fall back to the default of ending after a period of user inactivity.

**Example of a Programmatically Controlled Session**

In the example below, the current session ends and a new one begins when an item is bought.
using AppDynamics.Agent;
...
public async Task BuySaleItemAsync(SaleItem item)
{
    try
    {
        bool buySucceeded = await this.MobileService.InvokeApiAsync<SaleItem, bool>("buy", item);
        if (buySucceeded)
        {
            await UserDialogs.Instance.AlertAsync("Thanks for buying this item");
            Instrumentation.StartNextSession();
        }
    }
    catch (Exception e)
    {
        Debug.WriteLine(0"Unexpected error {0}", e.Message);
    }
}

Start and End Session Frames

You can use the ISessionFrame API to create session frames that will appear in the session activity. Session frames provide context for what the user is doing during a session. With the API, you can improve the names of user screens and chronicle user flows within a business context.

Use Cases

The following are common use cases for the ISessionFrame API:

- One screen performs multiple functions and you want more granular tracking of the individual functions.
- A user flow spans multiple screens or user interactions. For example, you could use the API to create the session frames "Login", "Product Selection", and "Purchase" to chronicle the user flow for purchases.
- You want to capture dynamic information based on user interactions to name session frames, such as an order ID.

ISessionFrame API

The table below lists the two methods and one property you can use with session frames. In short, you start a session frame with StartSessionFrame and then use the returned ISessionFrame object to rename and end the session frame.

<table>
<thead>
<tr>
<th>Class</th>
<th>Method/Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Instrument

**Method:**

```csharp
static ISessionFrame StartSessionFrame(string sessionFrameName)
```

Use this to start and name session frames. Naming session frames enables you to easily identify and track the frames in the Sessions Details dialog.

**Property:**

```csharp
string Name
```

Rename the session frame name. You assign the updated session frame name with this property from the `ISessionFrame` object returned from `StartSessionFrame`.

**Method:**

```csharp
static void End()
```

End the session frame. You call this method from the `ISessionFrame` object returned from `StartSessionFrame`.

---

**Session Frame Example**

In the following example, the `ISessionFrame` API is used to track user activity during the checkout process.
using AppDynamics.Agent;
...

namespace ShoppingApp {
    public partial class ShoppingCart : ContentPage {
        private ISessionFrame sessionFrame;
        private string orderId;
        ...
        void checkoutCartButtonClicked(object sender, EventArgs e) {
            // The checkout starts when the user clicks the checkout button.
            // This may be after they have updated quantities of items in the
            sessionFrame = Instrumentation.StartSessionFrame("Checkout");
        }
        void confirmOrderButtonClicked(object sender, EventArgs e) {
            // Once they have confirmed payment info and shipping information
            // are clicking the "Confirm" button to start the backend process
            // we may know more information about the order itself, such as a
            sessionFrame.Name = "$Checkout: Order ID {this.orderId}";
        }
        void processOrderCompleted(object sender, EventArgs e) {
            // Once the order is processed, the user is done "checking out" s-
            // the session frame.
            sessionFrame.End();
        }
        void checkoutCancelled(object sender, EventArgs e) {
            // If they cancel or go back, you'll want to end the session fram-
            // it will be left open and appear to have never ended.
            sessionFrame.End();
        }
    }
}

Add User Data

You can set a key/value pair of strings to record important events or information. Below is the method signature for setting user data:

static void AppDynamics.Agent.Instrumentation.SetUserData(string key, string value)

For example, you might want to log the user ID when the method for logging in the user is called:
using AppDynamics.Agent;

... void LogInUser(UserCredentials) {
    // Log in user
    ...
    // Set user data with the user name.
    Instrumentation.SetUserData("user_id", UserCredentials.ID);
}

This information is available in **Network Request Analyze** and is added to any crash snapshots that may be taken. Keys and values are limited to 2048 characters each.

You can also set user data with values of other types (long, boolean, double, DateTime) using the following methods:

- SetUserDataBoolean
- SetUserDataDate
- SetUserDataDouble
- SetUserDataLong

To remove user data, use the following methods:

- RemoveUserData
- RemoveUserDataBoolean
- RemoveUserDataDate
- RemoveUserDataDouble
- RemoveUserDataLong

**Xamarin SDK Documentation**

For the complete SDK API documentation, see the latest Xamarin SDK documentation or the previous versions listed below:

- [https://docs.appdynamics.com/javadocs/xamarin-sdk/4.5/4.5.0/html/](https://docs.appdynamics.com/javadocs/xamarin-sdk/4.5/4.5.0/html/)
- [https://docs.appdynamics.com/javadocs/xamarin-sdk/4.5/4.5.1/html/](https://docs.appdynamics.com/javadocs/xamarin-sdk/4.5/4.5.1/html/)
- [https://docs.appdynamics.com/javadocs/xamarin-sdk/4.5/4.5.2/html/](https://docs.appdynamics.com/javadocs/xamarin-sdk/4.5/4.5.2/html/)
- [https://docs.appdynamics.com/javadocs/xamarin-sdk/4.5/4.5.4/html/](https://docs.appdynamics.com/javadocs/xamarin-sdk/4.5/4.5.4/html/)
- [https://docs.appdynamics.com/javadocs/xamarin-sdk/4.5/4.5.5/html/](https://docs.appdynamics.com/javadocs/xamarin-sdk/4.5/4.5.5/html/)
- [https://docs.appdynamics.com/javadocs/xamarin-sdk/4.5/4.5.6/html/](https://docs.appdynamics.com/javadocs/xamarin-sdk/4.5/4.5.6/html/)
- [https://docs.appdynamics.com/javadocs/xamarin-sdk/50/50.0/html/](https://docs.appdynamics.com/javadocs/xamarin-sdk/50/50.0/html/)
- [https://docs.appdynamics.com/javadocs/xamarin-sdk/50/50.1/html/](https://docs.appdynamics.com/javadocs/xamarin-sdk/50/50.1/html/)
- [https://docs.appdynamics.com/javadocs/xamarin-sdk/50/50.2/html/](https://docs.appdynamics.com/javadocs/xamarin-sdk/50/50.2/html/)

After 4.5.6, the Xamarin Agent started using a version number different from that of the Controller and the other AppDynamics platform components. See **Mobile Agent Version and Deployment Support Matrix** for the minimum version of the Controller and the EUM Server required for complete support of all the Xamarin Agent features.
Instrument Cordova Applications

Follow the steps below to get your EUM App Key and instrument your Cordova-based apps.

1. Check Requirements
2. Get Your Application Key
3. Install and Initialize the Cordova Plugin
4. Customize the Cordova Application Instrumentation

For native mobile applications, following either the iOS Instrumentation or Android Instrumentation instructions. The Mobile Agents will inject the JavaScript Agent and auto-instrument your mobile application including the auto-instrumentation of the UIWebView/WebView.

Check Requirements

To use the AppDynamics Cordova plugin, your hybrid application must be using the following:

- Cordova 8.0.0+
- npm 3.10.10+
- Node.js v6.11.3+

Get Your Application Key

After you completed the Getting Started Wizard, you were given an EUM App Key. You will need this key when you modify the source code. In some cases, multiple mobile applications can share the same key.

If you have completed the Getting Started Wizard, but don't have your EUM App Key, see Get Your Application Key.

Install and Initialize the Cordova Plugin

The AppDynamics Cordova plugin is certified to work with the Ionic and PhoneGap frameworks. Thus, the installation instructions below are for Ionic and PhoneGap applications. You should also be able to use the plugin for other Cordova-based frameworks. See the plugin installation instructions for your Cordova-based framework.

Ionic

From the root directory of your Ionic application, run the following commands to install the plugin and build the mobile app. See the Ionic Cordova plugin documentation for more information.

1. Replace the values in brackets with the configurations for your app and run the command to add the configured plugin. For SaaS deployments:
1. `ionic cordova plugin add appd-plugin-eum-mobile --variable APP_KEY="<#EUM_APP_KEY#>" --variable SCREENSHOTS_ENABLED=<true|false> --variable LOGGING_LEVEL=[0-6] --save`

For on-prem deployments, replace the value for the variable COLLECTOR_URL with the URL to your EUM Server and the value for SCREENSHOT_URL with the URL to your on-prem Screenshot Service.

```
ionic cordova plugin add appd-plugin-eum-mobile --variable APP_KEY="<#EUM_APP_KEY#>" --variable SCREENSHOTS_ENABLED=<true|false> --variable LOGGING_LEVEL=[0-6] --variable COLLECTOR_URL="http(s)://<on-prem_eum-server_url>" --variable SCREENSHOT_URL="http(s)://<on-prem_screenshot_service_url>" --save
```

2. (Optional) You can also set variables to disable user interaction capture mode (iOS/Android) and set the URL to test the network connectivity of Cordova iOS applications.
   a. The interaction capture mode is on by default, but you can disable it by setting the variable INTERACTION_CAPTURE_MODE to false:

```
ionic cordova plugin add appd-plugin-eum-mobile --variable APP_KEY="<#EUM_APP_KEY#>" --variable INTERACTION_CAPTURE_MODE=false --variable SCREENSHOTS_ENABLED=<true|false> --variable LOGGING_LEVEL=[0-6] --variable COLLECTOR_URL="http(s)://<on-prem_eum-server_url>" --variable SCREENSHOT_URL="http(s)://<on-prem_screenshot_service_url>" --save
```

b. By default, the Cordova Plugin will use http://google.com to test the connectivity of Cordova iOS applications. You can, however, use a different URL to test the connectivity by setting the variable REACHABILITY_HOST to the URL. In the example below, the Cordova Plugin is configured to use http://example.com to test the connectivity of the Cordova iOS application.
2. b.

```shell
ionic cordova plugin add appd-plugin-eum-mobile
--variable APP_KEY="<$EUM_APP_KEY$>" --variable
REACHABILITY_HOST=http://example.com --variable
SCREENSHOTS_ENABLED=<true|false> --variable
LOGGING_LEVEL=[0-6] --save
```

3. Build the mobile app:

```shell
ionic cordova build [android, ios]
```

---

PhoneGap

From the root directory of your PhoneGap application, run the commands below to install the plugin. See the PhoneGap plugin documentation for more information.

1. Replace the values in brackets with the configurations for your app and run the command to add the configured plugin.

For SaaS deployments:

```shell
phonegap cordova plugin add appd-plugin-eum-mobile
--variable APP_KEY="<$EUM_APP_KEY$>" --variable
SCREENSHOTS_ENABLED=<true|false> --variable
LOGGING_LEVEL=[0-6] --save
```

For on-prem deployments, replace the value for the variable COLLECTOR_URL with the URL to your EUM Server and the value for SCREENSHOT_URL with the URL to your on-prem Screenshot Service.
1. Add the AppDynamics plugin to your Cordova project:

```bash
phonegap cordova plugin add appd-plugin-eum-mobile
--variable APP_KEY="<#EUM_APP_KEY#>" --variable
SCREENSHOTS_ENABLED=<true|false> --variable
LOGGING_LEVEL=[0-6] --variable
COLLECTOR_URL="http(s)://<on-prem_eum-server_url>" --variable
SCREENSHOT_URL="http(s)://<on-prem_screenshot_service-url>" --save
```

2. (Optional) You can also set variables to enable user interaction capture mode (iOS/Android) and troubleshoot the network connectivity of Cordova iOS applications.
   
a. To view the UI events triggered by user interactions in sessions, set the variable `INTERACTION_CAPTURE_MODE` to `true`:

```bash
phonegap cordova plugin add appd-plugin-eum-mobile
--variable APP_KEY="<#EUM_APP_KEY#>" --variable
INTERACTION_CAPTURE_MODE=true --variable
SCREENSHOTS_ENABLED=<true|false> --variable
LOGGING_LEVEL=[0-6] --save
```

b. To test the network connectivity of Cordova iOS applications, set the variable `REACHABILITY_HOST` to `TBD`:

```bash
phonegap cordova plugin add appd-plugin-eum-mobile
--variable APP_KEY="<#EUM_APP_KEY#>" --variable
REACHABILITY_HOST=TBD --variable
SCREENSHOTS_ENABLED=<true|false> --variable
LOGGING_LEVEL=[0-6] --save
```

3. Build the mobile app:

```bash
phonegap cordova build [android, ios]
```
Customize the Cordova Application Instrumentation

See Customize the Cordova Application showing how to use the Cordova SDK through examples and brief explanations.
Customize the Cordova Instrumentation

On this page:
- Change the App Key
- Collect Additional Types of Data
- Programmatically Control Sessions
- Start and End Session Frames
- Configure and Take Screenshots
- Enable Logging and Set Logging Level

Related pages:
- Instrument Cordova Applications
- Cordova SDK Documentation

Once you have instrumented your Cordova-based application with the Cordova Plugin, you can also use the APIs to customize the data for your app that appears in the Controller UI.

**Using the Cordova Plugin SDK API**

**Syntax**

To call SDK API methods, use the following syntax: `window.plugins.ADEUMMobilePlugin.<method>`

**Arguments**

The last two arguments for all of the SDK API methods should always be two functions. The first function should handle successful cases and the last method should handle failures.

For example:

```javascript
window.plugins.ADEUMMobilePlugin.changeAppKey("<EUM_APP_KEY>",
  (success) => {
    this.showAlert("changeAppKey return: success");
  },
  (error) => {
    this.showAlert("changeAppKey error:" + error);
  });
```

**Add Methods to Call SDK APIs**

To use the SDK APIs, you are recommended to create class methods that call the SDK APIs as shown below.
export class HomePage {

    someMethod(event) {
        window.plugins.ADEUMMobilePlugin.<method>(<arg1>,
        <success_function>, <failure_function>);
    }

    ...
}

Example

Thus, for the HomePage.js file, your HomePage class could have the method `takeScreenshot` that calls the SDK API method `screenshot` as shown here.

export class HomePage {

    takeScreenshot(event) {
        // Call the Cordova plugin SDK methods
        window.plugins.ADEUMMobilePlugin.screenshot(
            (success) => {
                this.showAlert("crash return: success");
            },
            (error) => {
                this.showAlert("crash error:" + error);
            });
    }

    ...
}

Change the App Key

To change the EUM application key, you use the method `changeAppKey` with the parameters below.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appKey</td>
<td>string</td>
<td>The EUM application key.</td>
</tr>
<tr>
<td>success</td>
<td>function</td>
<td>A user-defined function that is called when <code>changeAppKey</code> is successful.</td>
</tr>
<tr>
<td>error</td>
<td>function</td>
<td>A user-defined function that is called when <code>changeAppKey</code> fails.</td>
</tr>
</tbody>
</table>

For example, you could create a new method that takes a new app key and passed it to the SDK API method `changeAppKey`.
```javascript
changeAppKey(event, newAppKey) {
    window.plugins.ADEUMMobilePlugin.changeAppKey(newAppKey,
        (success) => {
            this.showAlert("changeAppKey return: success");
        },
        (error) => {
            this.showAlert("changeAppKey error:" + error);
        }
    );
}
```

Collect Additional Types of Data

You can use methods available in the `ADEUMMobilePlugin` class to collect five additional types of data:

- Info points
- Custom timers
- Custom metrics
- User data
- Breadcrumbs

Info Points

Information points allow you to track how your own code is running. You can see how often a method is invoked, and how long it takes to run by calling the `beginCall` method. When the callbacks `success` or `error` are called, the call to track the info point ends.

```
beginCall(name, functionName, args, success, error)
```

Parameters

The table below describes the parameters for the two methods:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the file or module where the info point is being recorded.</td>
</tr>
<tr>
<td>functionName</td>
<td>string</td>
<td>The function that is invoking <code>beginCall</code> to track an info point.</td>
</tr>
<tr>
<td>success</td>
<td>function</td>
<td>The user-defined callback for successful cases.</td>
</tr>
<tr>
<td>error</td>
<td>function</td>
<td>The user-defined callback for failed cases.</td>
</tr>
</tbody>
</table>

Example

For example, you can use create info points with something like the code below to determine how a method is invoked, and how long it takes to run:
beginCall(event) {
    window.plugins.ADEUMMobilePlugin.beginCall("home.ts",
    "callTrackerFunction", "event",
    (tracker) => {
        tracker.reportCallEndedWithReturnValue("Return from home.ts",
        (success) => {
            console.log("End call with return value
success:" + success);
        },
        (error) => {
            console.log("End call with return value
error:" + error);
        }
    },
    (error) => {
        console.log("Begin call error:" + error);
    }
});
}

Custom Timers

You can create custom timers to time any arbitrary sequence of events within your code, even spanning multiple methods. You create the custom timers using the SDK API methods `startTimer` and `stopTimer`.

- `startTimerWithName(name, success, error)`
- `stopTimerWithName(name, success, error)`

Parameters

The two methods take the following parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the custom timer. Allowed characters are [A-Za-z\s0-9]. Illegal characters are replaced by their ASCII hex value.</td>
</tr>
<tr>
<td>success</td>
<td>function</td>
<td>The user-defined callback for successful cases.</td>
</tr>
<tr>
<td>error</td>
<td>function</td>
<td>The user-defined callback for failed cases.</td>
</tr>
</tbody>
</table>

Example

For example, to track the time a user spends viewing a screen, the instrumentation could look like this:
Custom Metrics

You can also report custom metrics.

You create custom metrics with the `reportMetricWithName`:

```javascript
reportMetricWithName(name, value, success, error)
```

Parameters

The `reportMetricWithName` method takes the following parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the custom metric. The metric names must consist of alphanumeric characters. Illegal characters are replaced by their ASCII hex value.</td>
</tr>
<tr>
<td>value</td>
<td>number</td>
<td>If value is not a whole number an error will be returned.</td>
</tr>
<tr>
<td>success</td>
<td>function</td>
<td>User-defined success callback.</td>
</tr>
<tr>
<td>error</td>
<td>function</td>
<td>User-defined error callback.</td>
</tr>
</tbody>
</table>

Example

For example, the following method could be used to report custom metrics:
```javascript
reportMetric(event, data) {
    window.plugins.ADEUMMobilePlugin.reportMetricWithName(data.name, parseInt(data.value),
        (success) => {
            this.showAlert("reportMetricWithName : success");
        },
        (error) => {
            this.showAlert("reportMetricWithName error:" + error);
        }
    );
}
```

**Breadcrumbs**

Breadcrumbs allow you to situate a crash in the context of your user's experience. Set a breadcrumb when something interesting happens. If your application crashes at some point in the future, the breadcrumb will be displayed along with the crash report. Each crash report displays the most recent 99 breadcrumbs.

You create and leave breadcrumbs with the following SDK API method:

* leaveBreadcrumb(breadcrumb, mode, success, error)

**Parameters**

The method `leaveBreadcrumb` takes the following parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>breadcrumb</td>
<td>string</td>
<td>The string to include in the crash report and sessions. Truncated at 2048 characters; empty values are ignored.</td>
</tr>
<tr>
<td>mode</td>
<td>number</td>
<td>The mode determining where the breadcrumb will be displayed:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0 - for crashes only</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1 - for crashes and sessions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The mode defaults to crashes if the value is not parseable.</td>
</tr>
<tr>
<td>success</td>
<td>function</td>
<td>The user-defined callback for successful cases.</td>
</tr>
<tr>
<td>error</td>
<td>function</td>
<td>The user-defined callback for failed cases.</td>
</tr>
</tbody>
</table>

**Example**

This code example shows how to use the SDK API to leave a breadcrumb:
```javascript
breadcrumb(mode) {
    window.plugins.ADEUMMobilePlugin.leaveBreadcrumb("breadcrumb1", mode,
    (success) => {
        this.showAlert("leaveBreadcrumb return: success");
    },
    (error) => {
        this.showAlert("leaveBreadcrumb error:" + error);
    }
}
```

**Add Custom User Data**

You can set and later remove any string key/value pair you think might be useful with the following methods:

- `setUserData(key, value, success, error)`
- `removeUserData(key, success, error)`

**Parameters**

The following table describes the parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>string</td>
<td>The key identifying the key-value pair.</td>
</tr>
<tr>
<td>value</td>
<td>string</td>
<td>The value associated with the key.</td>
</tr>
<tr>
<td>success</td>
<td>function</td>
<td>The user-defined callback for successful cases.</td>
</tr>
<tr>
<td>error</td>
<td>function</td>
<td>The user-defined callback for failed cases.</td>
</tr>
</tbody>
</table>

**Example**

The code example below shows how to set and remove user data with the SDK API:
Programmatically Control Sessions

By default, a mobile session ends after a period of user inactivity. For example, when a user opens your application, the session begins and only ends after the user stops using the app for a set period of time. When the user begins to use the application again, a new session begins.

Instead of having a period of inactivity to define the duration of a session, however, you can use the following API to programmatically control when sessions begin and end.

```
setCustomData(event, data) {
    window.plugins.ADEUMMobilePlugin.setUserData(data.name, data.value,
        (success) => {
            this.showAlert("setUserData return: success");
        },
        (error) => {
            this.showAlert("setUserData error: " + error);
        }
    );
}
removeUserData(event, key) {
    window.plugins.ADEUMMobilePlugin.removeUserData(key,
        (success) => {
            this.showAlert("removeUserData return: success");
        },
        (error) => {
            this.showAlert("removeUserData error: " + error);
        }
    );
}
```

When you call the method `startNextSession` from `ADEUMMobilePlugin`, the current session ends and a new session begins. The API enables you to define and frame your sessions so that they align more closely with business goals and expected user flows. For example, you could use the API to define a session that tracks a purchase of a product or registers a new user.

Excessive use of this API will cause sessions to be throttled (excessive use is >10 calls per minute, but is subject to change). When not using the API, sessions will fall back to the default of ending after a period of user inactivity.

**Example of a Programmatically Controlled Session**

In the example below, the current session ends and a new one begins when the check out is made.
Start and End Session Frames

You can use Cordova Plugin to create session frames that will appear in the session activity. Session frames provide context for what the user is doing during a session. With the API, you can improve the names of user screens and chronicle user flows within a business context.

**Use Cases**

The following are common using session frames:

- One page performs multiple functions and you want more granular tracking of the individual functions.
- A user flow spans multiple pages or user interactions. For example, you could use the API to create the session frames "Login", "Product Selection", and "Purchase" to chronicle the user flow for purchases.
- You want to capture dynamic information based on user interactions to name session frames, such as an order ID.

**SessionFrame API**

The table below lists the three methods you can use with session frames. In short, you start a session frame with `startSessionFrame` and then use `updateName` and `end` to rename and end the session frame.

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>startSessionFrame</code></td>
<td><code>name</code> (string) - The name of the session frame. <code>success</code> (function) - The user-defined success callback. <code>error</code> (function) - The user-defined error callback.</td>
<td>Use this to start and name your session frame. You call this method from <code>window.plugins.ADEUMMobilePlugin</code>.</td>
</tr>
</tbody>
</table>
### Session Frame Example

In the following example, session frames are used to track user activity during the checkout process.

```javascript
declare var window: any;
@Component({
    ...
})
export class OrderPage {
    sessionFrame: any;
    ...
    
    checkoutCartButtonClicked() {
        // The user starting to check out starts when the user clicks the checkout button
        // this may be after they have updated quantities of items in their cart, etc.
        window.plugins.ADEUMMobilePlugin.startSessionFrame("Checkout", (sessionFrame) => {
            // The returned object is saved to the class property 'sessionFrame' so
            // the SessionFrame API methods 'updateName' and 'end' can be called from it later.
            this.sessionFrame = sessionFrame;
            
            (error) => {
                console.log("startSessionFrame call error:" + error);
            }
        });
    }
    
    confirmOrderButtonClicked() {
        // Once they have confirmed payment info and shipping information, and they
        // are clicking the "Confirm" button to start the backend process of checking out
        // we may know more information about the order itself, such as an Order ID.
        this.sessionFrame.updateName("Checkout: Order ID " + this.orderId,
            (success) => {
                
        
        
        
        
        
```
```javascript
console.log("Order has been placed and sessionFrame updated:" + this.orderId);
}
(error) => {
    console.log("Order has been placed but sessionFrame couldn't be updated because of the error " + error);
}

processOrderCompleted() {
    // Once the order is processed, the user is done "checking out" so we end
    // the session frame.
    this.sessionFrame.end(
        (success) => {
            console.log("Order was completed and sessionFrame ended: " + success);
        },
        (error) => {
            console.log("Order was completed but sessionFrame couldn't be ended because of: " + error);
        }
    );
}

checkoutCancelled() {
    // If they cancel or go back, you'll want to end the session frame also, or else
    // it will be left open and appear to have never ended.
    this.sessionFrame.end(
        (success) => {
            console.log("Order was cancelled and sessionFrame ended: " + success);
        },
        (error) => {
            console.log("Order was cancelled but sessionFrame couldn't be ended because of: " + error);
        }
    );
}
```
Configure and Take Screenshots

Mobile screenshots are enabled by default. These screenshots will show up in the Sessions Details dialog.

You can configure the Controller UI to automatically take screenshots or use the Cordova plugin to manually take a screenshot as shown below:

```javascript
screenshot(event) {
    // make a call to plugin
    console.log("screenshot click handler");
    window.plugins.ADEUMMobilePlugin.takeScreenshot(
        (success) => {
            this.showAlert("screenshot return: success");
        },
        (error) => {
            this.showAlert("screenshot error:" + error);
        });
}
```

This will capture everything, including personal information, so you must be cautious of when to take the screenshot.

Enable Logging and Set Logging Level

You use the method `loggingLevel` to enable and set the logging level. You can set logging to one of the following levels:

<table>
<thead>
<tr>
<th>Value</th>
<th>Logging Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>None</td>
<td>No logs are displayed. <strong>This level disables logging.</strong></td>
</tr>
<tr>
<td>1</td>
<td>Error</td>
<td>Only error messages are displayed. <strong>This is the default logging level.</strong></td>
</tr>
<tr>
<td>2</td>
<td>Warn</td>
<td>Warning and error messages are displayed.</td>
</tr>
<tr>
<td>3</td>
<td>Info</td>
<td>Warning, errors, and developer-focused messages are displayed.</td>
</tr>
<tr>
<td>4</td>
<td>Debug</td>
<td>Errors, warnings, developer information, and debugging messages are displayed.</td>
</tr>
<tr>
<td>5</td>
<td>Verbose</td>
<td>Errors, warnings, developer information, debugging, and troubleshooting messages are displayed.</td>
</tr>
<tr>
<td>6</td>
<td>All</td>
<td>All the supported log messages are displayed.</td>
</tr>
</tbody>
</table>

You enable logging by setting the logging level in the instrumentation configuration. For example, in this example, you are enabling
logging and setting the logging level to `Info`:

```javascript
window.plugins.ADEUMMobilePlugin.initWithConfiguration(
    {
        "appKey": "<EUM_APP_KEY>",
        "loggingLevel": 3
    },
    (success) => {
        this.showAlert("initWithConfiguration return: success");
    },
    (error) => {
        this.showAlert("initWithConfiguration error:" + error);
    }
);
```

**Cordova SDK Documentation**

For the Cordova SDK API reference documentation, see the latest JSDoc documentation or the previous versions listed below:

- https://docs.appdynamics.com/javadocs/cordova-plugin/1/1.1/
- https://docs.appdynamics.com/javadocs/cordova-plugin/1/1.2/
- https://docs.appdynamics.com/javadocs/cordova-plugin/1/1.3/
- https://docs.appdynamics.com/javadocs/cordova-plugin/1/1.4/
- https://docs.appdynamics.com/javadocs/cordova-plugin/1/1.5/
- https://docs.appdynamics.com/javadocs/cordova-plugin/1/1.6/
- https://docs.appdynamics.com/javadocs/cordova-plugin/1/1.7/
- https://docs.appdynamics.com/javadocs/cordova-plugin/1/1.8/
- https://docs.appdynamics.com/javadocs/cordova-plugin/1/1.9/

The version number for the Cordova Plugin is different from that of the Controller and AppDynamics platform components. Because of this difference, see Mobile Agent Version and Deployment Support Matrix for the minimum version of the Controller and the EUM Server required for complete support of all the Cordova Plugin features.
Instrument React Native Applications

Follow the steps below to get your EUM App Key and instrument your React Native application.

1. Get Your Application Key
2. Eject Your React Native Application
3. Install the React Native Agent
4. Add Required Libraries (iOS Only)
5. Add Permissions (Android Only)
6. Instrument Your Application Code
7. Build and Run Your React Native Applications (Optional)

Check Requirements

The React Native Agent has been tested and certified to work with React Native v0.59.x.

Because React Native is being developed rapidly and hasn't officially released v1, any minor update to React Native could potentially affect the functionality of the React Native Agent.

Get Your Application Key

From the Controller's Getting Started Wizard, create either an iOS or an Android application. You will be given an EUM App Key that you will use to instrument your application. In some cases, multiple mobile applications can share the same key.

If you have completed the Getting Started Wizard, but don't have your EUM App Key, see Get Your Application Key.

Eject Your React Native Application

If you created your React Native application with one of the commands in the following table, you will need to eject the application:

<table>
<thead>
<tr>
<th>Command</th>
<th>Why Ejection Is Needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>create-react-native-app</td>
<td>This command enables you to quickly set up and run a Native React application with no configuration but does not provide the full development environment required by the React Native Agent.</td>
</tr>
<tr>
<td>expo init</td>
<td>Expo projects do not currently support the React Native Agent module.</td>
</tr>
</tbody>
</table>

If you created your app with create-react-native-app, use npm to eject the app. If you used expo, run the expo command below to eject the app.

From your project directory, run the command appropriate for your use case:

```sh
npm
```
Install the React Native Agent

This module injects the native AppDynamics agents into your application and offers a JavaScript bridge to the instrumentation management interface.

1. Install and link the React Native Agent:

   ```
   yarn add @appdynamics/react-native-agent
   react-native link @appdynamics/react-native-agent
   ```

2. Build the configuration to enable the build-time instrumentation of your application:

   ```
   node node_modules/@appdynamics/react-native-agent/bin/cli.js install
   ```

   The React Native Agent CLI assumes you've kept the default project structure created with the `react-native` CLI tools.

3. If the above commands failed, see manually install the React Native package. Otherwise, proceed to Add Required Libraries (iOS Only) and/or Add Permissions (Android Only).

Manually Install the React Native Package (Only If Required)

Most users do not have to do manually install the package, but the command above may fail if your iOS and Android projects are
Follow these steps if you need to manually install the React Native package:

### Android

1. Add the following line to your `android/build.gradle`:

   ```gradle
   apply from: 
   '../node_modules/@appdynamics/react-native-agent/android/adem.gradle'
   ``

2. Make sure the file `settings.gradle` contains a reference to the Appdynamics agent module as shown below. This is should have been added automatically by the command `react-native link`.

   ```gradle
   include ':@appdynamics_react-native-agent'
   project(':@appdynamics_react-native-agent').projectDir = 
   new File(rootProject.projectDir, 
   '../node_modules/@appdynamics/react-native-agent/android')
   ``

3. Also, `android/app/build.gradle` should contain a reference to the Appdynamics agent module.

   ```gradle
   dependencies {
   implementation 
   project(':@appdynamics_react-native-agent')
   // ...
   }
   ``

### iOS

For iOS, link the Appdynamics agent module like any other native module. Make sure to add `libsqlite3` to your project as explained in Add Required Libraries (iOS Only).
Add Required Libraries (iOS Only)

1. From the React Native application root, open the iOS project file in the `ios` directory with XCode.
2. From XCode, select your project in the Project Navigator.
3. Navigate to the Build Phases tab.
5. Select `libsqlite3.tbd`.
6. Click Add.

This requirement will be removed in a future release.

Add Permissions (Android Only)

Add the following permissions to your app's `AndroidManifest.xml` (which should be located under `android/app/src/main/`).

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.your.application">

  // Add the following:
  <uses-permission
    android:name="android.permission.ACCESS_NETWORK_STATE"/>

  // ...
</manifest>
```

The React Native Agent tries to minimize its network footprint. Without this permission, the agent always assumes poor network conditions and some metrics might not get reported.

Instrument Your Application Code

In the `index.js` file of your React application, do the following:

1. At the top of the file near the other `import` statements, add the following line of code to import the Native React Agent:

   ```javascript
   import { Instrumentation } from '@appdynamics/react-native-agent';
   ```

2. In the same file and in the global scope, initialize the instrumentation with the following, making sure to replace `<EUM_APP_KEY>` with a string containing your EUM App Key.
3. Verify that your `index.js` file looks similar to the following:

```javascript
import {AppRegistry} from 'react-native';
import App from './App';
// === Add the following to import the React Native Agent
import { Instrumentation } from '@appdynamics/react-native-agent';

// Initialize the instrumentation
Instrumentation.start({
    appKey: 'YOUR-APP-KEY',
});

// Create a component
const App = () => {
    // Other components

    // Register the App component
    AppRegistry.registerComponent('my-application', () => App);
}
```

**Build and Run Your React Native Applications (Optional)**

1. Run the command to build and run the app for your platform.

```bash
react-native run-ios
```
1. Android

```
react-native run-android
```

2. From the Controller UI, verify that the instrumentation was successful.

**Point to an On-Premises EUM Server (Optional)**

To use an on-premises EUM Server, you pass the URL to the on-premises EUM Server when you initialize the instrumentation with the EUM App Key from Get Your Application Key.

```
Instrumentation.start({
    appKey: '<EUM_APP_KEY>',
    collectorURL: '<COLLECTOR_URL>'
});
```
You use the APIs to customize the data for your app that appears in the Controller UI. This page shows you how to use the React Native Agent APIs through example code.

React Native Agent Versions

The version number for the React Native Agent is different from that of the Controller and AppDynamics platform components. Because of this difference, see Mobile Agent Version and Deployment Support Matrix for the minimum version of the Controller and the EUM Server required for complete support of all the React Native Agent features.

Using the React Native Agent APIs

After you have installed and initialized the React Native Agent, you instrument your application code by importing the class, interface, or enumeration and then calling API methods.

Import Syntax

To access the React Native Agent APIs, use an import statement at the top of the file. The following line imports the class `Instrumentation`:

```javascript
import { Instrumentation } from '@appdynamics/react-native-agent';
```

API Method Syntax

Once the class, interface, or enumeration has been imported, call the API methods from the imported component. In this example, you set custom user data with `setUserDataBoolean` from the `Instrumentation` class.
Change the App Key

You can use the Native React Agent API to dynamically change the EUM app key. You receive the EUM App Key when creating a mobile app in the Controller UI. See Set Up and Access Mobile RUM for information about getting the EUM App Key.

Class

The API to change app keys is available through the `Instrumentation` class.

```javascript
import { Instrumentation } from '@appdynamics/react-native-agent';
```

Methods

The API consists of the following method:

- `changeAppKey(appKey)`

Method Parameters

The `changeAppKey` method takes the following parameter:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appKey</td>
<td>string</td>
<td>The EUM App Key.</td>
</tr>
</tbody>
</table>

Examples

In this example, you create a method that takes a new app key and passes it to the API method `changeAppKey`.

```javascript
import { Instrumentation } from '@appdynamics/react-native-agent';
...
private updateAppKey(newAppKey) {
  Instrumentation.changeAppKey(newAppKey);
}
```

Collect Additional Types of Data

The React Native APIs have methods that extend the application data types you can collect and aggregate with Mobile RUM. There are six basic kinds of extensions that you can create:
<table>
<thead>
<tr>
<th>Type of Data</th>
<th>Description</th>
<th>Specifications</th>
<th>Where Data is Displayed</th>
</tr>
</thead>
</table>
| Info points     | How often a method is invoked, and how long it takes to run.                | • Data is numeric  
• Names must consist of alphanumeric characters and/or spaces | • Metric Browser  
• Custom Data  
• Network Request Snapshots  
• Mobile Sessions  
• Network Request Analyze |
| Custom timers   | Any arbitrary sequence of events within your code timed, even spanning multiple methods. | • Data is numeric  
• Metric names must consist of alphanumeric characters and/or spaces | • Metric Browser  
• Custom Data |
| Custom metrics  | Any integer-based data you wish to collect.                                 | • Data is numeric  
• Metric names must consist of alphanumeric characters and/or spaces | • Metric Browser  
• Custom Data |
| User data       | Any string key/value pair you think might be useful.                       | • Data can be any type  
• Metric names have no restrictions | • Network Request Snapshots  
• Mobile Sessions  
• Network Request Analyze |
| Breadcrumbs     | The context for a crash.                                                   | • Data can be any data type  
• Metric names have no restrictions | • Network Request Snapshots  
• Mobile Sessions  
• Network Request Analyze |
| User interactions | Capture when users press buttons, click on lists, and select text.     | • Data can be any data type  
• Metric names have no restrictions | • Network Request Snapshots  
• Mobile Sessions  
• Network Request Analyze |

Report Info Points

Information points allow you to track how your own code is running. Unlike the other React Native APIs that require importing a module and using a method from that module, you add annotations to your code to report info points.

**Class/Interface**

The API is available through the `Instrumentation` class and the `InfoPoint` function.

```javascript
import { Instrumentation, InfoPoint } from '@appdynamics/react-native-agent';
```

**Methods**

With the `@InfoPoint` annotation, you only need to provide a function to report info points. The `@InfoPoint` annotation has two signatures:

- `@InfoPoint`
- `@InfoPoint({className: string, fnName: string})`

**Method Signatures**
Methods | Parameters | Description
---|---|---
@InfoPoint | None | Automatically reports the info points of the method that is being annotated.
@InfoPoint({className: string, fnName: string}) | Object | Manually specify the class and function names to annotate. For example: `@InfoPoint({ className: 'Automobiles', fnName: 'checkInventory' })`

**Examples**

The following examples show you how to use automatic and manual reporting of info points.

You are recommended to use automatic reporting if your code will not be minified: it's simpler and reports the same information. If your code will be minified, however, you should manually report the class and function names to avoid reporting the useless class and function names created by minification.

**Automatic Reporting**

Add the annotation `@InfoPoint` to automatically create an info point for the method below it. Info point reports how the method is invoked and how long it takes to run.

```javascript
import { Instrumentation } from '@appdynamics/react-native-agent';
...
@InfoPoint
public infoPointMethod(arg1, arg2, value) {
    console.log("Executing infoPointMethod!");
}
```

**Manually Reporting**

To create an info point for a specific method in a class (the symbol names), pass an object to the `@InfoPoint` annotation that specifies the class and method name as shown below.

```javascript
import { Instrumentation } from '@appdynamics/react-native-agent';
...
@InfoPoint({ className: 'MyClass', fnName: 'infoPointMethod' })
...
public infoPointMethod(arg1, arg2, value) {
    console.log("Executing infoPointMethod!");
}
```

**Set Custom Timers**
You create custom timers to time any arbitrary sequence of events within your code, even spanning multiple methods.

**Class/Interface**

The Custom Timers API is accessible through the `Instrumentation` class:

```javascript
import { Instrumentation } from '@appdynamics/react-native-agent';
```

**Methods**

You create the custom timers using the following React Native Agent API methods:

- `startTimerWithName(name)`
- `stopTimerWithName(name)`

**Method Parameters**

Both methods take the following parameter:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the custom timer. Allowed characters are [A-Za-z0-9]. Illegal characters are replaced by their ASCII hex value.</td>
</tr>
</tbody>
</table>

**Examples**

For example, to track the time a user spends viewing a screen, the instrumentation could look like this:

```javascript
import { Instrumentation } from '@appdynamics/react-native-agent';
...
private startCustomTimer() {
    Instrumentation.startTimer("My timer");
}
private stopCustomTimer() {
    Instrumentation.stopTimer("My timer");
}
```

**Create Custom Metrics**

The React Native API enables you to report custom metrics. You specify the name of the custom metric that will appear in the Controller UI.

**Class**

The Custom Metrics API is accessible through the `Instrumentation` class:
You create custom metrics with the `reportMetric`:

- `reportMetric(name, value)`

### Method Parameters

The `reportMetric` method takes the following parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Requirements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>The metric names must consist of alphanumeric characters. Illegal characters are replaced by their ASCII hex value.</td>
<td>The name of the custom metric.</td>
</tr>
<tr>
<td>value</td>
<td>number</td>
<td>The value must be a whole number, otherwise, an error will be returned.</td>
<td>The numeric value associated with the custom metric.</td>
</tr>
</tbody>
</table>

### Examples

For example, the following method could be used to report custom metrics:

```javascript
import { Instrumentation } from '@appdynamics/react-native-agent';
...
private reportMetrics() {
  Instrumentation.reportMetric("Normal metric", 23);
  Instrumentation.reportMetric("Large metric", Number.MAX_SAFE_INTEGER + 1);
  Instrumentation.reportMetric("Small metric", Number.MIN_SAFE_INTEGER - 1);
}
```

### Add Custom User Data

Use the Custom User Data API to set and remove key/value pairs of different data types. The key must be unique across your application.

### Class

The Custom User Data API is accessible through the `Instrumentation` class:
import { Instrumentation } from '@appdynamics/react-native-agent';

Methods

The API provides the following methods for setting and removing custom user data:

Methods for Setting Custom User Data

- `setUserData(key, value)`
- `setUserDataBoolean(key, value)`
- `setUserDataDate(key, value)`
- `setUserDataDouble(key, value)`
- `setUserDataInteger(key, value)`

Methods for Removing Custom User Data

- `removeUserData(key)`
- `removeUserDataBoolean(key)`
- `removeUserDataDate(key)`
- `removeUserDataDouble(key)`
- `removeUserDataInteger(key)`

Method Parameters

The following table lists the parameters for the methods for setting custom user data.

<table>
<thead>
<tr>
<th>Methods for Setting Custom User Data</th>
<th>Parameters</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setUserData</code></td>
<td>key</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>value</td>
<td>string</td>
</tr>
<tr>
<td><code>setUserDataBoolean</code></td>
<td>key</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>value</td>
<td>boolean</td>
</tr>
<tr>
<td><code>setUserDataDate</code></td>
<td>key</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>value</td>
<td>Date object</td>
</tr>
<tr>
<td><code>setUserDataDouble</code></td>
<td>key</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>value</td>
<td>number</td>
</tr>
<tr>
<td><code>setUserDataInteger</code></td>
<td>key</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>value</td>
<td>number</td>
</tr>
</tbody>
</table>

The following table lists the parameters for the methods for removing custom user data.

<table>
<thead>
<tr>
<th>Methods for Removing Custom User Data</th>
<th>Parameters</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>removeUserData</code></td>
<td>key</td>
<td>string</td>
</tr>
<tr>
<td><code>removeUserDataBoolean</code></td>
<td>key</td>
<td>string</td>
</tr>
<tr>
<td><code>removeUserDataDate</code></td>
<td>key</td>
<td>string</td>
</tr>
<tr>
<td><code>removeUserDataDouble</code></td>
<td>key</td>
<td>string</td>
</tr>
</tbody>
</table>
The method `removeUserData` is called when setting custom data to the value `null`. For example, the method `Instrumentation.setUserData("name", null)` will call `Instrumentation.removeUserData("name")`.

**Examples**

The following code example shows how to set and remove user data with the Custom User Data API:

```javascript
import { Instrumentation } from '@appdynamics/react-native-agent';
...
private setUserData() {
    Instrumentation.setUserData("userId", "AISJ1723871");
    Instrumentation.setUserDataBoolean("isVip", true);
    Instrumentation.setUserDataDate("purchaseDate", new Date(1234567890));
    Instrumentation.setUserDataDouble("monthlyVisits", 1.2345);
    Instrumentation.setUserDataInteger("totalPurchasedItems", 42);
}
private clearUserData() {
    Instrumentation.removeUserData("userId", null);
    Instrumentation.removeUserDataBoolean("isVip", null);
    Instrumentation.removeUserDataDate("purchaseDate", null);
    Instrumentation.removeUserDataDouble("monthlyVisits", null);
    Instrumentation.removeUserDataInteger("totalPurchasedItems", null);
}
```

**Leave Breadcrumbs**

Breadcrumbs enable you to situate a crash in the context of your user’s experience. Set a breadcrumb when something interesting happens. If your application crashes at some point in the future, the breadcrumb will be displayed along with the crash report. Each crash report displays the most recent 99 breadcrumbs.

**Class**

The Breadcrumb API is accessible through the `Instrumentation` class:

```javascript
import { Instrumentation } from '@appdynamics/react-native-agent';
```

**Methods**

You create and leave breadcrumbs with the following API method:

- `leaveBreadcrumb(breadcrumb, mode)`
**Method Parameters**

The method `leaveBreadcrumb` takes the following parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>breadcrumb</td>
<td>string</td>
<td>The string to include in the crash report and sessions. Truncated at 2048 characters; empty values are ignored.</td>
</tr>
<tr>
<td>mode</td>
<td>Enumeration</td>
<td>The mode determining where the <code>breadcrumb</code> will be displayed. You can report crashes only or crashes and sessions with the following modes: &lt;ul&gt;&lt;li&gt;BreadcrumbVisibility.CRASHESONLY&lt;/li&gt;&lt;li&gt;BreadcrumbVisibility.CRASHES_AND_SESSIONS&lt;/li&gt;&lt;/ul&gt;The default mode is <code>BreadcrumbVisibility.CRASHESONLY</code>. If the value is invalid, the default mode will be applied.</td>
</tr>
</tbody>
</table>

**Examples**

**Basic Usage**

The following example shows the syntax and usage of the Breadcrumb API.

```javascript
import { Instrumentation } from '@appdynamics/react-native-agent';
...
private leaveBreadcrumb() {
    Instrumentation.leaveBreadcrumb("Drop a breadcrumb button implementation", BreadcrumbVisibility.CRASHES_AND_SESSIONS);
}
```

**Extended Use Case**

Supposed your React Native app has a wizard that shows different steps of a process sequentially on the same screen. In the case of a crash or an "Application Not Responding" (ANR) error, you would want to know what step of the wizard caused the crash or ANR.

If your app had a wizard engine similar to the one below, you could leave a breadcrumb for each screen to track events and issues.

```javascript
async function wizard(...screens) {
    let currentScreen = 0;
    while (true) {
        const screen = screens[currentScreen];
        if (screen == null) return;
        // Report the current screen with AppDynamics instrumentation
        Instrumentation.leaveBreadcrumb('wizard screen ' + screen.name);
        currentScreen += await screen.action();
    }
}
```
The wizard engine could be used for a checkout:

```javascript
wizard({
  name: 'review cart', action:reviewCartAction },
  name: 'chose payment', action:chosePaymentAction },
  name: 'chose address', action:choseAddressAction },
  name: 'review order', action:reviewOrderAction },
  name: 'checkout', action:checkoutAction } }
```

If you receive an alert that your app is triggering a lot of ANRs, you can diagnose the problem through the mobile sessions where the ANRs occur. In the session, you will see the breadcrumbs associated with the ANRs. For example, you may discover that the breadcrumbs "review cart" and "chose payment" are associated with ANRs, but "chose address" is not. Further investigation of the payment screen could lead to the discovery that the process that is encrypting credit card numbers is running on the main thread and freezing the app.

Capture User Interaction

The React Native Agent can track certain UI events triggered by user interactions. Once user interactions have been captured, you can sort sessions by UI event and view UI events in the timeline of the session waterfall.

**Supported User Interactions**

You can capture when users do one or all of the following:

- press buttons
- select table cells
- select text fields
- select text views

**Security and Privacy Concerns**

The interaction capture mode is disabled by default for security and privacy reasons, as user interactions may contain sensitive information. Moreover, this potential security and privacy issue may be compounded if you enable both the capturing of UI interactions and screenshots.

**Class**

The Interaction Capture Mode API is accessible through the `InteractionCaptureMode` class, but you also need the `Instrumentation` class to start instrumentation with the configured user capture mode.

```javascript
import { Instrumentation, InteractionCaptureMode } from '@appdynamics/react-native-agent';
```

**Properties**

The following properties are accessed through the `InteractionCaptureMode` class and used to configure the user interaction capture. For example: `InteractionCaptureMode.All`
### Property Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Platform Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Track all the user interactions.</td>
<td>iOS, Android</td>
</tr>
<tr>
<td>ButtonPressed</td>
<td>Track button presses.</td>
<td>iOS, Android</td>
</tr>
<tr>
<td>ListViewItemsSelected</td>
<td>Track &quot;List Item&quot; clicks and focus changes for android.widget.AbsListView and its subclasses.</td>
<td>Android</td>
</tr>
<tr>
<td>None</td>
<td>Disables the tracking of any user interactions.</td>
<td>iOS, Android</td>
</tr>
<tr>
<td>TableCellSelected</td>
<td>Track table cell selection.</td>
<td>iOS</td>
</tr>
<tr>
<td>TextFieldSelected</td>
<td>Track text field selection.</td>
<td>iOS, Android</td>
</tr>
<tr>
<td>TextViewSelected</td>
<td>Track text view selection.</td>
<td>iOS</td>
</tr>
</tbody>
</table>

### Methods

The API methods take an array of user interaction capture modes and return an `InteractionCaptureMode` object that determines which user interactions will be captured. The methods do not mutate the object.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>with</td>
<td>Combines the multiple user interaction capture modes.</td>
</tr>
<tr>
<td>without</td>
<td>Excludes the specified user interaction modes from being captured.</td>
</tr>
</tbody>
</table>

### Examples

#### Set User Interaction Mode with API Properties

The following example configures the React Native Agent to capture all user interactions.

```javascript
import { Instrumentation, InteractionCaptureMode } from '@appdynamics/react-native-agent';
...
Instrumentation.start({
    appKey: '<EUM_APP_KEY>',
    // ...your other configurations

    // Capture all user interactions
    interactionCaptureMode: InteractionCaptureMode.All
})
```

#### Combine User Interaction Modes

In the following example, the `with` method combines the `ButtonPressed` (button presses) and `TextFieldSelected` (text field selected) modes with the `None` mode, effectively just enabling those two user interaction modes.
import { Instrumentation, InteractionCaptureMode } from '@appdynamics/react-native-agent';
...
Instrumentation.start({
  appKey: <EUM_APP_KEY>,
  // ...your other configurations

  // Only enable "ButtonPressed" and "TextFieldSelected" interaction modes, and disable the rest.
  interactionCaptureMode: InteractionCaptureMode.None.with({
    InteractionCaptureMode.ButtonPressed,
    InteractionCaptureMode.TextFieldSelected
  })
})

Excluding User Interaction Modes

The following example excludes the user interaction modes ButtonPressed (button presses) and TextFieldSelected (text field selected) from the mode InteractionCaptureMode.All (all user interactions). Effectively, the React Native Agent would be configured to only capture ListViewItemSelected ("List Item" clicks), TableCellSelected (table cell selection), and TextViewSelected (text view selected) modes.

import { Instrumentation, InteractionCaptureMode } from '@appdynamics/react-native-agent';
...
Instrumentation.start({
  appKey: <EUM_APP_KEY>,
  // ...your other configurations

  // Include all capture modes except "ButtonPressed" and "TextFieldSelected".
  interactionCaptureMode: InteractionCaptureMode.All.without({
    InteractionCaptureMode.ButtonPressed,
    InteractionCaptureMode.TextFieldSelected
  })
})

Programmatically Control Sessions

By default, a mobile session ends after a period of user inactivity. For example, when a user opens your application, the session begins and only ends after the user stops using the app for a set period of time. When the user begins to use the application again, a new
A session begins. Instead of having a period of inactivity to define the duration of a session, however, you can use this API to programmatically control when sessions begin and end.

**Class**

The API is accessible through the `Instrumentation` class:

```javascript
import { Instrumentation } from '@appdynamics/react-native-agent';
```

**Methods**

The API provides the following method for ending the current session and starting a new session:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>startNextSession()</code></td>
<td>When you call the method <code>startNextSession</code> from <code>Instrumentation</code>, the current session ends and a new session begins. The API enables you to define and frame your sessions so that they align more closely with business goals and expected user flows. For example, you could use the API to define a session that tracks a purchase of a product or registers a new user.</td>
</tr>
</tbody>
</table>

Excessive use of this API will cause sessions to be throttled (excessive use is >10 calls per minute, but is subject to change). When not using the API, sessions will fall back to the default of ending after a period of user inactivity.

**Examples**

In the following example, the current session ends and a new one begins when the check out is made.

```javascript
import { Instrumentation } from '@appdynamics/react-native-agent';
...
private checkoutCart(){
    if (currentCartItems!=null && currentCartItems.size()>0){
        CheckoutTask checkoutReq = new CheckoutTask();
        checkoutReq.execute(getEndpoint() + "cart/co");
        currentCartItemsMap.clear();
        convertItemsMaptoList();
        Instrumentation.startNextSession();
    } else {
        displayToast("There are no items in the cart");
    }
}
```

**Start and End Session Frames**

You can use the React Native Agent API to create session frames that appear in the session activity. Session frames provide context for what the user is doing during a session. With the API, you can improve the names of user screens and chronicle user flows within a
business context.

**Use Cases**

The following are common use cases for session frames:

- One page performs multiple functions and you want more granular tracking of the individual functions.
- A user flow spans multiple pages or user interactions. For example, you could use the API to create the session frames “Login”, “Product Selection”, and “Purchase” to chronicle the user flow for purchases.
- You want to capture dynamic information based on user interactions to name session frames, such as an order ID.

**Class/Interface**

The SessionFrame API is accessible through the Instrumentation class.

```java
import { Instrumentation } from '@appdynamics/react-native-agent';
```

To update the name and end a session frame, you would use the SessionFrame interface.

**Methods**

The following table lists the three methods you can use with session frames. In short, you start a session frame with `startSessionFrame` and then use `updateName` and `end` to rename and end the session frame.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>startSessionFrame(name)</code></td>
<td>Starts the session frame. You call this method from Instrumentation, and it returns SessionFrame object.</td>
</tr>
<tr>
<td><code>updateName(name)</code></td>
<td>Updates the name of the session frame. You call this method from the SessionFrame object.</td>
</tr>
</tbody>
</table>

**Method Parameters**

The methods `startSessionFrame` and `updateName` takes the following parameter:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the session frame.</td>
</tr>
</tbody>
</table>

**Session Frame Example**

In the following example, session frames are used to track user activity during the checkout process.
let sessionFrame: SessionFrame | undefined;

private onCheckoutCartButtonClicked() {
  // The user starts the checkout by clicking the checkout button.
  // This may be after they have updated the quantities of items in the
cart, etc.
  sessionFrame = Instrumentation.startSessionFrame("Checkout");
}

private onConfirmOrderButtonClicked() {
  // Once they have confirmed payment info and shipping information,
  // and they
  // are clicking the "Confirm" button to start the backend process of
  // checking out,
  // we may know more information about the order itself, such as an
  // order ID.
  if (sessionFrame) {
    sessionFrame.updateName("Checkout: Order ID " + orderId);
  }
}

private onProcessOrderCompleted() {
  // Once the order is processed, the user is done "checking out", so
  // we end the session frame.
  if (sessionFrame) {
    sessionFrame.end();
    sessionFrame = null;
  }
}

private onCheckoutCancled() {
  // If the user cancels or returns to the cart, you'll want to end the
  // session frame also, or else it will be
  // left open and appear to have never ended.
  sessionFrame.end();
  sessionFrame = null;
}

Configure and Take Screenshots

Mobile screenshots are enabled by default. These screenshots appear in the **Sessions Details** dialog.

**Class**

The Screenshots API is accessible through the **Instrumentation** class:
```javascript
import { Instrumentation } from '@appdynamics/react-native-agent';
```

**Methods**

The Screenshot API provides the following methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>takeScreenshot()</td>
<td>None</td>
<td>Asynchronously takes a screenshot of the current screen. This will capture everything, including personal information, so you must be cautious of when to take the screenshot. These screenshots will show up in the Sessions screen for this user. The screenshots are taken on a background thread, compressed, and only non-redundant parts are uploaded, so it is safe to take many of these without impacting the performance of your application.</td>
</tr>
<tr>
<td>screenshotsBlocked()</td>
<td>Promise&lt;boolean&gt;</td>
<td>Returns a Boolean indicating whether screenshot capture is blocked.</td>
</tr>
<tr>
<td>blockScreenshots()</td>
<td>Promise&lt;void&gt;</td>
<td>Blocks screenshot capture and returns a Promise that resolves when screenshots are effectively blocked.</td>
</tr>
<tr>
<td>unblockScreenshots()</td>
<td>Promise&lt;void&gt;</td>
<td>Unblocks screenshot capture if it is currently blocked. Otherwise, this has no effect. If screenshots are disabled through <code>AgentConfiguration.screenshotsEnabled</code> or through the Controller UI, this method has no effect. If screenshots are set to manual mode in the Controller UI, this method unblocks for manual mode only.</td>
</tr>
</tbody>
</table>

**Configure Screenshots**

**Disable Screenshots**

You can disable screenshots from the Controller UI or with the React Native API. To disable screenshots, set the property `screenshotsEnabled` to `false` when initializing the Native React Agent.

```javascript
Instrumentation.start({
    appKey: <EUM_APP_KEY>,
    screenshotsEnabled: false,
    ...
});
```

**Set On-Prem Screenshot Service**

If you have deployed an on-prem EUM Server, you will need to specify the URL to the EUM Server with the property `screenshotURL`. 
Instrumentation.start({
    appKey: '<EUM_APP_KEY>',
    screenshotURL: "https://<COLLECTOR_URL>:<PORT>",
    ...
});

Examples

Take Screenshots

You can configure the Controller UI to automatically take screenshots or use the React Native API to manually take a screenshot as shown below:

```javascript
private loadShoppingCart() {
    // Load shopping cart
    this.setState({
        shoppingCart: this.state.cart
    });
    // Manually take screenshot
    Instrumentation.takeScreenshot();
}
```

This will capture everything, including personal information, so precaution is necessary when taking screenshots

Block Screenshots

You can also block screenshots from being taken during the execution of a code block. This just temporarily blocks screenshots from being taken until you unblock screenshots. This enables you to stop taking screenshots in situations where users are entering personal data, such as on login and account screens.
private displayCustomerAccount() {
    // Check to see if screenshots are blocked
    if (!Instrumentation.screenshotsBlocked()) {
        // If screenshots aren't blocked, block them before showing
        customer details
        Instrumentation.blockScreenshots();
    }
    // Code to display customer details

    // After you're done, unblock screenshots
    Instrumentation.unblockScreenshots();
}

Enable Logging and Set Logging Level

You enable logging by setting the logging level in the instrumentation configuration. You are recommended to disable logging for production.

**Configuration Properties**

You enable and set the logging level with the following configuration property:

- `loggingLevel`

**Logging Levels**

You can set logging to one of the following levels:

<table>
<thead>
<tr>
<th>Enumeration</th>
<th>Logging Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>InstrumentationModule.LOGGING_LEVEL_NONE</code></td>
<td>None</td>
<td>No logs are displayed. <strong>This level disables logging.</strong></td>
</tr>
<tr>
<td><code>InstrumentationModule.LOGGING_LEVEL_INFO</code></td>
<td>Info</td>
<td>Warning, errors, and developer-focused messages are displayed.</td>
</tr>
<tr>
<td><code>InstrumentationModule.LOGGING_LEVEL_VERBOSE</code></td>
<td>Verbose</td>
<td>Errors, warnings, developer information, debugging, and troubleshooting messages are displayed.</td>
</tr>
</tbody>
</table>

**Examples**

In the following example, logging is enabled and the logging level is set to `Verbose`:
private async start() {
    try {
        await Instrumentation.start({
            appKey: this.state.appKey,
            loggingLevel: LoggingLevel.VERBOSE,
            anrDetectionEnabled: true,
            interactionCaptureMode: InteractionCaptureMode.None.with(
                InteractionCaptureMode.ButtonPressed,
                InteractionCaptureMode.ListViewItemSelected,
                InteractionCaptureModeTableCellSelected,
                InteractionCaptureMode.TextFieldSelected,
                InteractionCaptureMode.TextViewSelected
            )
        })
    }
}

---

**React Native API Documentation**

For the React Native Agent API reference documentation, see the [latest React Native Agent API documentation](https://docs.appdynamics.com/javadocs/react-native-agent/1/1.0/) or the previous versions:

- [https://docs.appdynamics.com/javadocs/react-native-agent/1/1.0/](https://docs.appdynamics.com/javadocs/react-native-agent/1/1.0/)
Confirm the Mobile Agent Connected to the Controller

Related pages:
- Instrument iOS Applications
- Instrument Android Applications
- Instrument Xamarin Applications

After you have instrumented your application, the Getting Started Wizard will show you a verification that the Mobile Agent has connected to the Controller.

If you left the Getting Started Wizard, you can always verify the instrumentation by doing the following:

1. In the Controller UI, open User Experience > Mobile Apps.
2. Check the list of registered mobile applications to verify that the application is registered with the Controller. You can also use view some basic information about the app such as the health, sessions, crashes, network request information, and whether the monitoring is enabled.

3. Start monitoring your application! See Monitor Your Applications with Mobile RUM.
Configure the Controller UI for Mobile RUM

Using the Controller UI, you can configure:

- how mobile requests are named
- the thresholds that cause network request snapshots to be considered slow, very slow or stalled
- percentile levels you would like to display, if any
- which network requests are sent to the Event Service
- if the IP address from which the request comes should be stored

To access mobile request configuration:

1. Open the mobile application in which you are interested.
2. From the left-hand navigation menu, click Configuration.
3. From the Configuration page, click Mobile App Group Configuration >.

To configure Mobile RUM from the Controller UI, your user account must belong to a role that has the Configure EUM permission. See End User Monitoring Permissions for more information.
Configure Mobile Network Request Naming

On this page:
- Default Network Request Naming Configuration
- Modify the Default Naming Configuration Rule
- Create Mobile Include Rules
- Create Mobile Exclude Rules
- Change Priority of Rules

Related pages:
- Network Request Limits
- Network Requests

The following sections show you how to modify the default naming configuration for network requests and create include/exclude naming rules for network requests.

Default Network Request Naming Configuration

By default, AppDynamics names network requests using:
- the hostname
- the first two segments of the URL

For example, if an application makes this HTTP request:

```
http://myapp.com/friends/profiles/12345
```

The default name that is displayed in the Controller UI for that request is:

```
myapp.com/friends/profiles
```

If this is adequate for your needs, you can leave the default as is.

The naming rules you configure here apply to all the mobile applications that are in the same Mobile App Group.

Modify the Default Naming Configuration Rule

You may want to configure a different default rule for naming your network requests to help you visualize the parts of your application more clearly. The task is similar to configuring naming rules for business transactions on the server side. Try to group logically related requests together while keeping unrelated requests in separate groups.

- If the default hostname and first two segments of the URL for all your requests are identical, you might want to name the requests based on the last segments or a selection of non-contiguous segments of the URL to distinguish among requests in the network requests list.
- You can also name the requests based on query parameters. For example, if the request passes an order number, you could specify that the value of the order-number query parameter is used in the network request name.
- You can also base the name on a regular expression run on the URL. AppDynamics uses the Java libraries for regular expressions. For more information see:
Modifying the Default Network Request Naming Rule

The default configuration covers how all your requests are named if you do not customize them further.

1. From the Network Request tab, scroll down to the Include Rules section.
2. Double-click Default Naming Configuration.
3. In the Include Rule dialog, select the elements you want to use for your default network request naming.
4. Click OK.
5. Click Save.

Create Mobile Include Rules

By default, the same request naming rule is applied to every URL that your application requests. If you want to apply different naming rules to different URLs, create include rules.

For example, if some requests call your own in-house server and others call out to a third-party API, you may want to see all the third-party API calls as one network request and use the default naming rules for the calls to your own server. You would create a custom naming rule that matches the third party calls and uses only the host in the default rule name or perhaps also include certain query parameters.

Creating an Include Rule

1. From the Network Request tab, scroll down to the Include Rules section.
2. Click Add.
3. In the Include Rule dialog, enter a name for the custom rule that you are creating.
4. Check the Enabled check box to enable the rule.
5. Select the check boxes and radio buttons and enter the match criteria for AppDynamics to use to name network requests.
6. Click OK.

Guidelines for Using Regular Expressions in Mobile Include Rules

When using regular expressions to match URLs in your include rules, you should note the following:

- URL strings are case-sensitive. So, although the page names displayed in the Network Request Dashboard are converted to lowercase, your regular expressions still need to match the case used in URLs that your include rule are trying to capture.
- Your regular expression should match the entire URL, from beginning to end, not just a section; otherwise, the rule will not match. This differs from using regular expressions in custom match rules for naming transactions, which only need to match sections of the URL.

Sample Include Rule

The following rule creates a custom match rule for requests in which the URL contains "ourpartner.com". This rule uses the protocol, the subdomain and the third and fourth segments of the URL in the network request name.
You can temporarily cancel the application of a custom naming rule by clearing the Enabled checkbox in the custom rule configuration. In this case, the default naming rule is applied to requests that would have been named by the disabled custom rule. To remove the rule permanently, select the custom rule in the Custom Naming Rules list and click the Delete icon.

Create Mobile Exclude Rules

If there are certain types of requests that you do not want to monitor, create custom exclude rules for them based on the URL and/or the application name. Excluded network requests are not reported or counted toward the network request limit of 500 requests per controller application.

**Creating an Exclude Rule**

1. From the Network Request tab, scroll down to the Exclude Rules section.
2. Click Add.
3. In the Exclude Rule dialog, enter a name for the exclude rule that you are creating.
4. Check the Enabled check box to enable the rule.
5. Select the check boxes and radio buttons and enter the match criteria for AppDynamics to use to name network requests.
6. Click OK.

You can temporarily cancel the application of an exclude rule by clearing the Enabled checkbox in the exclude rule configuration. To remove the rule permanently, select the exclude rule in the Exclude Rules list and click the Delete icon.

Change Priority of Rules

Rules are evaluated in the order that they appear in the include or exclude list. You can change the priority of the rules by dragging and dropping rules towards the top (higher priority) or towards the bottom of the list (lower priority). Custom rules are always evaluated before the default naming rule, beginning with the custom rule that has the highest priority.
Configure Mobile Network Request Thresholds

On this page:
- Default Network Request Thresholds
- Configure Network Request Thresholds

Related pages:
- Network Request Limits
- Network Requests

The Mobile Agent uses configurable thresholds to determine whether network request time is normal, slow, very slow or stalled. It uses these thresholds:

- To decide whether to create a mobile request snapshot
- For labeling network request experience in a network request snapshot

Default Network Request Thresholds

By default, the Mobile Agent uses the following default values to determine whether a request is slow or stalled:

- **Slow**: greater than 3 standard deviations
- **Very slow**: greater than 4 standard deviations
- **Stall**: greater than 45000 ms

You should configure these defaults to conform to your own criteria for your mobile applications. An absolute threshold rather than one based on the standard deviation is often more appropriate for mobile applications.

Configure Network Request Thresholds

1. From the Mobile App Group Configuration page, click the Settings tab.
2. In the Thresholds for Slow End User Experience section, set the thresholds for slow, very slow, and stalled in milliseconds or by standard deviations.
3. Click Save.
Configure Mobile Percentile Metrics

On this page:

- Access Configure Percentile Metrics
- Configure Percentile Metrics

Related pages:

- Mobile App Dashboard

Parts of the Controller UI for Mobile RUM rely on the processing done by the Events Service, including some of the widgets in the Mobile App Dashboard Overview, Network Requests Analyze, and Crash Analyze.

For the Mobile App Dashboard, Network Requests Analyze, and Crash Analyze, you can choose widgets to display the metrics collected using either averages or percentiles.

A percentile is a measure that indicates a value below which a given percentage of values in a set falls: for example, the 99th percentile means that 99% of all values are below this level.

Using percentiles can be a good way to reduce the impact of extreme outliers in performance metrics, which can be useful in the often noisy environments of end-user experience monitoring. Percentiles are also displayed in the Metric Browser.

You can:

- enable or disable percentile display of metrics.
- set up to four different percentile levels to be applied to metrics.

Access Configure Percentile Metrics

From the Configuration > Mobile App Group Configuration, click the Settings tab and navigate to the Configure Percentile Metrics section.

Configure Percentile Metrics

You can change the percentiles which are used to evaluate data via the Events Service.

Enable Percentile Metrics

Percentiles to Collect:

- 50
- 75
- 95
- 99

Up to four different percentiles can be collected per EU metric. Each percentile must be a whole number between 1 and 99.

1. Check the box to enable percentile metric display.
2. Add up to four percentile levels to collect. Each value must be a whole number between 1 and 99.

If you change the percentiles to collect new values, it takes some time for the recalculation to take place. If you look in the Metric Browser you see metrics based on the old percentiles until the moment that you update the values, when they change to reflect the change.
Configure Which Network Requests Are Sent to the Event Service

On this page:
- Access Network Requests Rules
- Configure Include Rules
- Configure Exclude Rules
- Change Priority of Rules

Related pages:
- Network Request Limits
- Network Requests

Your mobile app may make various kinds of network requests, and not all of them may be equally important to monitor in detail. For example, any requests to Google Analytics that your app may make are useful but probably aren’t as important to analyze as the requests it makes to your backend.

To manage the impact on your overall Event Service usage, you can create rules which specify which of these network requests should be sent on to the Event Service, either by excluding a request entirely, including a particular request or a sample of that request types by percentage, or by simply allowing the request to be sent on.

In general, the behavior follows this pattern:

- If no rules are specified, data on all network requests are sent on.
- If exclude rules are specified, and a network request satisfies a rule, that data is not sent on.
- If include rules are specified, any network request that satisfies a rule is sent on, based on sampling defined by the percentage indicated in the rule.
- If both include and exclude rules are specified, a network request that satisfies an include rule but does not satisfy an exclude rule is sent on.

Access Network Requests Rules

From the Mobile App Group Configuration page, select the Event Service tab.

Configure Include Rules

1. Click Add to create a new rule. The Include Rule popup appears.
2. Give your rule a display name.
3. Check Enabled to place the rule in force.
4. Specify the Network Request URL using any of the options in the dropdown.
5. If you want to sample this request type, select a percentage for sampling.
6. Click OK.
Configure Exclude Rules

1. Click Add to create a new rule. The Exclude Rule popup appears.

   ![Exclude Rule popup](image)

   - **Enabled**: Check to place the rule in force.
   - **Rule Name**: Specify a display name for the rule.
   - **Criteria**: Specify the Network Request URL using any of the options in the dropdown.
   - **Contains**: Select one of the criteria options.
   - **Cancel** / **OK**: Click Cancel to cancel or OK to save the rule.

2. Specify a display name for the rule.
3. Check Enabled to place the rule in force.
4. Specify the Network Request URL using any of the options in the dropdown.
5. Specify the percentage of URLs to sample.
6. Click OK.

Change Priority of Rules

Rules are evaluated in the order that they appear in the include or exclude list. You can change the priority of the rules by dragging and dropping rules towards the top (higher priority) or towards the bottom of the list (lower priority). Custom rules are always evaluated before the default naming rule, beginning with the custom rule that has the highest priority.
Configure Request IP Address Storage - Mobile
For security and privacy reasons, Mobile RUM does not store source IP addresses associated with network requests.

To have Mobile RUM store source IP addresses associated with network requests, following the instructions below.

1. From the Mobile App Group Configuration page, select the Settings tab.
2. Click the Mobile Network Request Naming, Thresholds & Percentiles tab.
3. Check the Store IP Address checkbox to enable IP address storage. (By default, the box is not checked.)
4. Click Save.
Configure Session Monitoring

On this page:

- Access Session Monitoring
- Set Inactivity Timeout

Related pages:

- Mobile Sessions

Sessions begin when a user starts using your application and ends after a configurable period of user inactivity. Increasing the period of user inactivity enables sessions to capture more user events, but this could impact the session limits of your account. See Mobile Sessions for more information.

Access Session Monitoring

1. Open the mobile application in which you are interested.
2. From the left-hand navigation menu, select Configuration.
3. Click Mobile App Group Configuration >.
4. Select the Settings tab.
5. Find the Configure Session Monitoring section.

Set Inactivity Timeout

From Configure Session Monitoring:

1. Set the number of minutes for the Session Inactivity Timeout field.
2. Click Save.
Configure Mobile Crash Alerts

On this page:
- Access Configure Mobile Crash Alerts
- Set Threshold for New Crash Alerts

Related pages:
- Crashes
- Crash Alerts

New crashes are shown in the Events widget, and you can configure Mobile RUM to send you alerts when new crashes occur. See Crash Alerts for more information.

To restrict the number of alerts for new crashes, follow the instructions below to configure the number of crashes that must occur before you are alerted.

Access Configure Mobile Crash Alerts

1. Open the mobile application in which you are interested.
2. On the left navigation bar, select Configuration.
3. Click Mobile App Group Configuration >.
4. Select the Settings tab.
5. Navigate to the Configure Mobile Crash Alerts section.

Set Threshold for New Crash Alerts

From Configure Mobile Crash Alerts:

1. Set the threshold for new crash alerts. The default value is 1.
2. Click Save.
Configure Application Not Responding Thresholds

On this page:

- Default Network Request Thresholds
- Configure Application Not Responding Thresholds

The Mobile Agent uses configurable thresholds to set the severity level for application-not-responding (ANR) errors. The severity levels for ANRs are **warning** and **critical**, and these levels can be used to set health rules and receive alerts for health rule violations.

You can use Controller UI to configure thresholds for warning and critical severity levels. These thresholds will be propagated to the Mobile Agents without the need to redeploy the application.

Default Network Request Thresholds

The Mobile Agent uses the following default values to determine the severity level:

- **Warning**: greater than 3000 ms
- **Critical**: greater than 5000 ms

You should set custom thresholds best suited for your mobile applications.

Configure Application Not Responding Thresholds

1. From the Mobile App Group Configuration page, click the Settings tab.
2. In the Configure Application Not Responding Thresholds section, set the thresholds for Warning Threshold and Critical Threshold in milliseconds.
3. Click Save.
Correlate Business Transactions for Mobile RUM

On this page:
- Requirements
- App Server Agents Supporting Business Transaction Correlation
- Benefits of Correlating Business Transactions
- How It Works
  - Enable Business Transaction Correlation
  - View Business Transaction Correlation

Related pages:
- Business Transactions

You can correlate mobile network requests with business transactions. In actuality, the correlation is made between instances of network requests (network request snapshots) and instances of business transactions (transaction snapshots).

The correlation enables you to map network requests made by your mobile application with an underlying backend application. Business transaction correlation is disabled by default.

Requirements

To correlate business transactions, you are required to do the following:

- purchase licenses for Application Performance Monitoring (APM) and End-User Monitoring (EUM)
- instrument an application with an app agent
- instrument a mobile application with a mobile agent
- enable business transaction correlation in the Controller UI

App Server Agents Supporting Business Transaction Correlation

To correlate business transactions, your business application must have one of the following App Server Agents installed:

- Java Agent
- .NET Agent - only .NET Framework is supported
- Node.js Agent
- PHP Agent
- Python Agent

Benefits of Correlating Business Transactions

By correlating business transactions with network request snapshots, you can identify potential issues with the backend application that are causing bad user experiences. For example, you might find that a server error or a database query is causing a slow or very slow user experience.

How It Works

When an end user takes an action with your mobile application that requires a network request:

1. The app agent does the following:
   - sends HTTP headers identifying the business transaction to the end user's mobile device.
   - aggregates backend metrics and sends them along with the business transaction identifiers to the Controller. This serves as the content for the transaction snapshot.
2. The Mobile Agent sends metrics and the business transaction identifiers (from the HTTP header) to the EUM Server. This serves as the content for the network request snapshot.
3. The Controller fetches the metrics and business transaction identifiers from the EUM Server and then uses the business transaction identifiers to correlate the network request snapshot with the transaction snapshots.
Enable Business Transaction Correlation

You need to configure the Controller to correlate business transactions. The Controller will map the business transactions with the network request snapshots based on the process described above in How It Works.

To enable business transaction correlation:

1. From the Application Dashboard, click Configuration.
2. Click User Experience App Integration >.
3. From the Business Transaction Correlation tab, check the Enable Business Transaction Correlation check box.

Specify Business Transactions to Include Correlation Headers

You can also specify which business transactions will include or exclude correlation headers. If you do not add request match rules or request exclude rules, correlation headers will be added to all requests.

To add a rule:

1. Click the plus icon to open the Create HTTP Request Match Rule dialog.
2. From the Create HTTP Request Match Rule dialog:
   a. Check the Method checkbox and select an HTTP method that you want to match. If you do not select an HTTP method, the rule will be applied to all HTTP requests.
   b. Check the URI checkbox and enter your criteria.
   c. Click Save to save the match rule.
3. From the Business Transaction Correlation tab, click Save.

View Business Transaction Correlation

There are several ways to navigate from a browser snapshot to its correlated business transaction. The following steps show you one possible way.

1. From the Mobile App Dashboard, click Network Requests.
2. Click Snapshots.
3. You should now see network transaction snapshots as shown below:

4. Double-click one of the browser snapshots to open the Network Request Snapshot Details dialog. If a correlated transaction snapshot exists, you will see a link in the Business Transactions section as shown here:
5. You can then click links in the transaction snapshot or transaction snapshot itself to view corresponding pages in APM.
Monitor Your Applications with Mobile RUM

Mobile RUM presents your application information in five ways:

- As an overview dashboard, with tabs for
  - widgets for visually displaying common metrics
  - map-based performance display
  - usage statistics
- As a detailed list of mobile sessions, following users as they interact with your application
- As a detailed list of network requests with dashboards
- As a dashboard display of application crash trends and detailed snapshots of individual crashes with stack traces
- As custom data, showing specific data you want the agent to collect for you

Mobile App Dashboard

This view is good for getting a high-level understanding of how your app is performing overall.

Mobile RUM Overview

The Overview tab is made up of a set of widgets showing common usage metrics and crash information.

For more information, see Mobile App Dashboard.
Mobile RUM Geo Dashboard

The Geo Dashboard provides high-level insight into how your application is performing across the world. The size of a dot indicates the number of network requests from that region, and the color indicates the average time per network request in that region (red is slow). You can click to drill down to areas of specific interest. You can also see the same information presented in tabular form by clicking the grid icon in the upper left of the panel.

For more information, see Mobile App Dashboard.

Mobile Usage Statistics

Graphic representations of overall usage statistics by device, carrier, OS version, connection type, and version are found under this tab.

For more information, see the Mobile App Dashboard.

Sessions
Tracking a user's entire interaction—which screens in the app they visit, and how long that visit lasts—can give you deep insight into user behavior and can help you fine-tune and improve your user experience. Mobile sessions group a user's path over time through your app, collecting crucial data on each screen view they visit in the course of a usage session. The data is stored in the AppDynamics Events Store, so it is a record of all your users' interactions with your app. Using a UI similar to that of Network Request Analyze, you can focus on just the data you need and visualize in multiple ways.

For more information, see Mobile Sessions.

Network Requests

This view is good for tracking the performance of individual request types and understanding any issues that may be emerging.

The Network Requests List

A network request is an HTTP request from an end user's instance of your application to a server-side application that is designed to accomplish a particular task. Use this list to see how each request type is behaving in general. You can also use this list to click through to a dashboard with detailed visualizations of that request's overall performance.
Network Request Analyze

Mobile RUM collects data on every network request that your mobile app makes, and it also takes detailed snapshots periodically in case performance issues have been detected. Over time, the network request metrics are rolled up, based on averages and percentiles. But sometimes you want to see results based on all the data. With Analyze, every single request is collected and stored in the Events Service. Using the Analyze tab you can see results based on this cumulative data in the Network Request Snapshots widget and you can see the results represented visually in charts.

For more information, see Network Request Analyze.
The Network Requests Snapshots List

When an individual network request experiences slow response times or has a network or HTTP error associated with it, a detailed snapshot is taken. Use this list to scan problem requests, and, using filters, begin to see where your issues may lie. You can also use this list to click through to the details of the snapshot, including, if your application server is instrumented, timings for business transactions on the server side that were involved in the request and links to any related transaction snapshots that may exist.

For more information, see Network Request Snapshots List.

Crashes

This view allows you to investigate crash trends and individual crashes.

The Crash Dashboard

The Crash Dashboard tab aggregates mobile application crash data over time, using the Events Service. This store contains all the data collected by the Mobile Agent. The dashboard shows crash trends, sets of unique crashes, and distribution of usage types and location.

For more information, see Crashes.
Crash Analyze

Crash Analyze provides a Crash Snapshots widget that lists records with detailed information on an individual crash. In this widget, you can sort and filter crash snapshots as well as select fields to view in a grid. The other widgets enable you to visualize crash data or view lists of key data points.

For more information, see Crash Analyze.

Custom Data

The iOS and Android SDKs allow you to customize the data you collect from your apps. You can add Timers & Metrics and Custom Metrics.

You can also view Info Points to watch the execution of specific methods. You set the info points when instrumenting your mobile application.
For more information, see Custom Data.
Mobile App Dashboard

On this page:
- Access the Mobile App Dashboard View
- Overview Dashboard
- Geo Dashboard
- Usage Stats

The Mobile App Dashboard gives you a high-level overview of how your application is performing, including the following:

- A widget-based overview
- A geo dashboard, displaying where your requests originate, including a variety of key performance metrics
- A set of aggregated usage statistics, by device, carrier, connection type, OS version, and app version

Access the Mobile App Dashboard View

1. Open the application in which you are interested.
2. From the left navigation bar of your application, select Mobile App Dashboard.
3. Click the tab for the view you want to access.

Overview Dashboard

The Overview Dashboard displays a set of configurable widgets, showing multiple graphs and lists featuring common high-level indicators of application performance. You can delete widgets, re-add them, resize them, and drag them to different locations.

Geo Dashboard

The Mobile App Geo Dashboard displays key performance metrics by geographic location.

The dashboard is divided into three panels:

- A main panel in the upper left that displays geographic distribution of mobile users on a map, if you clicked the map view icon, or on a grid, if you clicked the grid view icon.
- A panel on the right displaying key timings.
- A lower panel with dynamic trend graphs of KPIs.

The metrics displayed throughout the geo dashboard are for the country currently selected on the map or in the grid. For example, if you zoom down from the world view to France on the map, the panel on the right disappears and the trend graphs display data for France.

See Mobile RUM Metrics for definitions of the metrics.

Map View Labels

The map view displays load circles with labels for countries that are in the key timing metrics given in the right panel. Some countries and regions, however, are only displayed in the grid view.

Regions and Countries

Regions are subdivisions of a country, such as a state, province, or city. In the default map view, key performance metrics are displayed by country. The grid view, like the map view, by default displays key performance metrics by country, but can also be configured to show the metrics by region. Because the map view displays fewer regions than the grid view, if you do not see a region displayed in the map view, switch to the grid view.

See Browser RUM Countries and Regions by Geo Dashboard for a list of the countries and regions available in the map and grid views.

Unknown Locations in Map and Grid Views

An unknown location is one for which the agent cannot determine the country from which the request originated.
You may also see metrics reported for a location named Anonymous Proxy. The data for Anonymous Proxy represents the aggregated metrics from one or more private IP addresses that the agent cannot identify.

One of the effects of Unknown regions is that it possible for a country to display as slow (red circles) on the global map, but when you drill down to the country all its regions appear normal (green circles). Or a country may display as normal on the global map, but some subregions may display as slow when you drill down.

Usage Stats

This tab shows graphical and tabular breakdowns of usage by device, carrier, connection type, OS version, and app version. The breakdowns for the connection type are based on network requests. The other usage breakdowns are based on "app opens", not network requests. App opens are counted when a mobile application is booted or brought to the foreground. In the case of Android, the app opens occur when the method `onResume` is called for an activity. For iOS, the app opens occur when the method `applicationDidBecomeActive` is called.

The Android Agent obtains the carrier name by calling the Android SDK method `android.telephony.TelephonyManager.getSimOperatorName` and obtains the device name with the Android SDK property `android.os.Build.MANUFACTURER`. If the value for either the carrier or device name is "unknown", this indicates that the method `getSimOperatorName` or the property `MANUFACTURER` did not return a value.
Mobile Sessions

In most cases, a user’s interaction with your app is not limited to one action in one screen view. Sessions allow you to track your users’ interactions across time, as they navigate an entire session with your app.

By default, sessions begin when a user starts using your application and ends after a configurable period of user inactivity. You can also, however, programmatically control sessions through the Mobile Agents SDKs (Android, iOS, Xamarin, Cordova). Using Mobile Sessions you can analyze sessionized results from all requests, as stored in the AppDynamics Events Service.

As in Network Request Analyze, the main Sessions screen is made up of two tabs:

- Records
- Charts

You can also view detailed information about each session in the Session Details dialog.

Records

The Records tab lets you scan individual sessions and allows you to filter and sort to get exactly the data in which you are interested.

Session Details

Click View Details or double-click an item to see the information for a specific session.
Session Timeline

The sequence of session frames for the session is shown on the session timeline and on the bottom of the screen. Select a specific frame by clicking on an icon on the timeline to see details highlighted at the bottom of the screen. You can also select a frame row at the bottom to see it highlighted on the timeline.

Session Details Categories and Types

<table>
<thead>
<tr>
<th>Categories</th>
<th>Subcategories</th>
<th>Type</th>
<th>About</th>
</tr>
</thead>
<tbody>
<tr>
<td>Views</td>
<td>Activity</td>
<td></td>
<td>The current Activity in the foreground of an Android app. If the current Activity is not sufficient to understand what the user is doing, consider using the Session Frame APIs.</td>
</tr>
<tr>
<td></td>
<td>Root Views</td>
<td>RootView</td>
<td>The current Root View in the foreground of the iOS App. If the current Root View is not sufficient to understand what the user is doing, consider using the Session Frame APIs.</td>
</tr>
<tr>
<td></td>
<td>Session Frames</td>
<td>Session Frame</td>
<td>A manual API to label large user activities that are not captured out of the box by Activity or RootView tracking.</td>
</tr>
<tr>
<td>Screenshots</td>
<td></td>
<td>Screenshots</td>
<td>If you have configured mobile screenshots, they will appear in the timeline. They capture user events, and they can also be collected when in automatic mode.</td>
</tr>
<tr>
<td>User Interactions</td>
<td>Touches</td>
<td>Event</td>
<td>If you have enabled the Mobile Agent to capture UI events, you can view the UI events triggered by user interactions in the Timeline. See &quot;Enable User Interaction Capture Mode&quot; (iOS/Android) to learn how to capture UI events. If automatic screenshots are enabled, touches will be captured and include the screenshot that was captured at the same time. The touches are drawn on the screenshot.</td>
</tr>
<tr>
<td></td>
<td>Buttons</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Table Cells</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Text Views</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Text Fields</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network Requests</td>
<td>Network Requests Normal</td>
<td>Network Request Normal</td>
<td>Network requests include the requested URL, duration, and status details of the HTTP request. To jump to more Network Requests details, click .</td>
</tr>
<tr>
<td>Network Requests Slow</td>
<td>Network Requests Slow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network Requests Very Slow</td>
<td>Network Requests Very Slow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network Requests Stall</td>
<td>Network Requests Stall</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Errors And Warnings

<table>
<thead>
<tr>
<th>Application Not Responding Issues (ANRs)</th>
<th>ANR</th>
</tr>
</thead>
<tbody>
<tr>
<td>You can view when the application was non-responsive in the timeline. ANRs are reported when the UI thread does not respond to an event for two seconds or more.</td>
<td></td>
</tr>
</tbody>
</table>

To download or view the ANR Summary, click download or view respectively.

<table>
<thead>
<tr>
<th>Crashes</th>
<th>Crash</th>
</tr>
</thead>
<tbody>
<tr>
<td>When an app crashes the subsequent activity will appear in the timeline.</td>
<td></td>
</tr>
</tbody>
</table>

To download or view the Crash Summary, click download or view respectively.

<table>
<thead>
<tr>
<th>Errors</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>If you have reported errors with the Mobile Agent SDKs, the errors will also appear in the timeline. See &quot;Report Errors and Exceptions&quot; (iOS/Android) to learn how to instrument your mobile apps to report errors and exceptions.</td>
<td></td>
</tr>
</tbody>
</table>

To download or view the Error Summary, click download or view respectively.

### Custom Data

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Custom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom metrics display integer-based data on any metric you define in your application.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Timers</th>
<th>Custom</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom timers display how long something takes between any arbitrary start point and end point, even if they span multiple methods.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Info Points</th>
<th>Info Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information points display information on how a specific method in your code is performing.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Breadcrumbs</th>
<th>mb Breadcrumb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadcrumbs are short messages to mark interesting events in a session, controlled by the Breadcrumb APIs in the agent.</td>
<td></td>
</tr>
</tbody>
</table>

### System Events

<table>
<thead>
<tr>
<th>Connection Transition</th>
<th>System Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection transition indicates when a broadband transition has taken place, such as from Wifi to cellular.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Agent Init</th>
<th>System Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent Init indicates when the Mobile Agent was first initialized. It can help with debugging the application lifecycle.</td>
<td></td>
</tr>
</tbody>
</table>

### View Options

You can filter the categories in your timeline view by clicking View Options. Check all relevant categories for your search. All view options are checked by default.
Search Events

You can search for a specific event Description using the search box located in the top right of the session timeline screen.

Session Summary

You can see a summary of the session by clicking Session Summary.
Business Transactions

You can see a list of business transactions in the session if applicable. This tab only appears if you have business transactions in the selected session.

Click the business transaction name to redirect to the transaction flow map.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/cart/co.GET</td>
</tr>
<tr>
<td></td>
<td>/cart/{id}.GET</td>
</tr>
</tbody>
</table>

Charts

The Charts page provides you with a set of predefined widgets that offer visualizations of the data set you have created. As with the Charts pages of the other Analyze/Session features of the UI, you can delete, re-add, resize, and drag-and-drop to move all of the widgets.
Active Sessions Over Time
Add a graph showing active sessions within the selected time range.

Session Duration Time Distribution
Add a graph showing session duration times within the selected time range.

Sessions By Region
Add a geographical map showing sessions counts as well as the top five locations.

Device Name
Add a bar chart showing sessions by device names.

Device Manufacturer
Add a bar chart showing sessions by device manufacturers.

Application Version
Add a bar chart showing sessions by Application Versions.

Carriers
Add a bar chart showing sessions by Carrier.

OS Versions
Add a bar chart showing sessions by OS Versions.

Connection Types
Add a bar chart showing sessions by connection types.

Crashes By App Version - Top 5
Add a table showing top 5 crashed sessions by app versions.

Crashes By Device Name - Top 5
Add a table showing top crashed sessions by device name.

Crashes By Device Manufacturer - Top 5
Add a table showing top 5 crashed sessions by device manufacturer.

Crashes By Carrier - Top 5
Add a table showing top 5 crashed sessions by carrier.

Crashes By Connection Type - Top 5
Add a table showing top 5 crashed sessions by connection type.

Crashes By OS Version - Top 5
Add a table showing top 5 crashed sessions by OS versions.

Active Users By Region
Add a geographical map showing active users in session counts as well as the top five locations.

Countries By Session Count - Top 5
Add a table showing top 5 countries by session count.

Countries By Active Users - Top 5
Add a table showing top countries by active users.
Mobile Screenshots

On this page:
- AppDynamics Version Requirements
- Screenshot Limitations
- Types of Screenshots
- When Screenshots Are Taken

Mobile screenshots provide context to the data collected in the Controller UI. They also allow you to better understand the user experience.

Screenshots enable you to do the following:

- **Verify Presentation**: Determine if an app is displaying correctly on different devices.
- **Understand User Behavior**: Determine where users are clicking on the screen.
- **Troubleshoot Bugs**: Analyze bugs and errors by correlating code issues with screenshots. You’ll be able to see what the user is doing when issues happen.

AppDynamics Version Requirements

You are required to use the following AppDynamics software versions:

- Controller 4.3.2+
- iOS/Android Agents 4.3.2+
- AppDynamics Cordova Plugin 1+

Screenshot Limitations

Mobile screenshots have the following limitations:

- Only iOS/Android/Cordova platforms are supported.
- You can enable or disable screenshots or explicitly exclude screenshots for specific screens/activities.
- You cannot set a minimum/maximum number of screenshots per session.
- Touch points are not collected unless screenshots are in automatic mode.
- The touch points will not be included in screenshots from past sessions.
- The UI Event feature is not related to screenshots or touch points in any way. Thus, the screenshot will not be directly correlated with a UI event in the Waterfall.

Types of Screenshots

Mobile RUM captures conventional screenshots and touch-point screenshots that capture user touch events. Both types of screenshots have timestamps, but the latter will also plot “user touches” on the screen.

<table>
<thead>
<tr>
<th>Conventional Screenshots</th>
<th>Touch Point Screenshots</th>
</tr>
</thead>
</table>
The conventional screenshot will simply show what your users viewing. These screenshots can be taken programmatically or by enabling automatic screenshots in the Controller UI.

The touch point screenshot shows where users touched the screen when using your app. Touch points are only collected when automatic capture mode is enabled.

When Screenshots Are Taken

Screenshots can only be captured if the Mobile Agent and the Controller UI have enabled screenshots.

Once screenshots have been enabled, screenshots will be only taken if one or more of the following are true:

- The Mobile Agent uses the SDK API to manually take a screenshot.
- You have enabled automatic capture in the Controller UI.
Configure Mobile Screenshots

You have several methods for controlling when screenshots are taken. The following section lists the settings for disabling/enabling and taking screenshots.

**Screenshot Control Settings**

The table below lists the different settings for screenshots, the user tasks required for the settings, and a description of what the settings accomplish. Screenshots are enabled by default in the iOS and Android Agents, but disabled in the Controller UI. Thus, to use screenshots, you need to enable screenshots in the Controller UI.

<table>
<thead>
<tr>
<th>Screenshot Setting</th>
<th>User Tasks</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disable Screenshots From Being Taken</td>
<td>Perform one of the following:</td>
<td>Only Disable Screenshots with the iOS/Android SDK</td>
</tr>
<tr>
<td></td>
<td>Disable screenshots with the iOS/Android SDK.</td>
<td>Screenshots are globally disabled. The Controller UI cannot override the Mobile Agent setting.</td>
</tr>
<tr>
<td></td>
<td>Disable screenshots in the Controller UI.</td>
<td>Only Disable Screenshots with the Controller UI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The Controller will disable screenshot capture in the Mobile Agent. Because the Controller setting is propagated to the Mobile Agents through the EUM Server, the Mobile Agent may take screenshots before its screenshot setting has been updated. Once the Controller setting has been applied, however, the Mobile Agent will no longer take screenshots.</td>
</tr>
<tr>
<td>Enable Screenshots</td>
<td>☑ Enable screenshots in the Controller UI.</td>
<td>The Mobile Agent can take screenshots, and the Controller UI will display them in the session results.</td>
</tr>
<tr>
<td>Enable Screenshots - Manual Screenshots (Wi-Fi)</td>
<td>☑ Enable screenshots in the Controller UI.</td>
<td>You call the Mobile SDK to take screenshots. The screenshots are only uploaded to the Controller when the device is using Wi-Fi.</td>
</tr>
<tr>
<td></td>
<td>☑ Manually take screenshots using the iOS/Android SDKs.</td>
<td>Screenhots are automatically taken periodically. The screenshots are only transmitted when the device is using Wi-Fi.</td>
</tr>
</tbody>
</table>

[Image 183 to 195]
<table>
<thead>
<tr>
<th>Enable Screenshots - Manual Screenshots (Wi-Fi / Cellular Data)</th>
<th>You call the Mobile SDK to take screenshots.</th>
</tr>
</thead>
<tbody>
<tr>
<td>✅ Enable screenshots in the Controller UI.</td>
<td></td>
</tr>
<tr>
<td>✅ Configure the Controller UI to allow the use of cellular data.</td>
<td></td>
</tr>
<tr>
<td>✅ Manually take screenshots using the iOS/Android SDKs.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enable Screenshots - Automatic Screenshots (Wi-Fi / Cellular Data)</th>
<th>Screenshots are automatically taken periodically.</th>
</tr>
</thead>
<tbody>
<tr>
<td>✅ Enable screenshots in the Controller UI.</td>
<td></td>
</tr>
<tr>
<td>✅ Configure the Controller UI to automatically take screenshots.</td>
<td></td>
</tr>
<tr>
<td>✅ Configure the Controller UI to allow the use of cellular data.</td>
<td></td>
</tr>
</tbody>
</table>
Take Mobile Screenshots

On this page:

- Enable/Disable Screenshots
- Enable Automatic Screenshots
- Programmatic Take Screenshots

Related pages:

- Customize the iOS Instrumentation
- Customize the Android Instrumentation

The sections below show you how to enable/disable screenshots and take manual/automatic screenshots.

**Enable/Disable Screenshots**

The following sections show you how to enable or disable screenshots. If you want screenshots taken automatically, see Enable Automatic Screenshots.

Mobile screenshots are disabled by default for security and privacy reasons as screenshots may contain sensitive information.

**Use the Mobile SDKs to Disable/Enable Screenshots**

Mobile screenshots are enabled by default in the iOS and Android Agents. To disable screenshots, see the iOS SDK and Android SDK documentation.

**Configure the Controller UI**

1. Navigate to one of your mobile apps in the Controller UI.
2. Open the **Configuration** page.
3. Click **Mobile App Group Configuration**.
4. Scroll down to **Configure Mobile Screenshot** and check the **Enable Mobile Screenshot** check box.
4. (Optional) If you want to allow the application to take screenshots when using cellular data, check the User Cellular Data checkbox.

Enable Automatic Screenshots

Configure the Controller UI to Automatically Take Screenshots

1. Navigate to one of your mobile apps in the Controller UI.
2. Open the Configuration page.
3. Click Mobile App Group Configuration.
4. Scroll down to Configure Mobile Screenshot and confirm that the Enable Mobile Screenshot checkbox is checked.
5. Check the Enable Automatic Screenshots check box.

6. (Optional) If you want to allow the application to take screenshots when using cellular data, check the User Cellular Data checkbox.

Programmatically Take Screenshots

You can use the Android or iOS SDK to programmatically take screenshots. See the following pages to learn how:
- Android SDK: Configure and Take Screenshots
- iOS SDK: Configure and Take Screenshots
View Mobile Screenshots

On this page:
- View When Screenshots Were Taken
- View the Screenshots of an Activity

Mobile screenshots are included in the Session Details. You can view a timeline of the screenshots were taken during a user activity or all the screenshots taken during a user activity. The following sections detail how to view screenshots in the timeline of the Waterfall tab and how to view all the screenshots take for an activity from the Screenshots tab.

View When Screenshots Were Taken

Both conventional and touch-point screenshots will appear in the Waterfall tab at the time that it was taken.

If you click the camera icon in the waterfall, the screenshot will appear in a small dialogue as shown below.
View the Screenshots of an Activity

To view all the screenshots for an activity (screen), select the activity and then click the Screenshots tab. (The Screenshots tab will be empty if no screenshots were taken.)

Screen View Details - OffersView

<table>
<thead>
<tr>
<th>Timeline</th>
<th>Screenshots</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Taken at -3,077,172,291,300 ms

Taken at -1,538,586,145,650 ms

From the Screenshots tab, click one of the screenshots to view a larger version of the screenshot in a dialogue.
You can set mobile health rules to trigger actions when certain conditions are met or exceeded based on configured thresholds for mobile application metrics.

To configure alerts and responses, you can use the Getting Started Wizard or manually create alerting policies based on mobile health rules. Alerts and responses help you to anticipate and take action for problems with your mobile applications.

In the following sections, you will learn about the two types of mobile health rules, how to select mobile health rule types, create new mobile health rules, and finally, how to view the health results of your mobile applications.

Mobile Health Rule Types

When you select the health rule scope for mobile applications, you are given a set of health rules based on the two health rule types Mobile Apps and Network Requests. You can create new health rules using the mobile health rule types or use a custom health rule type.

Mobile Apps Health Rule Type

The Mobile Apps health rule type can be used to add conditions based on a wide range of mobile application metrics such as application starts, crashes, or network requests for the application. The network request metrics for the Mobile Apps health rule type refer to all the network requests of the application. For example, when you configure a threshold for the metric Slow Network Requests, the Mobile App type health rule will evaluate all the network requests of the mobile application in determining which network requests are slow.

Network Requests Health Rule Type

The Network Requests health rule can only be used to add conditions based on network-related metrics such as HTTP errors, network errors, network request time, and network requests per minute. The network request metrics for the Network Requests health rule type refer to specific network requests. For example, when you configure a threshold for the metric Slow Network Requests, the Network Requests type health rule will evaluate one network requests, such as calls to http://google.com in determining which network requests are slow.

Custom Health Rule Type

The Custom health rule type allows you to use any metric. This enables you to set health rules that have metrics from different health rule types. For example, you could have a custom health rule that is based on configured thresholds for mobile metrics and business transaction metrics.

Create Health Rules

The process and user flow for creating health rules for mobile applications are similar to creating health rules for other applications. Thus, if you haven't created policies or health rules before, see Configure Policies and Configure Health Rules for general instructions. The sections below are not going to cover creating policies, but instead focus strictly on the important differences when creating health rules for mobile applications.

The first example will show you how to use the default set of mobile health rules, modify one of the default mobile health rules, and
finally, how to create a new health rule using one of the mobile health rule types.

**Using Default Health Rules**

When you use the health rule scope for mobile applications, you are given a set of default health rules based on mobile health rule types. To enable these default health rules, click Alert & Respond > Health Rules to see the list of default health rules for mobile applications. You can use these default health rule types for your mobile applications by checking the Evaluate Health Rules checkbox as shown below.

![Health Rules](image)

**Modify Default Health Rules**

The default health rules for mobile applications are based on mobile health rule types and have default conditions based on preset values for mobile metrics. You may want to retain the default health rules, but want to customize the conditions for your mobile application. To do this, click the default health rule you want to modify and navigate to either the Critical Warning or Warning Condition panel. You can then change the parameters of the existing conditions or add new conditions.

**Select Mobile Applications for the Health Rules**

You can choose whether to apply the health rules for mobile applications. Depending on the mobile health rule type, you have different criteria to choose when to apply the health rules.

For health rules based on the health rule type **Network Requests**, you can apply the health rules for:

- all network requests in the mobile app group
- specified network requests
- network requests matching criteria
- network requests for specific mobile applications

For health rules based on the health rule type **Mobile Apps**, you can apply the health rules for:

- all mobile applications in a mobile app group
- specified mobile applications
- mobile applications matching criteria

**Create New Health Rules**

You may want to create additional health rules based on one of the mobile health rules or a custom health rule. You do this in the same way you would create any health rule, except from the Create Health Rule dialog, you would select either the mobile health rule type **Mobile Apps** or **Network Requests** as shown below.

In addition to getting alerts when mobile applications are unhealthy, you can also view the health of your mobile application from certain widgets the **Mobile App Dashboard** and in the **Events** and **Health Rule Violations** tabs.

**Overview Widgets**

From the **Mobile Dashboard**, you should see the **Mobile App Health** widget. The widget displays health rule violations, crash metrics, as well as the network request health.
### Mobile App Health

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile App HR Violations</td>
<td>14</td>
</tr>
<tr>
<td>Network Request HR Violations</td>
<td>0</td>
</tr>
<tr>
<td>New Crashes</td>
<td>0</td>
</tr>
</tbody>
</table>

### Crash Metrics

- **Crash Rate**: 5.8%
- **Crashes**: 919

### Code Issues Metrics

- **Critical Code Issues**: -

### HTTP Error Metrics

- **HTTP Error Rate**: 0.0%
- **HTTP Errors**: 0

### Network Request Health

- **Critical**: 0
- **Warning**: 0
- **Normal**: 8

### Request Scorecard

<table>
<thead>
<tr>
<th>Speed</th>
<th>Normal</th>
<th>Slow</th>
<th>Very Slow</th>
<th>Stall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>86.4%</td>
<td>2.8%</td>
<td>10.6%</td>
<td>0.1%</td>
</tr>
<tr>
<td></td>
<td>24.06k</td>
<td>793</td>
<td>2.95k</td>
<td>29</td>
</tr>
</tbody>
</table>

## Events

The **Events** tab displays the health rule violations for your mobile application. See [Monitor Events](#) for more information.
You can sort the health rule violations for your mobile application by event types, health rules, or custom events you created.

**Health Rule Violations**

The **Health Rule Violations** tab lists the health rule violations that have happened over the specified time or that are still open.

Double-clicking one of the health rule violations in the list opens the **Health Rule Violation Details** dialog showing the type of health rule violation, the timeline, the events, and a description of the violation.
Network Requests

A network request is an HTTP request from your mobile app to a server-side application. The iOS Agent detects network requests when the underlying implementation is handled by the `NSURLSession` classes. The Android Agent detects network requests when the underlying implementation is handled by the `HttpURLConnection`, `HttpClient`, `OkHttp`, or `ch.boye.httpclientandroidlib.Connection` classes.

You can use the agent SDK to set up other HTTP classes if desired.

View Network Requests

There are different ways of viewing network request data in the Network Requests View:

- The Network Requests list displays current network request types to your applications. You can sort the list according to key metrics such as the slowest response time, highest error rate, highest load, etc. You can view a network request dashboard that summarizes aggregate performance for a specific network request type.
- Network Request Analyze allows you to sort and filter a store of all the network request data your agents have collected and to see visualizations based on that data.
- A Network Request Snapshot reports information on an individual instance of a network request. Snapshots are useful for examining the details of the worst-performing requests. Access these snapshots from the network request snapshots list.

Network request data is also displayed in the Mobile App Dashboard.

- The Overview tab has the following widgets showing information about network requests:
  - Network Request Scorecard: Displays the number and percentage of normal, slow, very slow, and stalled network requests. Clicking on the widget opens the Network Requests list.
  - Network Request Health: Displays the number of network requests evaluated as normal, critical, or warning based on default or configured health rules. Clicking on the widget opens the Network Requests list.
  - Network Request Time By Country: Displays a heat map of the world based on the number of network requests. Clicking on the widget opens the geographic view.
  - Requests Per Minute: Displays a bar graph showing the number of requests per minute over the specified time period. Clicking on the widget opens the geographic view.
  - Network Requests Time Distribution: Displays a bar graph showing the number of requests made at different network request times. In addition, the graph gives the percentage rank of the number of network requests in its frequency distribution of network request time. For example, the 95th percentile (5,419 ms) indicates that 95% of the network requests had a network request time of 5,419 ms or less. Clicking on the widget opens the Network Requests list.
  - Network Request Time Trend: Displays a line graph showing the average network response time over a specified time period. Clicking on the widget opens the geographic view.
  - HTTP Errors: Displays the number and rate of HTTP errors. Clicking on the widget opens the Network Requests list.
  - HTTP and Network Errors Trend: Displays a line graph comparing the number of HTTP and network errors over a specified time period. Clicking on the widget opens the geographic view.
- The geographic view reports aggregated mobile data by geographic location. Monitor the geographic view to learn which countries have the highest number of requests, the longest request times, and the most errors.
- Usage stats display key network request metrics by various criteria: device, carrier, operating system version, connection type, and application. For example, you can see which carriers are the slowest or which devices are producing the most errors.
Access the Network Requests View

1. Open the application in which you are interested.
2. In the left navigation bar, select **Network Requests**.
3. Click the tab for the view you want to access.

The Controller processes a maximum of 2000 network requests per mobile app group and 500 network requests per mobile application. See **Network Request Limits** for suggestions on how to configure network request detection to stay under this limit.
Network Requests List

On this page:
- How the Network Requests List is Organized

Related pages:
- Network Request Dashboard
- Network Request Limits
- Mobile RUM Metrics

The **Network Requests** list shows all the network requests types from your instrumented mobile application, along with their key performance indicators.

**How the Network Requests List is Organized**

The **Network Requests** list is a table that displays aggregated metrics for current requests, with one row for each request type. The columns display the name of the network request and the aggregated metrics associated with it. See Mobile RUM Metrics for descriptions of these metrics.

Click a column header to sort the list based on the column's metric. For example, if you want to sort by the slowest requests, click the Network Request Time (ms) column header. You can toggle the column to switch between ascending and descending order. Use View Options to configure the table. Check With Load if you want to show only network request types that have experienced active load, that is, those for which there were one or more measured network requests over the selected time period.

**To view a network request dashboard**

1. Select the network request type in the list.
2. Either click Details or just double-click the network request.

**More Actions Menu**

Use the More Actions menu to select one or more requests in the list and perform the following actions on them.

- **Exclude Request(s)** Use this option to direct AppDynamics to ignore the selected request(s) and stop reporting metrics for them. You can use the View Excluded Requests option to see requests that have been excluded and then you can "un-exclude" them.
- **Rename Request** Use this option to rename one selected request in the AppDynamics console.
- **Delete Request(s)** Use this option to remove the request(s) from the list. If AppDynamics discovers a deleted request again it will reappear in the list. To prevent it from re-appearing, use Exclude Request(s).
Network Request Dashboard

On this page:
- How the Network Request Dashboard is Organized

Related pages:
- Network Requests
- Network Requests List
- Network Request Analyze
- Mobile RUM Metrics

Each network request type has its own dashboard that graphically displays key performance indicators for that type over the selected time range. To select the time range, use the general time range dropdown at the top right of the UI.

How the Network Request Dashboard is Organized

The Network Request Dashboard displays summary key network request metrics for the time selected. To see any particular metric in the metric browser, click the metric value (in link blue).

The trend graphs for the key performance indicators are:
- **Network Request Time**: Average times in milliseconds.
- **Total Server Time**: Displayed only if the mobile request is correlated with a server-side application. The total server time is the interval between the time that the server-side application receives the network request to the time that it finishes processing the request. Use this graph to determine, on average, how much time is spent on the network versus how much time is spent on the server to process the user's request.
- **Load**: Total Requests and Requests per Minute.
- **Errors**: Network Errors and HTTP Errors in total and per minute.
- **Related Business Transactions**: If the request is correlated with a server-side application, the dashboard lists business transactions associated with the request below the performance metrics.

You can hover over any data point on any of the trend graphs to see the metric for a precise point:
Mobile RUM collects data on every network request that you have instrumented, and it also takes detailed snapshots periodically in case performance issues have been detected.

Based on network request performance data, Mobile RUM calculates metrics which are associated with each network request and with predefined aggregations across network requests. You can focus on certain aspects of the performance data using more specific or flexible criteria, such as all network requests from a specific country and originating from a specific carrier.

With Analyze, every single network request event is collected and stored by the AppDynamics Events Service. Using Analyze, you can see results based on this cumulative data in tabular form or as charts.

Double-click a network request snapshot in the Network Request Snapshots widget to see the information for a specific request.
Mobile RUM Network Request Analyze Versus Mobile Requests Analytics

The data shown in the Analyze page is processed and stored by the AppDynamics Platform Events Service. Network Request Analyze provides a number of widely used visualization types to let you explore your application's performance. A separate product, AppDynamics Mobile Analytics, has components called Mobile Requests and Mobile Crash Reports. These components are based on the same Events Service and use the same data, but offer additional capabilities, including:

- additional predefined widgets, such as the funnel widget
- ADQL for searching the data
- creating custom widgets
- manipulating multiple dashboard types
- longer retention time for storing data

AppDynamics Mobile Requests Analytics requires a license separate from the Mobile RUM license.
Network Request Snapshots List

A network request snapshot captures the details of one instance of a network request. Examine these details to troubleshoot the causes of slow network request performance.

You access the list using the **Network Request Snapshots** tab. The network request snapshots list is a table that displays a row for each network request snapshot. The columns describe the properties of each snapshot.

**To view a network request snapshot**

1. Select the network request snapshot in the list.
2. Either click **Details** or just double-click the request.
Network Request Snapshots

Network request snapshots capture detailed information about individual network request instances by your application. They can help you troubleshoot the causes of poorly performing mobile applications.

When Network Request Snapshots are Captured

Mobile RUM starts capturing snapshots when user experience becomes slow, based on how you have configured the thresholds for slow, very slow, and stalled. See Configure Mobile Network Request Thresholds.

Periodic snapshots of normal user experience are also captured at least once per minute.

Network Request Snapshot Content

A network request snapshot contains summary data about the individual request as well as any business transactions associated with the request if correlation with an instrumented server-side application is available.

The snapshot contains extensive metrics for the request in the Summary data, including:

- **User Experience**: Normal, Slow, Very Slow, or Stalled.
- **Time**: when the request was received by the EUM Cloud or EUM Server collector, in UNIX epoch time.
- **Mobile Network Request**: a link to the network request dashboard for the network request of which this snapshot is an instance. See Network Request Dashboard.

Business Transactions in Network Request Snapshots

When a network request snapshot is associated with one or more business transactions on an instrumented server-side application, the business transactions are listed in the Business Transactions panel in the network request snapshot. You can click the link to see the business transaction dashboard for the associated business transaction.

If transaction snapshots for an associated business transaction were captured at the same time as the network request snapshot, they are linked in the Transaction Snapshots panel of the network request snapshot. If a call graph icon is displayed for a snapshot in the transaction snapshot list, a full or partial call graph is available for that transaction snapshot. This allows you to examine the cause of performance problems on the server side. Click the link to see the associated transaction snapshot.

Transaction snapshots are triggered on the server when slow or stalled business transactions are identified when a diagnostic session is started or periodically based on a configured interval. In general, slow, very slow, and stalled transactions are more likely to trigger a transaction snapshot on the server than transactions operating within the normal range. For more information about when server-side transaction snapshots are captured, see Transaction Snapshots.

How to Access Network Request Snapshots from Transaction Snapshots

If a transaction snapshot seen from the server side also has generated a correlated network request snapshot, a mobile snapshot link appears in the top right of the transaction snapshot flow map. Click the link to open the network request snapshot.

Archiving Network Request Snapshots

Normally network request snapshots are purged after two weeks. You can archive a snapshot beyond the normal snapshot lifespan to retain it for future analysis.

To archive a snapshot, click the Archive button in the upper right corner of the snapshot window.

You can view archived snapshots by checking Archived as a view option in the network request snapshots list.

Customers with on-premises controllers can modify the default two-week period by configuring the event.retention.period property.
ty in the Controller Settings section of the Administration console.
Crashes

On this page:

- View Crash Information in Widgets
- Access the Crash Dashboard

Related pages:

- Crash Dashboard
- Crash Analyze
- Crash Snapshots
- Crash Snapshot Properties

Use Crashes to get trend and detailed information on mobile application crashes.

A crash snapshot is a detailed report on a particular crash including the code that was executing when the application crashed. Crash snapshots help you understand the causes of crashes.

View Crash Information in Widgets

The Crash Metrics and Events widget are on the Mobile App Dashboard. The Crash Metrics widget displays the crash rate and the number of crashes for your mobile application. The Events widget displays new crash events.

1. Open the application in which you are interested.
2. From the Mobile App Dashboard, you'll find see both the Events and Crash Metrics Scorecard widgets.
3. You can click Crash Metrics or click one of the new crash events in the Events widget to get to the Crash Dashboard.

Access the Crash Dashboard

The Crash Dashboard is where you can view both summary and detailed information about crashes.

1. Open the application in which you are interested.
2. On the left navigation bar of your application, select Crashes.
Crash Dashboard

On this page:
- Summary Crash Trend
- Unique Crashes
- Crash Status

Related pages:
- Crash Analyze
- Crash Snapshots
- Crash Snapshot Properties

The Crash Dashboard aggregates mobile application crash data over time, using the Events Service. This service collects and stores all the data collected by the mobile agent.

The Crash Dashboard is divided into two panels and has the App Version dropdown that enables you to view crash data for different versions of your application.

Summary Crash Trend

This panel displays a running total of crashes, unique crashes, impacted users, crash rate, and crash trends. The crash trends is a timeline of crash rates. It also reports any iOS crash reports that were uploaded without the accompanying dSYM file. For more information on how to use dSYM files with crashing reports, see Get Human-Readable Crash Snapshots.

Unique Crashes

Multiple crashes can be caused by the same underlying code issue. The Unique Crashes panel displays a list of crashes grouped by common characteristics and displays basic information about the crash. You can view open, closed, or all crashes.

Unique Crash Details

To see more detail per crash, click the crash that interests you, in blue. The dashboard for that crash appears, with a header, trend bar graph, crash distribution charts, and a snapshot of the crucial details common to all the crash snapshots. See the section Crash
Summary on the Crash Analyze page for more information.

**Crash Status**

In addition to viewing crash details, from the **Unique Crashes** panel, you can select a unique crash, click **Actions**, and set the status to either open or closed. You can set the status to mark those unique crashes that you want to ignore or have fixed the root cause of.

When a crash is marked as closed, the crash will no longer trigger a new crash event, so you will not see the crash in the **Events** widget of the Mobile App Dashboard, the **Events** page, and it won’t be included in alerts for new crashes.
Crash Analyze

On this page:
- Crash Snapshots Widget
- Chart and Tabular Widgets

Related pages:
- Instrument an iOS Application
- Instrument an Android Application Manually

Crash Analyze lets you view and analyze the results based on the cumulative data. With Crash Analyze, every crash event is collected and stored by the AppDynamics Events Service.

The Crash Analyze page has the Crash Snapshots widget for viewing a list of current crash snapshots, with one row for each snapshot, and a selection of different widgets for visualizing the cumulative data in charts.

**Crash Snapshots Widget**

The Crash Snapshots tab lets you scan individual crashes and allows you to filter and sort to get exactly the crash set in which you are interested. The columns of the table are fields representing the crash snapshot properties. See Crash Snapshot Properties for descriptions of these properties. Click View Details to see the information for a specific crash.
Chart and Tabular Widgets

The chart widgets provide you with a set of predefined widgets that offer visualizations of the data set you have created. The tabular widgets highlight key metric points such as top crashes. You can delete, re-add, re-size, and drag-and-drop to move all of the widgets. You can also set criteria to visualize a subset of the data.

Crash snapshots can also be viewed in the context of a separate product, AppDynamics Mobile Analytics. It offers additional mechanisms to analyze and visualize crash patterns, including:

- additional predefined widgets, such as the funnel widget
- ADQL for searching the data
- custom widgets
- manipulating multiple dashboard types
- longer retention time for storing data

AppDynamics Mobile Crash Report Analytics requires a license separate from the Mobile RUM license.
Crash Snapshots

On this page:
- Crash Snapshot Details

Related pages:
- Crash Analyze
- Crash Snapshot Properties
- Get Human-Readable Crash Snapshots

Crash snapshots provide detailed information for one application crash. When an instance of an instrumented application crashes, a crash snapshot is created.

The snapshot provides information to help you analyze the cause of the crash, including:
- the crashed function
- the source file containing the crashed function
- the line number in the source file, if available
- the stack trace of the application at the time of the crash

For iOS applications, crash snapshots are based on:
- fatal signals (SIGSEGV, etc)
- unhandled Objective-C exceptions

For Android applications, crash snapshots are based on:
- unhandled Java exceptions

**Crash Snapshot Details**

The Crash Snapshot Details dialog has the two panels Crash Summary and Crash Distribution. In short, the Crash Summary pane displays basic crash information and the stack trace. The Crash Distribution pane displays widgets with distributed crash metrics by the crash group as well as the stack trace.

**View Crash Snapshot Details**

1. From the Records tab, select a crash snapshot in the list.
2. Click View Details.

**Crash Summary**

The Crash Summary panel of the snapshot displays the key properties of the snapshot. If you have customized your instrumentation (Android/iOS) to include breadcrumbs, they are displayed in the Events Prior to Crash panel. Up to 99 breadcrumbs are displayed, from the most recent back. Sometimes not all of this information is available. If you have added user data (Android/iOS), it also is displayed here.

The Stack Trace panel displays the call stack of the crashed application, showing the thread in which the crash occurred. This is the full stack trace for this specific crash, not the aggregated stack trace shown in the Unique Crash Dashboard. You can view the formatted stack trace that groups threads and parses the output or the raw log output for the stack trace.

Click Download to grab the stack trace in a file that you can forward to developers or port to logging platforms. To see the crash in the context of the mobile session, click View Session.
If the information in the stack trace is cryptic, it is possible that source code for your iOS app was not symbolicated or the source code for your Android app was obfuscated. See Get Human-Readable Crash Snapshots for information about why this happens and what you can do about it.

Crash Distribution

The Crash Distribution panel displays smaller-sized widgets from the Charts tab with distributed metrics for the crash group as well as the stack trace.
Crash Snapshot Properties

This is the list of crash snapshot properties. They appear in the Crash Snapshot Details window when you open a crash snapshot from Crash Analyze.

- **Carrier**: name of the mobile carrier.
- **Connection Type**: active connection type at the time of the crash, if known.
- **Crashed File/Line/Function**: name of the source file containing the crashed function, with the line number, if available. The name of the topmost function on the crashed thread's call stack. If this function is an Objective-C method, this name includes the class name. For Android, this name is the fully qualified name of the topmost method on the uncaught exception's stack trace.
- **Crash Time**: timestamp when the crash occurred, based on the mobile device's clock.
- **Device/Manufacturer**: model or manufacturer name of the mobile device on which the crash occurred.
- **Events Prior to Crash**: Any breadcrumbs you might have set up.
- **Exception Name**: name of the fatal signal (iOS) or uncaught exception (Android) associated with the crash.
- **Memory Usage**: the memory used by the app when the crash occurred.
- **Mobile App**: the name of the app that crashed.
- **OS Version**: operating system version of the mobile device on which the crash occurred.
- **Platform**: iOS or Android.
- **Request Timestamp**: when the beacon arrived at the Collector. Snapshot only.
- **User Data**: Any user data you might have added.
- **Version**: the version string of the crashed application.

**Crash Snapshot list only**

- **Country**: the originating country.
- **Crash id**: the unique identifier for the crash.
- **Deobfuscated**: For Android: True if this crash report has been matched with a ProGuard mapping file and deobfuscated; false otherwise. See Upload the Proguard Mapping File in Instrument an Android Application Manually and Get Human-Readable Crash Snapshots. A false value for this property does necessarily indicate that the crash report will not be human-readable since it is possible that the application in question was not obfuscated.
- **Symbolicated**: For iOS. True if this crash report has been matched with a dSYM file and symbolicated; false otherwise. The application must have been compiled with the Debugging Information Format set to DWARF with dSYM File for a crash report to exist. See Upload the dSYM File and Get Human-Readable Crash Snapshots.
Get Human-Readable Crash Snapshots

On this page:
- iOS dSYM File
- ProGuard Mapping File for Android

Related pages:
- Crash Snapshots
- Upload the dSYM File
- Manually Upload Mapping Files

The information in most raw crash stack traces is not fully human readable. To make your crash stack traces more easily understood, you need to provide a platform-specific mapping file that can translate the raw data into human-readable output. Normally, you upload the file at the time that you instrument your mobile application.

For iOS, see Upload the dSYM File. For Android, see Manually Upload Mapping Files.

This topic explains the advantages of providing these files.

iOS dSYM File

For iOS applications, the raw data in the stack traces in crash snapshots consists of memory addresses of stack frames that point to executable application code. It also includes symbols and memory offsets for the system library code used by the application. Such a partially symbolicated stack trace looks something like this:

```plaintext
Thread 11 Crashed:
0  libobjc.A.dylib  0x38cb50fc objc_retain + 12
1  SomeApp       0x003a2204 0xc6000 + 2998788
2  SomeApp       0x003a0854 0xc6000 + 2992212
3  SomeApp       0x003a09d4 0xc6000 + 2992596
4  SomeApp       0x003948e4 0xc6000 + 2943204
5  Foundation   0x2f3a2dc2 __NSThread__main__ +
1058
6  libsystem_pthread.dylib  0x392ce5a _pthread_body + 138
7  libsystem_pthread.dylib  0x392cebca _pthread_start + 98
8  libsystem_pthread.dylib  0x392cccc thread_start + 4
```

AppDynamics attempts to display stack traces with the names of functions with offsets into those functions to help you identify the line of code that was executing when the application crashed. To get the symbols that map to the executable code, it needs the dSYM (desymbolication) file for the crashed application.

If a dSYM has been uploaded for a crashed application, in the crash list the associated crash snapshot is Symbolicated true.

If the dSYM file for the crashed application has been uploaded, the symbolicated stack trace shows the function name and the offset into the function where the app crashed. It looks something like this:

```plaintext
Thread 0 Crashed:
0  libobjc.dylib  objc_release + 0x14
1  CoreFoundation  CFStringCreateWithFormat + 0x4
2  UIKit        -[UIButton init] + 0x696
3  MyApp        createUI (MyAppDelegate.m:42)
```

The dSYM file is created when the application source code is compiled with the Debugging Information Format set to DWARF with dSYM file. AppDynamics recommends that you build all the iOS apps that you want to monitor using this option and then upload the dSYM file to AppDynamics. The best time to do this is when you instrument the app.

If a dSYM has been uploaded for a crashed application, in the crash list the Symbolicated column for the associated crash snapshot is true.
If the `symbolicated` property is false and you want to see user-friendly stack traces in your crash snapshots for this application, you need to locate and upload the dSYM file for the crashed application.

**ProGuard Mapping File for Android**

If an Android app was not obfuscated to prevent reverse engineering, you should see human-readable stack traces in your crash snapshots by default.

If the code was obfuscated, however, AppDynamics needs the ProGuard mapping file to be able to deobfuscate the app. The best procedure is to upload this file at the time you build the app.

If a ProGuard mapping file has been uploaded for a crashed application, in the crash list the `Deobfuscated` column for the crash snapshot is true.

If the `deobfuscated` property is false and the stack traces you see in the crash snapshots are obfuscated, you need to locate and upload the mapping file for the application.
Crash Alerts

On this page:

- Set Up Alerts for Crashes
- Create Email Digest for Crashes

Related pages:

- Configure Mobile Crash Alerts

You can configure AppDynamics to send you alerts when your mobile application crashes. You can configure email alerts for health rule violations based on some default health rules or by manually creating a policy with a notification action or creating an email digest.

Crashes that have the status "Closed" do not trigger alerts.

Set Up Alerts for Crashes

Setting up alerts for crashes is similar to setting up alerts for other health rule violations:

1. Configure Mobile Crash Alerts. This will configure the default number of crashes.
2. If desired, update the Critical Criteria and Warning Criteria for the health rule Crash Rate is much higher than normal. See Configure Health Rules for more information.
3. Create a policy that will be triggered when the health rule for crashes is violated. You'll need to configure an action to be taken when the policy is triggered.
4. After you set up the email alerts, when a health rule is violated, an email will be sent indicating whether there was a warning or critical alert and providing links to view crash and session details.

Create Email Digest for Crashes

You can create the same type of notification for crashes by creating an email digest.

From the Create Email Digest dialog, you check the Mobile Crash checkbox as shown below.
You can then use the **Email Digest** wizard to add recipients, a custom message, and schedule the frequency to send emails.
Code Issues

On this page:
- How to Catch and Report Code Issues
- Access the Code Issues Dashboard
- View Code Issue Widgets

Code issues consist of caught non-fatal exceptions or application not responding (ANR) issues. A code issue snapshot is a detailed report on a particular event including the code that was executing when the exception or ANR occurred.

Code issue snapshots help you understand the causes of these issues. Use Code Issues to get trend and detailed information on application non-responsive issues (ANRs) and caught errors and exceptions.

How to Catch and Report Code Issues

All the Mobile Agents except the iOS Agent will automatically detect and report ANRs. For iOS, you have to enable ANR detection. To catch and report errors and exceptions, you instrument your mobile applications with iOS, Android, and Xamarin SDKs.

See the section "Report Errors and Exceptions" (iOS, Android, Xamarin) to learn how to use the SDKs to manually catch and report errors and exceptions.

Access the Code Issues Dashboard

The Code Issues Dashboard is where you can view both summary and detailed information about code issues.

1. Open the application in which you are interested.
2. On the left navigation bar of your application, select Code Issues.

View Code Issue Widgets

The Summary Code Issue Trend and Unique Code Issues widgets are on the Code Issues Dashboard. The Summary Code Issue Trend widget displays the number of code issues, impacted users, and unique code issues as well as graph the code issues on a timeline. The Unique Code Issues widget displays details about the unique code issues such as the severity, the issue type, runtime, impacted users, and mobile app version.

To view code issue information:

1. Open the application in which you are interested.
2. From the Code Issues Dashboard, you’ll find both the Summary Code Issue Trend and Unique Code Issues widgets.
3. You can click one of the unique code issues in the Unique Code Issues widget to view the Code Issues Details dialog.
Code Issues Dashboard

On this page:
- Summary Code Issue Trend
- Unique Code Issues

The Code Issues Dashboard aggregates mobile application caught exceptions and ANRs over time, using the Events Service. This service collects and stores all the data collected by the mobile agent.

The Code Issues Dashboard shown below is divided into two panels and has the dropdown App Version to view code issues for different versions of your application.

Summary Code Issue Trend

This panel displays a running total of code issues, unique code issues, impacted users, the percent of impacted users, and code issue trends. The code issues trend is a timeline of code issues.

Unique Code Issues

Multiple code issues can be caused by the same underlying issue. The Unique Code Issues panel displays a list of code issues grouped by common characteristics and displays basic information about the issue.
Unique Issue Details

To see more detail per code issue, select the unique code issue that interests you and click Details. The Unique Issue dialog has the two tabs Issue Summary and Issue Distribution.

Issue Summary

The Issue Summary tab has two main panels. The top panel shows summary information such as the total code issues, the issue type, impacted users, the runtime, and a code issue trend bar graph.
The bottom panel shows a sequence of threads in the stack trace that you can expand for more information, a Properties section with general information about the device, and a User Data section with any set user data.

iOS

The stack trace for iOS applications shows the concatenated image and symbol names. For example, the concatenated "UIKit" (an image name) and "bar" (the symbol name) forms "UIKitbar".

Android

The stack trace for Android applications provides the class, method, and line number for each stack frame.
**Issue Distribution**

The **Issue Distribution** tab displays the same stack trace panel, but the top panel dashboard instead displays the distribution charts for the code issue.
The **Code Issues Analyze** page has the **Code Issue Events** widget for viewing a list of current code issue snapshots, with one row for each snapshot, widgets for visualizing code issues data, and widgets for highlighting key data points.

**Code Issues Analyze** shown below enables you to view and analyze the results based on cumulative data based on configured criteria. You can delete, re-add, re-size, and drag-and-drop to move all of the widgets. You can set criteria to visualize a subset of the data. Every code issue event in **Code Issues Analyze** is collected and stored by the AppDynamics Events Service.

**Code Issue Events Widget**

The **Code Issue Events** widget given below enables you to scan individual code issues, filter, and sort to get exactly the code issue in which you are interested. The columns of the table are fields representing the code issue properties. See **Code Issue Snapshot Properties** for descriptions of these properties.
Other Code Issue Widgets

The other widgets fall into the following three categories: tables, charts, a summary, and a hybrid.

Table Widgets

The table widgets display rows of code issues in tabular form. The **Code Issue Events** widget lists the code issue snapshots.

Chart Widgets

The chart widgets visualize the code issue data. For example, you can view the code issues data in pie charts showing the ratio of ANRs to caught errors/exceptions, severity levels, operating systems, etc.
Summary Widget

The Summary widget just lists totals for key statistics.

<table>
<thead>
<tr>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
</tr>
<tr>
<td>Total Code issues</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>Total Unique Code issues</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>Impacted Users</td>
</tr>
</tbody>
</table>

Hybrid Widget

The Code Issues By Region widget is a hybrid, displaying geographic data in a map and listing the data points in a table.
Code Issues in Mobile Analytics

Code issue snapshots can also be viewed in the context of a separate product, AppDynamics Mobile Analytics. It offers additional mechanisms to analyze and visualize code issue patterns, including:

- additional predefined widgets, such as the funnel widget
- ADQL for searching the data
- custom widgets
- manipulating multiple dashboard types
- longer retention time for storing data

AppDynamics Analytics Mobile Non-Fatal Issues Data requires a license separate from the Mobile RUM license.
**Code Issue Snapshots**

Code issue snapshots provide the detailed information given in the list below for one code issue. A code issue snapshot is created when an instance of an instrumented application experiences an ANR, slow UI, or manually reports a caught exception. To manually report caught exceptions, you use either the iOS SDK or the Android SDK.

- the thread that contains a function that is not responding or has a caught exception
- the source file containing the function that is not responding or has the caught exception
- the line number in the source file, if available
- the stack trace of the application at the time of the code issue

For iOS applications, code issue snapshots are based on:

- ANRs
- handled Objective-C exceptions
- slow UI

For Android applications, crash snapshots are based on:

- ANRs
- handled Java exceptions
- slow UI

**Code Issue Snapshot Details**

The Issue Snapshot Details dialog has the two panels Issue Summary and Issue Distribution. In short, the Issue Summary panel displays basic information about the code issue and the stack trace. The Issue Distribution panel displays widgets with distributed issue metrics by code issue group as well as the stack trace.

**View Code Issue Snapshot Details**

2. From the Code Issue Events widget, double-click one of the code issue snapshots.

**Issue Summary**

The Issues Summary panel of the snapshot displays the key properties of the snapshot. Unlike with crashes, code issues do not include breadcrumbs.

The Stack Trace panel displays the call stack of the code issue, showing the thread in which the caught exception or ANR occurred. This is the full stack trace for this specific code issue, not the aggregated stack trace shown in the Unique Issue Dashboard. You can view the formatted stack trace that groups threads and parses the output or the raw log output for the stack trace.
Click **Download** to grab the stack trace in a file that you can forward to developers or port to logging platforms. To see the code issue in the context of the mobile session, click **View Session**.

**Issue Distribution**

The **Issue Distribution** panel displays smaller-sized widgets from the **Analyze** tab with distributed metrics for the code issue group.
Code Issue Snapshot Properties

This is the list of code issue snapshot properties. They appear in the Code Issue Snapshot Details window when you open a code issue snapshot from Code Issue Analyze.

- **Agent ID**: The ID of the Mobile Agent reporting the code issue.
- **Agent Version**: The version of the Mobile Agent reporting the code issue.
- **Application Key**: The application key.
- **Carrier**: The name of the mobile carrier.
- **City**: The originating city.
- **Client Time**: The timestamp of a device when the beacon was made.
- **Connection Type**: The active connection type at the time of the code issue, if known.
- **Country**: The originating country.
- **Description**: The type of code issue, such as an application not responding (ANR) or a caught non-fatal exception.
- **Device/Manufacturer/Model**: The model or manufacturer name of the mobile device on which the code issue occurred.
- **Event Type**: Type of code issue (i.e. ANR, Caught Exception)
- **Jailbroken**: Flag indicating whether the mobile device has been jailbroken.
- **Group ID**: The event group id after grouping.
- **Mobile App**: The name of the app that had the code issue.
- **OS Version**: The operating system version of the mobile device on which the code issue occurred.
- **Platform**: iOS or Android.
- **Region**: The originating region, such as the province or state.
- **Request Timestamp**: When the beacon arrived at the Collector. Snapshot only.
- **Severity**: The severity level set for the code issue.
- **Symbolication/Obfuscation**: Flag indicating whether the application code has been symbolicated/obfuscated.
- **User Data**: Any user data you might have added.
- **Version**: The version string of the application that had the code issue.
Code Issue Alerts

You can configure AppDynamics to send you alerts when your mobile application has code issues. You can configure email alerts for health rule violations based on the default health rule or by manually creating a policy with a notification action.

**How to Set Up Alerts for Code Issues**

Below is the basic workflow for setting up alerts for code issues:

1. **Report Errors**: In your application code, use the iOS, Android, or Xamarin SDK to report errors. Set the severity to either `INFO`, `WARNING`, or `CRITICAL`. Only errors with the severity of `CRITICAL` trigger alerts with the default health rule.

2. **Configure Thresholds**: Set the warning and critical thresholds for application not responding issues:
   b. From `Configure Application Not Responding Thresholds`, set the values for `Warning Threshold` and `Critical Threshold`. Any value that is less than the warning threshold will be labeled as `Info`.

3. **View alerts for code issues**.

**Report Errors with the Mobile SDKs**

See "Report Errors and Exceptions" ([iOS](#)/[Android](#)/[Xamarin](#)) to report errors and set the severity for iOS, Android, or Xamarin applications.

**Configure Threshold for Code Issue Alerts**

You can configure the thresholds for ANRs that must be reached before you are alerted. See `Configure Application Not Responding Thresholds` for instructions.

**View Code Issue Alerts**

The health rule `Critical Code Issues exist` is used to send alerts when one of the following occurs:

- You have used the Android or iOS SDK to report an error with the severity of `critical`.
- The time that your application has not responded exceeded the critical threshold.

After you set up the email alerts, when a health rule is violated, an email will be sent indicating whether there was a warning or critical alert and providing links to view code issue and session details.
Custom Data

On this page:
- Custom Data Types
- View Custom Data

Related pages:
- Customize the iOS Instrumentation
- Customize the Android Instrumentation

Mobile RUM provides substantial information on crashes and network and server-related performance. You may want, however, to customize the Mobile Agent to collect other metrics and data to understand how your application is performing for your users.

For information on customizing the agent, including sample code, see the following:
- Customize the iOS Instrumentation
- Customize the Android Instrumentation

To see the results of your data collection, you use the Custom Data view.

Custom Data Types

There are three kinds of custom data that you can collect:
- Info Points
- Custom Timers
- Custom Metrics

There are two other types of custom data: the breadcrumb and user data. Breadcrumbs are displayed only in Crash Snapshots. User data is available in Crash Analyze and Network Request Analyze. See the agent SDK docs for more information.

Info Points

An information point allows you to collect information on how a specific method in your code is performing. How many times was a particular method executed? Was an exception thrown? If you need to monitor multiple methods, you can create multiple info points.

If you know there are important methods you always want to monitor, create static info points as described in Customize the iOS Instrumentation and Customize the Android Instrumentation.

Custom Timers

Custom timers allow you to time how long something takes between any arbitrary start point and end point, even if they span multiple methods. For example, how long did it take to repaint a frame buffer? How long did the app take to start up?

Custom Metrics

Custom metrics allow you to collect data on any metric you define in your application. Metrics can be any integer-based data. Over the specified time frame, Mobile RUM calculates the minimum, maximum, average, sum, and count of these values.

View Custom Data

Your data is shown in tabular format in two tabs.

Custom Timers & Metrics

Click on any value (in blue) to display the same metric in the Metric Browser.
**Info Points**

The **Info Points** tab shows any current info points.
Troubleshoot Mobile Applications

You can use Mobile RUM to investigate two different kinds of problems that can arise with your mobile applications:

- Slow Network Requests
- Mobile Application Crashes
Troubleshoot Slow Network Requests from Mobile Applications

On this page:
- Identify Your Slowest Network Request Types
- Access Details of Individual Instances of Slow Requests

Related pages:
- Network Requests

The sections below show you how to identify slow network requests and investigate the cause of the problem.

**Identify Your Slowest Network Request Types**

1. Open the application in which you are interested.
2. In the left navigation bar, click **Network Requests**.
3. Select the **Network Requests** tab.
4. Click the top of the **Network Request Time (ms)** column, then toggle it to sort the network requests with the slowest ones at the top.
5. Skip over network requests that you expect to run for a long time or that have very little load (low Requests per Minute).
6. Select and double-click one of the slow network requests that you want to investigate.
7. In the network request dashboard, view the **Key Performance Indicators** at the top of the **Network Request Dashboard**. For example:
   - If the value for Network Request Time is large, the request or response body may be too large and is taking a while to transmit, or the data connection might be slow.
   - If the backend server is instrumented and the value for Total Server Time makes up a significant amount of the delay, scroll down to the **Related Business Transactions** section to investigate related business transactions on the server side.

**Access Details of Individual Instances of Slow Requests**

1. Select the **Snapshots** tab. The **Snapshots List** opens.
2. Click **Filters**.
3. In the **Network Request Names** dropdown under **Network in the Filters** panel, check the check box for the network request that you identified in Identify your slow network requests, then click **Search**. This restricts the list to snapshots for that network request only.
4. Click **Filters** again to close the filters panel.
5. In the list, click the top of the **Network Request Time (ms)** column, then toggle it to sort the network request snapshots with the slowest requests at the top.
6. Select and double-click one of the slow network requests.
   - The network request snapshot displays the details of the slow request.
   - If this request is associated with a server-side application that is also instrumented, scroll down to see if transaction snapshots for this request are available on the server side.
   - If transaction snapshots are available and if most of the time for this network request is spent on the server, click through to the related transaction snapshots to understand what is causing the slow performance on the server. See **Transaction Snapshots**.
Troubleshoot Mobile Application Crashes

On this page:
- Identify the Applications that Crash the Most or That Affect the Most Users

Related pages:
- Crash Dashboard
- Crash Snapshots

Use crash dashboards and crash snapshots to troubleshoot mobile application crashes.

Identify the Applications that Crash the Most or That Affect the Most Users

1. Open the application in which you are interested.
2. On the left navigation bar, click Crashes.
3. Select the Crash Dashboard tab.
4. Check the Unique Crashes list.
5. Sort either by Total Crashes or Impacted Users, depending on what you want to know. In the screenshot below, the list is sorted by impacted users.

6. View the Summary Crash Trend graph to discover particularly problematic time periods.

7. To find crashes in the problematic timeframe, click Analyze and drag up on the timeline where the most crashes occurred.

8. You should see the crash list reduced to those that occurred during the custom time period.
9. Double-click one of the unique crashes to open the Crash Details dialog.

10. From here, you can view the stack trace of the crash snapshot, note the thread and function in which the crash occurred. For some crashes, the crashed line number is also available. Optional: Click Download to get a text version of the stack trace to send to your application development team.

11. To access a complete data set for crashes that belong to this crash group, click Find Sessions. This opens the Mobile Sessions view with a filter for this Crash Group already selected.
Get More Information about Mobile RUM

Use the following to find out more about Mobile RUM.

- Mobile RUM Metrics
- Mobile RUM Supported Environments
- Mobile RUM Licenses
- Network Request Limits
Mobile RUM Metrics

On this page:
- Crash Metrics
- HTTP Errors
- Network Errors
- Network Requests
- Crashes via Analyze

AppDynamics displays key metrics for Mobile RUM on the various dashboards in the Mobile UI and in the Metric Browser.

With the exception of App Crashes per Minute, the crash information displayed in the Controller UI is based on data stored in the Events Store and not displayed in the Metric Browser.

Crash Metrics

- **App Crashes per Minute**: the arithmetic average number of crashes

HTTP Errors

An HTTP error occurs when an HTTP request is sent and a response is received, but the response status code indicates that an error occurred. These errors suggest that the network is working correctly, but there is a problem on the client side (4xx status codes) or the server side (5xx status codes) that prevented normal handling of the request.

- **HTTP Errors Per Minute**: the arithmetic average of errors per minute for errors that return an HTTP response code between 400 and 599
- **HTTP Errors (total)**: the total number of HTTP errors over the selected time range (shown in the Geo Dashboard)

Network Errors

A network error is any occurrence that prevents the HTTP request from being sent or the HTTP response from being received successfully. Typical causes of network errors include:

- Host cannot be resolved.
- Host refused connection.
- Connection timed out.
- Device is offline.
- General connectivity problems.

- **Network Errors per Minute**: the arithmetic average of network errors per minute
- **Network Errors (total)**: the total number of network errors over the selected time range (shown in the Geo Dashboard)

Network Requests

Network request metrics are reported for each platform and instrumented mobile application.

- **Network Request Time**: the arithmetic average interval in milliseconds between the time that a mobile application initiates a request by calling the system API and the time that the system returns the response to the application
- **Requests per Minute**: the arithmetic average number of HTTP requests per minute
- **Total Requests**: the total of HTTP requests per minute over the selected time range (shown in the Geo Dashboard)
- **Request Content Length**: the total number of bytes of data in the body of the request. The body is the part that comes after the blank line below the headers.
- **Response Content Length**: the total number of bytes of data in the body of the response.

Crashes via Analyze

With the exception of **App Crashes per Minute**, the crash information displayed in the Controller UI is based on data stored in the Events Store and not displayed in the Metric Browser.
Crash measurements are created using the Analyze event store. They are displayed in the **Crash Dashboard**.

- **Total Crashes**: the total number of crashes over the selected time range
- **Total Unique Crashes**: the total number of unique crash types over the selected time range
- **Impacted Users**: the total number of unique users affected by a crash
- **Crashes by Usage Stats**: the percentage of crashes based on device, OS version, carrier, and connection type
- **Crashes by Geo Location**: the total number of crashes by origin over the selected time range
Mobile RUM Supported Environments

See the following for the list the supported environments for Android, iOS, Xamarin, and Cordova:

- Android Agent
- iOS Agent
- Xamarin Agent
- Cordova Plugin
- React Native Agent

Mobile RUM Support

**Android Agent**

<table>
<thead>
<tr>
<th>Supported Environments</th>
<th>Name</th>
<th>Supported Version(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Android</td>
<td>2.3.3+</td>
</tr>
<tr>
<td>Architecture</td>
<td>arm64-v8a, armeabi, armeabi-v7a, mips, mips64, x86, and x86_64</td>
<td>-</td>
</tr>
<tr>
<td>Frameworks</td>
<td>Ant</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Gradle</td>
<td>0.6.3+</td>
</tr>
<tr>
<td></td>
<td>Maven</td>
<td>3.1.1+</td>
</tr>
<tr>
<td>Crash Reporters</td>
<td>Mobile RUM does not officially support 3rd-party crash reporters.</td>
<td>N/A</td>
</tr>
<tr>
<td>HTTP Libraries</td>
<td>HttpURLConnection/HttpsURLConnection</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>HttpClient</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>OkHttp</td>
<td>2.2.0–3.9.0</td>
</tr>
<tr>
<td></td>
<td>ch.boyeretrofitandroidlib</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>retrofit</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>retrofit2</td>
<td>-</td>
</tr>
</tbody>
</table>

Other HTTP libraries can be added by using the agent SDK. See [Customize the Android Instrumentation](#) for more information.

<table>
<thead>
<tr>
<th>Android Gradle Plugin Version</th>
<th>Minimum Android Agent Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6.3</td>
<td>3.8.2.0</td>
</tr>
<tr>
<td>0.7.3–0.9.2</td>
<td>3.8.3.0</td>
</tr>
<tr>
<td>0.10.0–0.12.0</td>
<td>3.9.0.0</td>
</tr>
<tr>
<td>1.1.2–1.1.3</td>
<td>4.0.3.3</td>
</tr>
<tr>
<td>1.3.0–1.4.0</td>
<td>4.0.7</td>
</tr>
<tr>
<td>1.5.0–2.2.0</td>
<td>4.2.0`</td>
</tr>
<tr>
<td>&gt;= 2.3.0 (forces Build Tools 25.0.0 or higher)</td>
<td>4.2.9</td>
</tr>
</tbody>
</table>
*The Android Agent Plugin versions before 4.2.9 are not compatible with Android Build Tools versions 24+ because of the Java 8 bytecode generated in class files.*

### iOS Agent

<table>
<thead>
<tr>
<th>Supported Environments</th>
<th>Name</th>
<th>Supported Version(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>iOS</td>
<td>9+</td>
</tr>
<tr>
<td>Architecture</td>
<td>Apple 32-bit ARM, Apple 64-bit A7</td>
<td>-</td>
</tr>
<tr>
<td>Framework</td>
<td>XCode</td>
<td>8+</td>
</tr>
<tr>
<td>Crash Reporters</td>
<td>Mobile RUM does not officially support 3rd-party crash reporters.</td>
<td>N/A</td>
</tr>
<tr>
<td>Apple WatchKit Extension Environments</td>
<td>watchOS</td>
<td>1</td>
</tr>
<tr>
<td>HTTP Libraries</td>
<td>NSURLConnection</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>NSURLSession</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Alamofire - Initialize the iOS Agent before initializing Alamofire to ensure that network requests are monitored.</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Other HTTP libraries can be added by using the agent SDK. See Customize the iOS Instrumentation for more information.</td>
<td>-</td>
</tr>
</tbody>
</table>

### Xamarin Agent

<table>
<thead>
<tr>
<th>Supported Environments</th>
<th>Name</th>
<th>Supported Version(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Android</td>
<td>2.3.3+</td>
</tr>
<tr>
<td></td>
<td>iOS</td>
<td>9+</td>
</tr>
<tr>
<td>Architecture</td>
<td>Xamarin.Android Supported CPU Architectures</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Apple 32-bit ARM, Apple 64-bit A7 - See 32/64-bit platform considerations for information regarding the correct architecture for your iOS version.</td>
<td>-</td>
</tr>
<tr>
<td>Framework</td>
<td>.NET - Mono</td>
<td>4.8+</td>
</tr>
<tr>
<td>Crash Reporters</td>
<td>Mobile RUM does not officially support 3rd-party crash reporters.</td>
<td>N/A</td>
</tr>
<tr>
<td>Apple WatchKit Extension Environments</td>
<td>Not supported</td>
<td>-</td>
</tr>
<tr>
<td>HTTP Libraries Cordova Plugin</td>
<td>N/A - The Xamarin Agent does not support automatic instrumentation for network requests made with any library. You will need to manually instrument HTTP network requests regardless of what library is used.</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Supported Environments</th>
<th>Name</th>
<th>Supported Version(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Android</td>
<td>4.4 - 8.1</td>
</tr>
<tr>
<td></td>
<td>iOS</td>
<td>9+</td>
</tr>
<tr>
<td>Architecture</td>
<td>Android: arm64-v8a, armeabi, armeabi-v7a, mips, mips64, x86, and x86_64</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>iOS: Apple 32-bit ARM, Apple 64-bit A7</td>
<td>-</td>
</tr>
<tr>
<td>Framework</td>
<td>Apache Cordova</td>
<td>7+</td>
</tr>
<tr>
<td>Crash Reporters</td>
<td>Mobile RUM does not officially support 3rd-party crash reporters.</td>
<td>N/A</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>Apple WatchKit Extension Environments</td>
<td>Not supported</td>
<td>-</td>
</tr>
<tr>
<td>HTTP Libraries</td>
<td>Android:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• HttpURLConnection/HttpsURLConnection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• HttpClient</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• OkHttp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• ch.boye.httpclientandroidlib</td>
<td></td>
</tr>
<tr>
<td></td>
<td>iOS:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• NSURLConnection</td>
<td></td>
</tr>
</tbody>
</table>

**React Native Agent**

**Supported Environments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Supported Version(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Android</td>
</tr>
<tr>
<td></td>
<td>4.4 - 8.1</td>
</tr>
<tr>
<td></td>
<td>iOS</td>
</tr>
<tr>
<td></td>
<td>9+</td>
</tr>
<tr>
<td>Architecture</td>
<td>Android: arm64-v8a, armeabi, armeabi-v7a, mips, mips64, x86, and x86_64</td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>iOS: Apple 32-bit ARM, Apple 64-bit A7</td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Framework</td>
<td>React Native</td>
</tr>
<tr>
<td></td>
<td>7+</td>
</tr>
<tr>
<td>Crash Reporters</td>
<td>Mobile RUM does not officially support 3rd-party crash reporters.</td>
</tr>
<tr>
<td>Apple WatchKit Extension Environments</td>
<td>Not supported</td>
</tr>
<tr>
<td>HTTP Libraries</td>
<td>Android:</td>
</tr>
<tr>
<td></td>
<td>• HttpURLConnection/HttpsURLConnection</td>
</tr>
<tr>
<td></td>
<td>• HttpClient</td>
</tr>
<tr>
<td></td>
<td>• OkHttp</td>
</tr>
<tr>
<td></td>
<td>• ch.boye.httpclientandroidlib</td>
</tr>
<tr>
<td></td>
<td>iOS:</td>
</tr>
<tr>
<td></td>
<td>• NSURLConnection</td>
</tr>
<tr>
<td></td>
<td>• NSURLSession</td>
</tr>
</tbody>
</table>
Mobile RUM Licenses

On this page:
- License Key
- License Editions
- Usage Period
- Total Mobile RUM Agents Licensed
- Monthly Active Agents (MAA) Allocated
- Monthly Active Agents Consumption
- Crash Monitoring Support
- Overages

This topic describes how to interpret the details of your Mobile RUM license information.

License Key

This is the unique identifier that AppDynamics uses to associate end-user data, Browser RUM, Browser Synthetic, and Mobile RUM, to your account. From a practical perspective, you only need to know this information for troubleshooting purposes. The same key applies to all three services. However, each product has its own types and metrics for allowed usage.

License Editions

The two supported license editions for Mobile RUM are described in the table below.

<table>
<thead>
<tr>
<th>License Edition</th>
<th>Description</th>
</tr>
</thead>
</table>
| Pro             | Mobile Pro (Paid) license covers the number of Mobile Pro license units you have purchased.  
|                 | • For details about license units, see the following sections on the License Entitlements and Restrictions:  
|                 |   • Mobile Real User Monitoring (SaaS)  
|                 |   • Mobile Real User Monitoring (on-prem) |
| Lite            | Mobile Lite (Free) provides one Mobile Lite license unit. If you have not purchased a Mobile Pro account, you will receive a Mobile Lite account. Mobile Lite gives you:  
|                 | • Crash reporting  
|                 | • Percentile-based and static health rules and alerts  
|                 | • API Access |

Usage Period

The usage period for mobile is always the current month, even if you have a multi-year license. The usage period begins and the meter resets on the first of each month at 12:00 am Pacific time.

Total Mobile RUM Agents Licensed

For a Mobile Pro license, this is the equivalent of the total number of license units licensed by your account. For a Mobile Lite license, this is one Lite license unit. One Lite license unit includes access to the Crash Dashboard and Crash Snapshots (covering the previous 365 days) for 5 million devices.

Monthly Active Agents (MAA) Allocated
The monthly active agent (MAA) is a unique installation of an application instrumented with an AppDynamics Mobile Real User Monitoring Agent and launched on a device in a given calendar month.

This is the number of active agents per month allocated by your license.

Each Mobile Pro license unit provides a number of active agents per month per native mobile app instrumented with AppDynamics Mobile SDKs. To get your total number of active agents, AppDynamics multiplies the number of licenses you have by the number of active agents per license.

You cannot carry over unused active agents from month to month.

**Monthly Active Agents Consumption**

The Monthly Active Agents is the number of active agents during the current month. The month begins on the 1st at 12:00 am Pacific time, at which time this value is reset to zero.

If this value is greater than Monthly Active Agents Allocated and your license allows overages, you are incurring overage charges.

When your consumed monthly usage is at 90% of your month allocation you will see a warning in the Mobile RUM dashboard.

**Crash Monitoring Support**

The Mobile RUM Pro and Lite Edition licenses provide free crash reporting for 2 million agent installations. The Mobile RUM Pro edition license also provides crash reporting for allocated monthly agents.

**Overages**

How overages are handled is determined by the terms of your license agreement.

If your license does not allow overages, AppDynamics continues reporting mobile metrics generated by active agents who started using your application before you consumed all your allocated monthly active agents. However, it does not report metrics for additional active agents. For example, if your license allows 50,000 monthly active agents, the agent does not report metrics generated by the 50,001st active agent and beyond but will continue to report on the first 50,000 for the rest of the month. Keep in mind that if there are patterns in which certain types of users activate the application in the first part of the month, you may be missing what other types of users are experiencing. Limiting the number of active agents licensed is not a recommended way to do sampling.

If your license does allow overages and your usage exceeds the limit, AppDynamics continues reporting mobile metrics for additional users and bills you for the overage at the unit rate stipulated by your license agreement.

If you need to stop incurring overage charges, you can disable mobile monitoring by clearing the Enable Mobile checkbox in the Mobile RUM configuration window. This will stop mobile monitoring and stop overage charges after a delay of approximately one minute. See "Disable Mobile RUM on Your Controller" in Set Up and Configure Mobile RUM.
Network Request Limits

On this page:

- Remove Network Requests Without Load
- Exclude Requests That Do Not Need to Be Monitored
- Group Network Requests of Similar Type

Related pages:

- Configure Mobile Network Request Naming

The EUM Server (on-prem/SaaS) can process a maximum of 2000 network request types per mobile app group and 500 network requests per mobile application.

If the number of network requests for an application exceeds one of these limits, AppDynamics continues to monitor network requests that have already been discovered, but new network requests will not be discovered. Should a limit be exceeded, a warning appears on the network request list.

You can use the following techniques to keep your usage under the two network request limits.

Remove Network Requests Without Load

Removing network request types that have been discovered but are not receiving any load is a good place to start to reduce the number of your registered requests.

Deleting network requests does not prevent them from being re-discovered in the future if the request once again comes under load. To prevent specific network requests from ever being discovered, you must create exclude rules. See the section "Creating Mobile Exclude Rules" in Configure Mobile Network Request Naming.

Delete network requests with no load

1. In the Network Requests list, uncheck the With Load checkbox.
2. Refresh the list.
3. Find the requests that have no load. They are candidates for removal.
4. Select the requests that you want to delete.
5. In the More Actions dropdown, click Delete Request(s).

Exclude Requests That Do Not Need to Be Monitored

The Mobile Agent may be detecting network requests types that are not interesting for you to monitor. You have two methods to exclude these types of network requests.

The first method is to add custom exclude rules from Configuration > Mobile Request Naming, Thresholds, & Percentiles. See Creating Mobile Exclude in Rules for explicit instructions.

Using custom exclude rules, network requests for new data that match the exclude rules will not be registered. The load on the existing network requests that match the exclude rule will drop to zero. To delete any network requests of the excluded type that have already been registered from the network request list, follow the procedure described in Delete network requests with no load.

The second method is to select an existing network request from the Network Requests list and then exclude it. This will change the app's exclude rules so that incoming data matching that request will not be tracked. In addition, the historical data for that request will be deleted.

To use the second method for the exclude rules that prevent a network request from being tracked and remove historical data for that network request:

1. From Network Requests, select the network request you want to exclude.
2. Click Actions > Exclude Requests to open the Exclude Requests dialog.
3. Click Exclude.
Group Network Requests of Similar Type

Review the default network request naming rule described in Configure Mobile Network Request Naming. It is possible that the default rule is generating many more network requests types than are desirable.

For example, perhaps your application loads images dynamically and stores them on your server with URLs like http://myapp.com/image/image1234.jpg. This would cause a separate network request to be generated for each image, which is probably not what you want. You could create a custom naming rule to group all the image URLs as a single network request. See Configure Mobile Network Request Naming for instructions.

After you have created custom rules to reduce the number of network requests detected, unregister the network requests for those are now covered by the custom rule, following the procedure described in Delete network requests with no load.
Mobile Agent Version and Deployment Support Matrix

On this page:
- Post Release Support for the Mobile Agents
- Deployment Support for Mobile Agent Features

Related pages:
- Mobile RUM Supported Environments

Using the AppDynamics Mobile Agents with the following recommended minimum Controller/EUM Server versions will ensure that all features for the specified releases will function correctly and that there will be no unexpected side effects.

<table>
<thead>
<tr>
<th>Mobile Agent</th>
<th>Agent Version(s)</th>
<th>Minimum Controller / EUM Server Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>React Native Agent</td>
<td>1.0.x</td>
<td>4.5.8</td>
</tr>
<tr>
<td>iOS Agent</td>
<td>50.0.x</td>
<td>4.5.0</td>
</tr>
<tr>
<td></td>
<td>50.1.x</td>
<td>4.5.8</td>
</tr>
<tr>
<td>Android Agent</td>
<td>5.0.x</td>
<td>4.5.0</td>
</tr>
<tr>
<td></td>
<td>5.1.x</td>
<td>4.5.8</td>
</tr>
<tr>
<td>Xamarin Agent</td>
<td>50.0.x+</td>
<td>4.5.0</td>
</tr>
<tr>
<td></td>
<td>50.1.x</td>
<td>4.5.8</td>
</tr>
<tr>
<td>Cordova Plugin</td>
<td>1.1.x – 1.7.x</td>
<td>4.5.0</td>
</tr>
<tr>
<td></td>
<td>1.8.x</td>
<td>4.5.8</td>
</tr>
</tbody>
</table>

Post Release Support for the Mobile Agents

Although Mobile Agents may be compatible with future releases of the Controller and EUM Server, AppDynamics only provides support for Mobile Agent versions with the Controller and EUM Server for two years after the release of the agent.

Deployment Support for Mobile Agent Features

For deployment support for individual mobile features, see the Mobile Agent SDK documentation:

- iOS SDK Documentation
- Android SDK Documentation
- Xamarin SDK Documentation
- Cordova SDK Documentation
- React Native Agent Documentation
# Mobile Agent Feature Support

The table below lists the feature support for the Mobile Agents. For instrumentation instructions, see the custom instrumentation pages (Android, iOS, Xamarin, Cordova).

<table>
<thead>
<tr>
<th>Feature</th>
<th>Mobile Agents</th>
<th>Supported Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Errors</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Crash Reporting</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>App Not Responding (ANR)</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Error Reporting</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Crash Reporting Callback</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td><strong>Network</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Automatic Network Request Capture</td>
<td>✔️</td>
<td>URLConnection, HttpClient, OkHttp, OkHttp3, HttpClientAndroidLib</td>
</tr>
<tr>
<td>Manual Network Request Reporting</td>
<td>✔️</td>
<td>NSURLConnection, NSSession</td>
</tr>
<tr>
<td>Server Correlation</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Connection Transition Reporting</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Network Request Callback</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td><strong>User Interaction</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Activity Tracking</td>
<td>✔️</td>
<td>(Use the SessionFrame API)</td>
</tr>
<tr>
<td>Root View Tracking</td>
<td>✗</td>
<td>(Use the SessionFrame API)</td>
</tr>
</tbody>
</table>

Copyright © AppDynamics 2012-2019
<table>
<thead>
<tr>
<th>Feature</th>
<th>Captured User Interactions</th>
<th>Captured User Interactions</th>
<th>Hybrid Support (JavaScript Agent Support)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UI Tracking</strong></td>
<td>![Checkmark]</td>
<td>![Cross]</td>
<td>![Cross]</td>
</tr>
<tr>
<td>Fragment Tracking</td>
<td>![Checkmark]</td>
<td>![Cross]</td>
<td>(Use the SessionFrame API)</td>
</tr>
<tr>
<td>View Controller Tracking</td>
<td>![Cross]</td>
<td>![Checkmark]</td>
<td>(Use the SessionFrame API)</td>
</tr>
<tr>
<td>Screenshots</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
<td>![Cross]</td>
</tr>
<tr>
<td>Screenshots Touch Points</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
<td>![Cross]</td>
</tr>
<tr>
<td><strong>Custom Data</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Static Info Points</td>
<td>![Checkmark]</td>
<td>![Cross]</td>
<td>![Cross]</td>
</tr>
<tr>
<td>Manual Info Points</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
</tr>
<tr>
<td>Breadcrumbs</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
</tr>
<tr>
<td>User Data</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
</tr>
<tr>
<td>Custom Metrics</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
</tr>
<tr>
<td>Session Frame API</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
</tr>
<tr>
<td>Custom Timers</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
</tr>
<tr>
<td>Programmatic Session Control API</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
</tr>
<tr>
<td><strong>Configuration</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom Collector Channel</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
<td>![Cross]</td>
</tr>
<tr>
<td>Set App Key After Initialization</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
</tr>
<tr>
<td><strong>Captured User Interactions</strong></td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
<td>![Cross]</td>
</tr>
<tr>
<td>EditText</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
<td>![Cross]</td>
</tr>
<tr>
<td>Button</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
<td>![Cross]</td>
</tr>
<tr>
<td>List Selection</td>
<td>![Checkmark]</td>
<td>![Checkmark]</td>
<td>![Cross]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>Base Page Entry and</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Virtual Page as Mobile</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network Requests</td>
<td></td>
<td>❌</td>
<td></td>
</tr>
<tr>
<td>Ajax Calls as</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Mobile Network Requests</td>
<td></td>
<td>❌</td>
<td></td>
</tr>
</tbody>
</table>
IoT Monitoring

AppDynamics IoT Monitoring enables you to track and understand the transactions of your IoT applications. Because IoT devices are diverse, both in terms of the platforms they use and their business function, AppDynamics has developed a REST API in addition to language SDKs to provide the maximum flexibility for reporting IoT data. This API can be used from any device that supports HTTPS and is connected to the Internet.

IoT Monitoring requires application developers to instrument their code. To make this process easier, AppDynamics has developed C/C++ and Java SDKs, so that developers using the platforms supporting these languages can leverage the features of the SDK instead of using the REST API. The IoT SDKs use the REST APIs to report IoT Data to the EUM Server, where the data is aggregated and made available to the AppDynamics Controller and the Events Service as shown in the diagram below.

Benefits of IoT Monitoring

IoT Monitoring enables you to capture and diagnose network requests, errors, and custom application and domain-specific events for applications running on devices that impact business objectives.

With IoT Monitoring, you can monitor the following for devices:

- **Availability**
- **Network Performance:**
  - slow interaction
IoT Monitoring Use Cases

IoT applications are typically embedded applications running on connected devices. This section focuses on how IoT Monitoring can collect and report data for the most common categories of IoT applications.

In the tabs below, you can view these IoT app categories and see how a typical example of each would use the three supported IoT Monitoring events to report data.

### Retail

This category of IoT apps process orders, payments, and manage product inventory. Examples include point of sale (PoS) devices, smart shelves, and gateways (local servers that aggregate information within certain geography/region).

<table>
<thead>
<tr>
<th>Device Example</th>
<th>Device Information</th>
<th>Example Network Requests</th>
<th>Custom Events Examples</th>
<th>Handled Errors</th>
<th>Exceptions/Crashes</th>
</tr>
</thead>
</table>
| **Point of Sale** | • device ID  
• app version  
• hardware versions  
• firmware versions | • checkout  
• payment transaction | • total sales  
• number of items sold  
• discount applied  
• total checkout time  
• total processing time | • credit card declined  
• unable to read credit card  
• incorrect pin code | • crashes  
• reboots  
• frozen UI  
• timeouts  
• null point exceptions |

### Media

IoT apps in this category enable users to seamlessly experience media content across different devices. An IoT app on a set-top box (STB) can quickly change the channel/content based on what the user was watching or listening to on their computer or phone.

<table>
<thead>
<tr>
<th>Device Example</th>
<th>Device Information</th>
<th>Example Network Requests</th>
<th>Custom Events Examples</th>
<th>Handled Errors</th>
<th>Exceptions/Crashes</th>
</tr>
</thead>
</table>

---

- latency between the device and backend services
- **Usage**: User and system behavior and patterns
- **Errors and Exceptions**
### Connected Cars

Because automotive head units are often computers, IoT apps are used to manage user subscriptions and monitor the user experience.

<table>
<thead>
<tr>
<th>Device Example</th>
<th>Device Information</th>
<th>Example Network Requests</th>
<th>Custom Events Examples</th>
<th>Handled Errors</th>
<th>Exceptions/Crashes</th>
</tr>
</thead>
</table>
| Infotainment System | • device ID  
• app version  
• hardware versions  
• firmware versions | • media play  
• pause  
• stop requests  
• ad performance | • media downloads  
• media skips  
• subscription details  
• total play time | • play errors  
• song not available  
• download errors | • crashes  
• reboots  
• frozen UI  
• timeouts  
• null point exceptions |

### Smart Homes

These IoT apps improve the security as well as reduce the energy and maintenance costs of intelligent buildings and smart homes. Examples include home security devices and smart thermostats, smart keyless locks, and smart lights.

<table>
<thead>
<tr>
<th>Device Example</th>
<th>Device Information</th>
<th>Example Network Requests</th>
<th>Custom Events Examples</th>
<th>Handled Errors</th>
<th>Exceptions/Crashes</th>
</tr>
</thead>
</table>
| Smart Homes | • device ID  
• app version  
• hardware versions  
• firmware versions | • media play  
• pause  
• stop requests  
• ad performance | • media downloads  
• media skips  
• subscription details  
• total play time | • application loading errors  
• connectivity errors  
• audio errors  
• launching errors | • crashes  
• reboots  
• frozen UI  
• timeouts  
• null point exceptions |
### Home Security
- device ID
- app version
- hardware versions
- firmware versions
- home sensor status
- communication between gateway devices and sensors
- health alerts
- sensor health
- number of alerts
- connectivity status
- total roundtrip time to trigger an alarm
- total time to set security status
- device wake-up/sleep
- alarm notification
- connectivity issues
- crashes
- reboots
- frozen UI
- timeouts
- null point exceptions

### Travel
This category of IoT apps monitors and improves the travel experience. You can find IoT apps used in airport kiosks, luggage tracking devices, and embedded in cabin seats to monitor passenger comfort.

<table>
<thead>
<tr>
<th>Device Example</th>
<th>Device Information</th>
<th>Example Network Requests</th>
<th>Custom Events Examples</th>
<th>Handled Errors</th>
<th>Exceptions/Crashes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airport Kiosk</td>
<td>device ID</td>
<td>fetching passenger records</td>
<td>number of users completing a transaction</td>
<td>passenger records not found</td>
<td>crashes</td>
</tr>
<tr>
<td></td>
<td>app version</td>
<td>payment processing</td>
<td>number of issued boarding passes</td>
<td>printer error</td>
<td>reboots</td>
</tr>
<tr>
<td></td>
<td>hardware versions</td>
<td></td>
<td>total user activity time</td>
<td>connectivity issues</td>
<td>frozen UI</td>
</tr>
<tr>
<td></td>
<td>firmware versions</td>
<td></td>
<td>total time spent on each screen</td>
<td></td>
<td>timeouts</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>null point exceptions</td>
</tr>
</tbody>
</table>
IoT SDKs

IoT Monitoring provides SDKs for C/C++ and Java languages. Some of the features of IoT SDKs are:

- APIs for reporting instrumentation data through custom, network, and error events
- Serialization of all events into JSON format before sending them to EUM Server
- API to plugin application HTTPS stack into SDK which will be used to communicate with EUM Server
- Business Transaction (BT) correlation
- Auto-disable when monitoring is turned off from the Controller UI

For languages other than C++ and Java, you can use the REST API to send custom, network request, and error events.

IoT Monitoring UI

IoT Monitoring also provides the following features through the Controller UI:

- Dashboards for monitoring devices, network requests, as well as errors and exceptions
- Predefined and custom widgets to visualize reported application data
- Analytics for device application data through AppDynamics Application Analytics
- Configuration for naming and excluding network requests

IoT Monitoring Workflow

The IoT SDKs and REST API enable you to report application data for a large number of different kinds of devices. This flexibility allows for a great deal of customization in the kind of the data you report, the quantity of data you report, and the timing of when the data is sent to the EUM Server.

To monitor customized data, you will need to better understand the data generated by your application and your device instrumentation. The goal of this page is to guide you through the process of understanding your application data, instrumenting your device application, and creating custom dashboards, so you can monitor device performance and activity.

It is recommended that you follow the steps below as a guideline to monitor your devices:

1. Define your monitoring goals.
2. Identify what data needs to be captured.
3. Determine which events to use to report the data.
4. Obtain an EUM App Key.
5. Report events.
6. Monitor the availability, usage, and performance of your devices.
7. Build widgets.
8. Diagnose problems through filters.
9. Improve monitoring and troubleshoot.

Define Monitoring Goals

AppDynamics IoT Monitoring enables you to monitor the availability, performance, and usage of your device. You should devise a plan that prioritizes your monitoring goals, considering the needs of all stakeholders (developers, DevOps, business units), and lists the resources that need to be monitored. Also, you should consider who will be instrumenting the device application, who will be performing the monitoring, and the parties who should be notified when something goes wrong.

Identify Data to Capture

Next, you'll need to analyze your device to identify what application data can be used to meet your monitoring goals. For example, if your device is a car, you would want to look at network requests to check device availability or report custom data for the wear and tear on its components or road conditions.

Determine Which Event Types to Use to Report Data
You report app data through the three types of events shown in the table below. Based on this information, map the most suitable event types to the data you identified and your monitoring goals.

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Monitoring Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom</td>
<td>Usage</td>
</tr>
<tr>
<td></td>
<td>Business</td>
</tr>
<tr>
<td>Network Request</td>
<td>Availability</td>
</tr>
<tr>
<td></td>
<td>Performance</td>
</tr>
<tr>
<td></td>
<td>Usage</td>
</tr>
<tr>
<td>Error Events</td>
<td>Performance</td>
</tr>
</tbody>
</table>

**Obtain an EUM App Key**

Follow the instructions provided in Get an EUM App Key.

**Report Events**

You can use the IoT SDKs or REST API to capture and report key data points using one of the event types. If you're using a language other than C/C++ or Java, use the IoT REST API, which gives you the flexibility to report data from any platform.

Follow the getting started instructions to learn how to install the SDK and instrument your IoT application:

- Instrument Applications with the IoT C/C++ SDK
- Instrument Applications with the IoT Java SDK

**Monitor Your IoT Application**

To monitor your IoT applications, you will need to understand the function and purpose of the three IoT dashboards. See Monitor Applications with the IoT Dashboards for a brief introduction of each dashboard and usage instructions.

**Build Widgets Based on Your Data Models**

Each dashboard has a set of predefined widgets for basic performance and activity monitoring. You will need to create your own widgets, however, to optimize the monitoring capabilities of IoT Monitoring. Based on the events you reported, create custom widgets to monitor device activity and performance. See the section Build Custom Dashboard Widgets to learn how to build widgets with the IoT Widget Builder Wizard.

**Diagnose Problems**

When you do discover deviations from expected behavior, you can then add criteria to filter results. This enables you to narrow the results to certain devices or possible causes.

**Troubleshoot Issues and Improve Monitoring**

For performance issues, you can open the device details to analyze errors and slow network requests. The device details allow you to trace the error event you are interested in and even download the stack trace. The developer can then use this information to determine the root cause of the problem and devise a possible solution. Going back to our point-to-sale device example, you might discover that most of the failed payments were due to network request errors.

You can use the usage information to improve your monitoring or the devices themselves. For example, an inventory management application indicates that several stores sell a smaller quantity of a certain product. It turns out that this product is often out of stock at these stores. You could improve the monitoring of your IoT app by using custom events to report when a product is sold out and then creating custom widgets to let you know which stores need to restock.
IoT Analytics

All IoT data is processed and stored by the AppDynamics Platform Events Service. A separate product component, AppDynamics Business iQ, has an event type called Connected Device Data. This event type is based on the same Events Service and uses the same data. Viewing this event type from the Analytics UI offers additional capabilities, including:

- Additional predefined widgets, such as the funnel widget
- ADQL for searching the data
- Creating custom widgets
- Manipulating multiple dashboard types
- Longer retention time for data storage

IoT Analytics does not require a license separate from the IoT Monitoring license.

To view Analytics data for connected devices, users need Connected Devices Permissions for the application (identified by a specific App Key).

IoT Licenses and Limits

Currently, AppDynamics only offers the IoT PRO license. To learn about the license entitlements and restrictions, contact your AppDynamics account representative.
Set Up and Access IoT Monitoring

On this page:

- SDKs vs REST API
- Get an EUM App Key
- Report IoT Data

This page will help you decide whether to use an IoT SDK or the IoT REST API, get an EUM App Key, and direct you to instructions for capturing and reporting data.

SDKs vs REST API

The IoT SDKs use the REST API to transmit data to the EUM Server. By handling HTTP requests, serializing JSON, and managing events in memory, the SDKs make it easier to capture and report data, so you can focus on instrumenting your application. For platforms other than C++ or Java, you can take advantage of the wide variety of HTTP(S) and JSON libraries to use the IoT REST API to report data.

Get an EUM App Key

To get an EUM App Key, you need to either create an IoT app manually or use the Getting Started Wizard, which is recommended if you are a new user. You will need the EUM App Key to use the IoT REST API or IoT SDKs to report device information and events to the EUM Server. Multiple IoT applications can share the same key.

Create an IoT Application with the Getting Started Wizard

From your Controller:

1. Click User Experience from the top navigation bar.
2. Select the Connected Devices tab.
3. Click Get Started.
4. From the Create Application dialog, select Create an Application using the Getting Started Wizard.
5. Click OK.
6. For step one of the Getting Started Wizard - Connected Devices page, enter a name for your IoT app.
7. Click Continue.
8. From step two, select which method you want to use to report IoT data (Java SDK, C/C++ SDK, REST API).
9. Copy the values for the App Key and the Collector URL. You'll be using both to report IoT data.

Manually Create an IoT Application

From your Controller:

1. Click User Experience from the top navigation bar.
2. Select the Connected Devices tab.
3. Click Get Started.
4. From the Create Application dialog, select Create an Application manually.
5. Enter a name for your IoT app.
6. Click OK.

7. From the Connected Devices tab, select IoT app and click to copy the App Key to your clipboard.

Report IoT Data

Based on the method you plan on reporting data, use your App Key and follow the instructions for one of the following tutorials:

- Instrument Applications with the IoT C/C++ SDK
- Instrument Applications with the IoT Java SDK
- Instrument Applications with the IoT REST APIs
Correlate Business Transactions for IoT Monitoring

You can correlate IoT network request events with business transactions. The correlation is made between beacons containing network request event information and instances of business transactions (transaction snapshots).

The correlation enables you to map network request events made by your IoT application with an underlying backend business application. Business transaction correlation is disabled by default.

By correlating business transactions with network request snapshots, you can identify potential issues with the backend business application that are causing bad user experiences. For example, you might find that a server error or a database query is causing a slow or very slow user experience.

Requirements

To correlate business transactions, you are required to do the following:

- purchase licenses for Application Performance Monitoring (APM) and End-User Monitoring (EUM)
- instrument a business application with an app agent
- instrument an IoT application with one of the IoT SDKs (C/C++ or Java) or the IoT REST API
- enable business transaction correlation in the Controller UI

App Server Agents Supporting Business Transaction Correlation

To correlate business transactions, your business application must have one of the following app server agents installed:

- Java Agent
- .NET Agent
- Node.js Agent
- PHP Agent
- Python Agent

How It Works

The steps below outline how IoT applications correlate network requests with business transactions:

1. The IoT application attaches AppDynamics HTTP headers to a network request to the app server agent. These AppDynamics HTTP headers instruct the app server agent to send back correlation headers.
2. The app server agent does the following:
   - sends HTTP response headers identifying the business transaction to the IoT application.
   - aggregates backend metrics and sends them along with the business transaction identifiers to the Controller. This serves as the content for the transaction snapshot.
3. The IoT Agent sends business transaction identifiers (from the HTTP response headers) as part of IoT network request event to the EUM Server. This serves as the content for the network request.
4. The Controller fetches the events and business transaction identifiers from the EUM Server and then uses the business transaction identifiers to correlate the network request with the transaction snapshots.

Enable Business Transaction Correlation
You need to configure the Controller to correlate business transactions. The Controller will map the business transactions with the network request snapshots based on the process described above in How It Works.

To enable business transaction correlation:

1. From the Application Dashboard, click Configuration.
2. Click User Experience App Integration >.
3. From the Business Transaction Correlation tab, check the Enable Business Transaction Correlation check box.

Specify Business Transactions to Include Correlation Headers (Optional)

You can also specify which business transactions will include or exclude correlation headers. If you do not add request match rules or request exclude rules, all requests will have correlation headers.

To add a request rule:

1. Click the plus icon to open the Create HTTP Request Match Rule dialog.
2. From the Create HTTP Request Match Rule dialog:
   a. Check the Method checkbox and select an HTTP method that you want to match. If you do not select an HTTP method, the rule will be applied to all HTTP requests.
   b. Check the URI checkbox and enter your criteria.
   c. Click Save to save the match rule.
3. From the Business Transaction Correlation tab, click Save.
1. Correlate IoT Network Request Events with Business Transactions

After you have enabled business transaction correlation, your IoT application can then request business transaction headers from the business application. You can use the IoT SDKs or the IoT REST API to send the returned response headers from the business application with beacons to report IoT network request events and correlate those events with the business transaction.

To learn how to correlate business transactions for IoT applications, see "Correlate Business Transactions with Network Requests" for either the IoT C/C++ SDK, IoT Java SDK, or the IoT REST API.

2. View Business Transaction Correlation

There are several ways to navigate from a network request snapshot to its correlated business transaction. The following steps show you one possible way.

   1. From the IoT Device Dashboard, double-click one of the device records in the Unique Devices widget:
2. You should now see a list of network requests in the Activity Stream tab. You can click a link in the activity stream to view the transaction flow map for a business transaction.

3. Click the Business Transactions tab to view a list of correlated business transactions.
4. You can then click a business transaction to view the **Transaction Flow Map** shown below, which provides a visual representation of the components and activities of the business application during the correlated business transaction.
Instrument Applications with the IoT C/C++ SDK

On this page:
- Run the Sample C++ Application
- Upgrade the IoT C++ SDK

The IoT C++ SDK provides APIs to instrument C++ applications running on connected devices such as industrial or home gateways, points of sale, smart TVs or car infotainment systems. This getting started will show you how to install the C++ SDK and instrument your IoT application.

Follow the steps below to get your EUM App Key and instrument your IoT C/C++ apps.

1. Learn About the C/C++ SDK
2. Check the Requirements
3. Get the IoT C++ SDK
4. Install the C++ SDK in Your Application
5. Add the SDK Headers
6. Initialize the SDK
7. Register Network Interface
8. Add and Send Events
9. Correlate Business Transactions with Network Requests (Optional)
10. Verify the Instrumentation in the Controller UI
11. Customize the IoT C++ Instrumentation (Optional)

Learn About the C/C++ SDK

You should know that the C++ SDK:

- Operates within the application thread and doesn’t spawn any new threads.
- Keeps all the event data in memory and doesn’t persist on disk.
- Provides an API to register for the network interface.
- Uses application’s HTTPS stack to communicate with the EUM Server.
- Provides an API to fetch SDK log messages. Application developers have to manage logs by writing the log messages to `std::cerr` or to a log file.
- Uses the open-source `json-c` library that is statically linked.
- Makes synchronous blocking API calls that are not thread-safe. Application developers are responsible for making thread-safe calls.

Check the Requirements

Before you get started, make sure you meet the following requirements:

- GNU C++ Compiler (g++) version 4.2 for 32/64-bit architectures
- Any Linux distribution based on glibc 2.20+
- HTTPS stack for sending beacons to the EUM Cloud
- EUM App Key

Get the IoT C++ SDK

You can get the C++ SDK by cloning or downloading the IoT C++ SDK from GitHub. Follow the instructions given in Installation to build the IoT C++ SDK.
Upgrade the IoT C++ SDK

From the root directory of your clone of the IoT C++ SDK from GitHub:

- Update the repository: $ git pull origin master
- Follow the instructions given in Installation to rebuild the IoT C++ SDK.

Install the C++ SDK in Your Application

The C++ SDK is packaged as a tar zip file and contains the following:

- **include** - directory containing headers for the public API for use with the C++ SDK
- **lib** - the directory containing the shared object files for the C++ SDK

Add the SDK Headers

Copy or move the `include` directory, which contains the SDK header files into your application directory, and include it in your code to access the SDK APIs.

```cpp
#include "appd_iot_interface.h"
....
{
```

Initialize the SDK

You must initialize the C++ SDK by providing the SDK and device configuration as input parameters and then calling the function `appd_iot_init_sdk` as shown below. The SDK configuration takes in parameters for the app key, log level, and the EUM Collector URL. The SDK uses the EUM Collector URL to send data to the EUM Server. The device configuration contains information to identify a unique device.
#include "appd_iot_interface.h"

....
{
    // Declare config variables for the SDK and device.
    appd_iot_sdk_config_t sdkcfg;
    appd_iot_device_config_t devcfg;
    appd_iot_init_to_zero(&sdkcfg, sizeof(sdkcfg));
    appd_iot_init_to_zero(&devcfg, sizeof(devcfg));

    // Set the initialization configurations for the SDK
    sdkcfg.appkey = "<EUM_APP_KEY>";

    // Set the device configurations
    devcfg.device_id = "1111";
    devcfg.device_type = "SmartCar";
    devcfg.device_name = "AudiS3";

    // Initialize the instrumentation
    appd_iot_init_sdk(sdkcfg, devcfg);
}
Add and Send Events

To understand the different types of events, you will work with the sample smart car IoT application given in the sections below.

Custom Events

Custom event to capture technical stats of a "SmartCar".
#include "appd_iot_interface.h"

....

{  
    appd_iot_custom_event_t custom_event;
    appd_iot_init_to_zero(&custom_event, sizeof(custom_event));

    custom_event.type = "SmartCar Stats";
    custom_event.summary = "Technical Stats of SmartCar";
    custom_event.timestamp_ms = ((int64_t)time(NULL) * 1000);
    custom_event.data = (appd_iot_data_t*)calloc(2, sizeof(appd_iot_data_t));
    appd_iot_data_set_integer(&custom_event.data[0], "Speed mph", 65);
    appd_iot_data_set_double(&custom_event.data[1], "Oil Temperature", 220);

    appd_iot_add_custom_event(custom_event);
    free(custom_event.data);

    ....

    appd_iot_send_all_events();
}

Network Request Events

Network Request event to capture the performance of an HTTPS call to get weather information.
```c
#include "appd_iot_interface.h"
...
{
    appd_iot_network_request_event_t network_event;
    appd_iot_init_to_zero(&network_event, sizeof(network_event));

    network_event.url = "https://apdy.api/weather";
    network_event.resp_code = 202;
    network_event.duration_ms = 10;
    network_event.req_content_length = 300;
    network_event.req_content_length = 100;
    network_event.timestamp_ms = ((int64_t)time(NULL) * 1000);
    network_event.data = (appd_iot_data_t*)calloc(1, sizeof(appd_iot_data_t));
    appd_iot_data_set_string(&network_event.data[0], "city", "San Francisco");

    appd_iot_add_network_request_event(network_event);
    free(network_event.data);

    ....
    appd_iot_send_all_events();
}
```

**Error Events**

The Error event below is used to capture Bluetooth errors in the SmartCar app.
#include "appd_iot_interface.h"

....

{
    appd_iot_error_event_t error_event;
    appd_iot_init_to_zero(&error_event, sizeof(error_event));

    error_event.name = "Bluetooth Connection Error";
    error_event.message = "connection dropped due to bluetooth exception";
    error_event.severity = APPD_IOT_ERR_SEVERITY_CRITICAL;
    error_event.timestamp_ms = ((int64_t)time(NULL) * 1000);
    error_event.data = (appd_iot_data_t*)calloc(1, sizeof(appd_iot_data_t));
    appd_iot_data_set_integer(&error_event.data[0], "Bluetooth Error Code", 43);

    appd_iot_add_error_event(error_event);
    free(error_event.data);

    ....

    appd_iot_send_all_events();
}

Correlate Business Transactions with Network Requests (Optional)

To correlate business transactions (BTs) with network requests, you need to instrument a business application and enabled business transactions in the Controller UI. See Correlate Business Transactions for IoT Monitoring to learn more.

The steps below show you how to get the BT response headers and use them to correlate the BT with an IoT Network Request event.

1. Set the AppDynamics HTTP headers ADRUM and ADRUM_1 as part of a network request to your business application.
/* Initialize all the data structures for the request and response. */
appd_iot_http_req_t http_req;
appd_iot_http_req_t http_resp;

/* Initialize the request and response. */
appd_iot_init_to_zero(&http_req, sizeof(http_req));
appd_iot_init_to_zero(&http_resp, sizeof(http_resp));

/* Provide the URL to your instrumented business app that is enabled for business transaction correlation. */
http_req.url = "<url-to-your-business-app-enabled-for-bt>";

/* Add your other HTTP request parameters here: */
...

/* Call the SDK method to get the headers for ADRUM and ADRUM_1. */
const appd_iot_data_t* correlation_headers =
appd_iot_get_server_correlation_headers();

for (size_t i = 0; i < APPD_IOT_NUM_SERVER_CORRELATION_HEADERS; i++)
{
    appd_iot_data_set_string(&http_req.headers[i],
correlation_headers[i].key, correlation_headers[i].strval);
}

/* Make the request, and assign the response to a variable. */
http_resp = http_curl_req_send_cb(&http_req);

2. The call will return response headers (i.e., ADRUM_*) that contain information for correlating the business transaction. If you were to print these BT response headers, you would see something like the following:

ADRUM_0: clientRequestGUID:0f5c7602-9b69-4e40-85a6-e0abf288accf
ADRUM_1:
globalAccountName:eum-mobile_4debdbad-3f8e-4f6d-8faf-e5f5781ec0d7
ADRUM_2: btId:3867
ADRUM_3: serverSnapshotType:f
ADRUM_4: btDuration:829

3. Add these BT response headers to the network event that you send to the EUM Server:
/* Create a network event to report to the EUM Server. */
appd_iot_network_request_event_t network_event;
appd_iot_init_to_zero(&network_event,
sizeof(appd_iot_network_request_event_t));

/* Add information about the network event that you want to report. */
network_event.url = "<url-to-your-business-app-enabled-for-bts>";
network_event.resp_code = http_resp->resp_code;

/* Assign the returned BT response headers from the call to the business app to the headers of the request. */
network_event.resp_headers = http_resp->headers;

// Add the network event to beacon to send to the EUM Server. */
appd_iot_add_network_request_event(network_event);
appd_iot_send_all_events();

4. In the Controller UI, you should be able to view the correlated business transaction in the Device Details dialog.

**Compile and Run Your App with the SDK Library File**

1. Compile your program. For example, if the driver file is `main.cpp`:

```bash
$ g++ -c main.cpp -I<appd_iot_sdk_dir>/include
```

2. Create the binary with the object code of your application and linking AppDynamics IoT C++ SDK library.

```bash
Linux

$ g++ main.o <appd_iot_sdk_dir>/lib/libappdynamics_iot.so -o main
```
You can combine steps 1 and 2 into one step as below:

Mac

$ g++ main.o
<appd_iot_sdk_dir>/lib/libappdynamics_iot.dylib -o main

You can combine steps 1 and 2 into one step as below:

$ g++ main.cpp -o main -I<appd_iot_cpp_sdk_dir>/include
-L<appd_iot_cpp_sdk_dir>/lib -lappdynamics_iot

3. Set the environment variable `DYLD_LIBRARY_PATH` to the PATH where the SDK library is installed. This will let dynamic linker know the directory to search for shared libraries.

Linux

$ export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<appd_iot_cpp_sdk_dir>/lib

Mac

$ export
DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:<appd_iot_cpp_sdk_dir>/lib
4. Run your program. For example:

```
$ ./main
```

Verify the Instrumentation in the Controller UI

See [Confirm the IoT Application Reported Data to the Controller](#) to verify the instrumentation.

Customize the IoT C++ Instrumentation (Optional)

You can further customize the IoT C++ instrumentation using the IoT C++ SDK. See the [latest IoT C++ SDK documentation](#) or the previous versions listed below:

- [https://docs.appdynamics.com/javadocs/iot-cpp-sdk/4.5/4.5.0/](https://docs.appdynamics.com/javadocs/iot-cpp-sdk/4.5/4.5.0/)
- [https://docs.appdynamics.com/javadocs/iot-cpp-sdk/4.5/4.5.1/](https://docs.appdynamics.com/javadocs/iot-cpp-sdk/4.5/4.5.1/)
- [https://docs.appdynamics.com/javadocs/iot-cpp-sdk/4.5/4.5.2/](https://docs.appdynamics.com/javadocs/iot-cpp-sdk/4.5/4.5.2/)
- [https://docs.appdynamics.com/javadocs/iot-cpp-sdk/4.5/4.5.4/](https://docs.appdynamics.com/javadocs/iot-cpp-sdk/4.5/4.5.4/)

Run the Sample C++ Application

The sample C++ application sends sample data for Custom, Network Request, and Error events. The data mocks a smart car application, capturing usage information, network performance, and errors.

To run the sample app, follow the instructions given in [Sample Application using IoT C++ SDK](#).
Instrument Applications with the IoT Java SDK

On this page:
- Run the Sample Java Application
- Upgrade the IoT Java SDK

The IoT Java SDK can be included in IoT Java applications running on an edge device like gateways, points of sale, car infotainment centers. This getting started will show you how to install the Java SDK and instrument your IoT application.

Follow the steps below to get your EUM App Key and instrument your IoT C/C++ apps.

1. Check the Requirements
2. Get the IoT Java SDK
3. Add the SDK Dependencies to the Gradle Configuration
4. Add the Instrumentation Code
5. Add and Send Events
6. Verify the Instrumentation in the Controller UI
7. Correlate Business Transactions with Network Requests (Optional)
8. Enable Logging for the SDK (Optional)
9. Customize the IoT Java Instrumentation (Optional)
10. Troubleshooting the IoT Java SDK

This Java SDK differs from the AppDynamics Java Agent in that it is a very lightweight library specially designed for lower-end devices. A lot of flexibility is also built into the way event information can be extended and the instrumentation code configured as well as controlled.

Check the Requirements

Before you get started, make sure you meet the following requirements:

- A device running one of the following version of Java Runtime:
  - Java SE 7
  - Java SE Embedded 7
  - Java SE 8
  - Java SE Embedded 8
- HTTPS interface to send beacons to the EUM Server
- EUM App Key

Get the IoT Java SDK

You can get the Java SDK by cloning or downloading the IoT Java SDK from GitHub. Follow the instructions given in Build the SDK to build the IoT Java SDK.

If you're using the IntelliJ IDE, add the file `lib/appd-iot-sdk.jar` to your project by following the instructions given in Working with module dependencies. Confirm that the JAR file shows up under External Projects in your IntelliJ project.

Upgrade the IoT Java SDK

From the root directory of your clone of the IoT Java SDK from GitHub:

- Update the repository: `git pull origin master`
- Follow the instructions given in Build the SDK to rebuild the IoT Java SDK.

Add the SDK Dependencies to the Gradle Configuration
Add the following to your `build.gradle` file:

```groovy
dependencies {
    runtime group: 'org.slf4j', name: 'slf4j-api', version: '1.7.25'
    runtime group: 'com.google.guava', name:'guava', version:'18.0'
    runtime group: 'com.google.code.gson', name: 'gson', version: '2.8.0'
}
```

Add the Instrumentation Code

**Import the IoT SDK**

In your application file, add the `import` statement that includes the Java IoT SDK:

```java
import com.appdynamics.iot.Instrumentation;
```

**Configure the IoT Java Agent**

Configure the instrumentation by providing the EUM App Key and the URL to the EUM Collector. If the EUM Collector URL is not specified, the default SaaS Collector URL is used.

```java
import com.appdynamics.iot.AgentConfiguration;
AgentConfiguration.Builder agentConfigBuilder =
    AgentConfiguration.builder();
AgentConfiguration agentConfig = agentConfigBuilder
    .withAppKey(<EUM_APP_KEY>)
    .build();
```

**Set Device Info**

You are required to set the name and ID for devices. The name should consist of a short string that identifies the type and model of the device, such as "EV Model 3" or "Thermostat Model Star7". The device ID must be a unique identifier for the device, such as a UUID, the VIN number of a car, or the MAC address of the device.

The example code below sets the device ID to a random UUID and the name to "Smart Shelf".

```java
```
import java.util.UUID;
import com.appdynamics.iot.DeviceInfo;
...
DeviceInfo.Builder deviceInfoBuilder = DeviceInfo.builder("Smart Shelf P1", UUID.randomUUID().toString());
DeviceInfo deviceInfo = deviceInfoBuilder.withDeviceName("Smart Shelf").build();

Set Version Info
You can set the versions for the firmware, hardware, OS, and software as shown below.

import com.appdynamics.iot.VersionInfo;
...
VersionInfo.Builder versionInfoBuilder = VersionInfo.builder();
VersionInfo versionInfo = versionInfoBuilder
  .withFirmwareVersion("2.3.4")
  .withHardwareVersion("1.6.7")
  .withOsVersion("8.9.9")
  .withSoftwareVersion("3.1.1").build();

Initialize the Agent
To initialize the agent, pass the AgentConfiguration object, the DeviceInfo object, and the VersionInfo object to the start method:

Instrumentation.start(agentConfig, deviceInfo, versionInfo);

Build and Run the Application
Use your favorite Java IDE or CLI environment to build and run the application. Note that the AppDynamics IoT Java SDK needs to be in the build and runtime classpath.

For instructions on adding libraries to the classpath:
- IntelliJ IDE: Creating a library
- Eclipse: Classpath Variables
- Linux/Mac/Windows: PATH and CLASSPATH
For instructions to build and run the app:

- IntelliJ: Building and Running the Application
- Eclipse: Running your programs
- Gradle: Building Java Projects with Gradle

Add and Send Events

The following sections will show you how to create and send the supported events: Custom, Network Request, and Error.

Create a Basic Custom Event

A Custom event can be used to report any performance, device, or business logic data. It is the most general, configurable and flexible data type available.

The Custom event builder takes two required parameters.

- **Event Type:** A short human-readable description of the event, such as "FL Pressure Drop".
- **Description:** A string describing the event, such as "Front Left Tire Pressure Drop".

To make reporting this event meaningful, it is recommended that you provide a timestamp and at least one kind of datatype.

1. Create a basic custom event.

```java
import com.appdynamics.iot.events.CustomEvent;
...
CustomEvent.Builder builder = CustomEvent.builder("FL Pressure Drop", "Front Left Tire Pressure Drop");
long eventStartTime = System.currentTimeMillis();
long duration = 6000;
builder.withTimestamp(eventStartTime).withDuration(duration);
builder.addLongProperty("PSI Drop", 37);
CustomEvent customEvent = builder.build();
```

Additional information can be added to the `CustomEvent`. For details, see the `CustomEvent` class in the latest Java IoT SDK documentation.

2. Add the custom event to the instrumentation (this adds it to the in-memory buffer).

```java
Instrumentation.addEvent(customEvent);
```

3. Send all the events to the EUM Server. This is a blocking call, so the application can send it on a separate thread as shown above.

```java
Instrumentation.sendAllEvents();
```
Send a Network Event

1. Report a Network Request Event using the `HttpRequestTracker` class. This call automatically adds an event to the in-memory buffer, so you need to explicitly import the class.

```java
import com.appdynamics.iot.HttpRequestTracker;
...
String url = "http://ip.jsontest.com/?callback=showMyIP";
// Add a Network Event
try {
    URL thisUrl = new URL(url);
    // [AppDynamics Instrumentation] Get a Tracker
    HttpURLConnection con = (HttpURLConnection)
        thisUrl.openConnection();
    final HttpRequestTracker tracker =
        Instrumentation.beginHttpRequest(thisUrl);
    con.setRequestMethod("POST");
    con.setRequestProperty("Accept-Language", "en-US,en;q=0.5");

    int responseCode = con.getResponseCode();
    con.setDoInput(true);
    con.setDoOutput(true);
    DataOutputStream wr = new
        DataOutputStream(con.getOutputStream());
    wr.flush();
    wr.close();
    System.out.println("Response Code :" + responseCode);

    // [AppDynamics Instrumentation] Retrieve the headers from the response
    Map<String, List<String>> headerFields = null;
    System.out.println("Sending 'POST' request to URL :" + url);
    BufferedReader in;
    String inputLine;
    new InputStreamReader(con.getErrorStream()));
    if (responseCode >= 200 && responseCode < 300) {
        in = new BufferedReader(new
            InputStreamReader(con.getInputStream()));
    } else {
        in = new BufferedReader(
    } else {
        in = new BufferedReader(
            new InputStreamReader(con.getInputStream()));
    if (headerFields != null && headerFields.size() > 0)
            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);
            }
    in.close();
    // [AppDynamics Instrumentation] Initiate adding
    NetworkRequestEvent
```
if (responseCode >= 200 && responseCode < 300) {
    tracker.withResponseCode(responseCode).withError(response.toString()).reportDone();
    .withResponseHeaderFields(headerFields)
    .reportDone();
} else {
    tracker.withResponseCode(responseCode).reportDone();
} else {
    tracker.withResponseCode(responseCode)
}
// End: Add for AppDynamics Instrumentation - Initiate adding NetworkRequestEvent
} catch (MalformedURLException e) {
    e.printStackTrace();
1. Catch the exception:

```java
} catch (Exception ex) {
    ex.printStackTrace();
}
```

2. Send all the events to the EUM Server. This is a blocking call. The application can send it on a separate thread. The recommendation is to batch a number of events together before calling the `sendAllEvents` method.

```java
Instrumentation.sendAllEvents();
```

**Send an Error Event**

1. Report an Error Event using the API.

```java
try {
    // Force creating an exception
    float f = (5 / 0);
} catch (Throwable t) {
    Instrumentation.addErrorEvent(t, Instrumentation.Severity.ALERT);
}
```

2. Send all the events to the EUM Server. This is a blocking call. The application can send it on a separate thread.

```java
Instrumentation.sendAllEvents();
```

**Verify the Instrumentation in the Controller UI**

See [Confirm the IoT Application Reported Data to the Controller](#) to verify the instrumentation.

**Correlate Business Transactions with Network Requests (Optional)**

To correlate business transactions (BTs) with network requests, you need to have instrumented a business application and enabled business transactions in the Controller UI. See [Correlate Business Transactions for IoT Monitoring](#) to learn more.

The steps below show you how to get the BT response headers and use them to correlate the BT with an IoT Network Request event.

1. Make a network request that includes the AppDynamics HTTP request headers `ADRUM` and `ADRUM_1` to one of your business...
import com.appdynamics.iot.HttpRequestTracker;
...
// Create a network request to the business app.
String url = "<url_to_business_application>";
URL thisUrl = new URL(url);
[AppDynamics Instrumentation] Get a Tracker
HttpURLConnection con = (HttpURLConnection)
thisUrl.openConnection();
final HttpRequestTracker tracker =
Instrumentation.beginHttpRequest(thisUrl);
con.setRequestMethod("POST"); // Some HTTP method: GET, POST, PUT...

// Add the AppDynamics HTTP headers ADRUM and ADRUM_1 to the request.
con.setRequestProperty("ADRUM", "isAjax:true");
con.setRequestProperty("ADRUM_1", "isMobile:true");

// Make the request to your business app.
con.setDoInput(true);

2. The call will return response headers that contain information for correlating business transaction. If you were to print these BT response headers, you would see something like the following:

{
ADRUM_1=[globalAccountName:customer1_78203698-278e-428f-8726-bb381219c6cb],
null=[HTTP/1.1 200 OK],
ADRUM_0=[clientRequestGUID:2ff45113-6746-4c94-b6d0-4af26055613c],
ADRUM_3=[btERT:269],
ADRUM_2=[btId:4423],
Server=[Jetty(9.4.z-SNAPSHOT)],
ADRUM_5=[btDuration:327],
ADRUM_4=[serverSnapshotType:f],
Content-Length=[514],
}

3. Send a beacon containing the BT response headers to the EUM Server:
// Fetch the response headers, which will include the BT headers (ADRUM_0, ADRUM_1, ...).
Map<String, List<String>> headerFields = con.getHeaderFields();

// Add the BT response headers to the request body of the Network Request event.
// that you're reporting.
tracker.withResponseCode(responseCode).withError(response.toString()).reportDone();
    .withResponseHeaderFields(headerFields)
    .reportDone();

// Report the Network Request event to the EUM Server.
Instrumentation.sendAllEvents();

4. In the Controller UI, you should be able to view the correlated business transaction in the Device Details dialog.

Enable Logging for the SDK (Optional)

The IoT Java SDK uses the Simple Logging Facade for Java (SLF4J) as the logging framework. You can use your favorite logging engine that is compatible with SLF4J.

If no binding is found on the classpath, then SLF4J will default to a no-operation implementation and display console messages like the following:

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation //
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for
further details.
```

To use the java.util.logging engine, add the following line to the build.gradle file:

```
dependencies {
    ....
    runtime group: 'org.slf4j', name: 'slf4j-jdk14', version: '1.7.25'
    ....
}
```
To see all the debug messages from the library, append the following line to the bottom of the file /Library/Java/JavaVirtualMachines/<your-jdk-version>/Contents/Home/jre/lib/logging.properties:

```
com.appdynamics.iot.level = FINEST
```

**Customize the IoT Java Instrumentation (Optional)**

You can further customize the IoT Java instrumentation using the IoT Java SDK. See the latest IoT Java SDK documentation or the previous versions listed below:

- [https://docs.appdynamics.com/javadocs/iot-java-sdk/4.5/4.5.0/](https://docs.appdynamics.com/javadocs/iot-java-sdk/4.5/4.5.0/)
- [https://docs.appdynamics.com/javadocs/iot-java-sdk/4.5/4.5.1/](https://docs.appdynamics.com/javadocs/iot-java-sdk/4.5/4.5.1/)
- [https://docs.appdynamics.com/javadocs/iot-java-sdk/4.5/4.5.2/](https://docs.appdynamics.com/javadocs/iot-java-sdk/4.5/4.5.2/)
- [https://docs.appdynamics.com/javadocs/iot-java-sdk/4.5/4.5.4/](https://docs.appdynamics.com/javadocs/iot-java-sdk/4.5/4.5.4/)

**Run the Sample Java Application**

The sample Java application sends sample data for Custom, Network Request, and Error events. The data mocks a smart car application, capturing usage information, network performance, and errors.

To run the sample app, follow the Getting Started instruction given in the [iot-java-sdk GitHub repository](https://github.com/appdynamics/iot-java-sdk).

**Troubleshooting the IoT Java SDK**

This section provides instructions for debugging common issues.

**Unable to Link the IoT Java Agent**

If you are getting the following error when trying to link the IoT Java Agent, it's because of a dependency on log4j.
loader constraint violation: when resolving method
"org.slf4j.impl.StaticLoggerBinder.getLoggerFactory()Lorg/slf4j/I_LOGGER_FACTORY;" the class loader (instance of com/intellij/ide/plugins/cl/PluginClassLoader) of the current class, org/slf4j/ILoggerFactory, and the class loader (instance of com/intellij/util/lang/UrlClassLoader) for the method's defining class, org/slf4j/impl/StaticLoggerBinder, have different Class objects for the type org/slf4j/I_LOGGER_FACTORY used in the signature
java.lang.LinkageError: loader constraint violation: when resolving method
"org.slf4j.impl.StaticLoggerBinder.getLoggerFactory()Lorg/slf4j/I_LOGGER_FACTORY;" the class loader (instance of com/intellij/ide/plugins/cl/PluginClassLoader) of the current class, org/slf4j/ILoggerFactory, and the class loader (instance of com/intellij/util/lang/UrlClassLoader) for the method's defining class, org/slf4j/impl/StaticLoggerBinder, have different Class objects for the type org/slf4j/I_LOGGER_FACTORY used in the signature
at org.slf4j.LoggerFactory.getLoggerFactory(LoggerFactory.java:273)
at org.slf4j.LoggerFactory.getLogger(LoggerFactory.java:241)
at org.slf4j.LoggerFactory.getLogger(LoggerFactory.java:254)
at com.appdynamics.iot.Instrumentation.<clinit>(Instrumentation.java:39)
...

To correct the problem, you need to remove the dependency that you added to enable logging. Thus, remove the line specifying the group org.slf4j shown below from dependencies:

dependencies {
    ....
    runtime group: 'org.slf4j', name: 'slf4j-jdk14', version: '1.7.25'
    ....
}
Instrument Applications with the IoT REST APIs

On this page:
- Run the Sample Python Application

The IoT REST APIs enables you to report instrumentation data directly to the EUM Server. You can use any platform or language that has support for HTTPS requests and JSON.

This getting started describes how to create a JSON request body, form the resource URI, and make an HTTPS request to the IoT REST APIs to report instrumentation data for the three supported events.

Follow the steps below to get your EUM App Key and use the IoT REST API:

1. Check the Requirements
2. Form the IoT REST URLs
3. Create the JSON Request Body
4. Transmit the Beacon Data
5. Verify the Instrumentation in the Controller UI
6. Correlate Business Transactions with Network Requests (Optional)
7. Customize the IoT REST API Instrumentation (Optional)
8. Troubleshoot the IoT REST API Instrumentation

Check the Requirements

Before you get started, make sure you meet the following requirements:

- Get an EUM App Key
- Platform/language that supports HTTPS requests.
- JSON support

Form the IoT REST URLs

To form the IoT Monitoring REST resource URL, you will need to know the IoT REST API base URL and port as well as your App Key.

IoT REST API Base URL

The IoT REST API base URL depends on your deployment:

https://iot-col.eum-appdynamics.com/eumcollector/iot/v1

IoT Endpoints

With your App Key, you can form the IoT resource endpoints. See the Summary of the IoT endpoints for the list of supported resource endpoints and their descriptions.

Create the JSON Request Body

You report device information and events in a JSON request body. The JSON includes an array of beacon objects, with each beacon
object containing device data and events. The array enables you to transmit data from multiple devices in one request to the EUM Server. You can transmit up to 200 beacons per request.

Save the JSON below to a file (e.g., testBeacon.json) and replace the values for the timestamp properties with integers representing the UNIX Epoch time in milliseconds. The JSON contains the three supported events customEvents, networkRequest Events, and errorEvents for a smart thermostat. In the next two steps, you will validate and send the JSON as a beacon to the IoT REST API.
[{
  "deviceInfo": {
    "deviceType": "Thermostat",
    "deviceId": "4e75d70d-a3f9-474b-bacf-0f4a57fa944c"
  },
  "versionInfo": {
    "hardwareVersion": "Board Rev. 13A",
    "firmwareVersion": "123.5.31",
    "softwareVersion": "9.1.3",
    "operatingSystemVersion": "Linux 13.4"
  },
  "customEvents": [
    {
      "timestamp": <UNIX_Epoch_time_in_milliseconds>,
      "eventType": "Temperature Reading",
      "eventSummary": "Temperature: 25° c",
      "doubleProperties": {
        "celsius": 25.0
      }
    }
  ],
  "networkRequestEvents": [
    {
      "timestamp": <UNIX_Epoch_time_in_milliseconds>,
      "duration": 245,
      "url": "https://api.company.com/v1/temperature",
      "statusCode": 200,
      "requestContentLength": 32,
      "responseContentLength": 0,
      "doubleProperties": {
        "reportedTemperature": 25.0
      }
    }
  ],
  "errorEvents": [
    {
      "timestamp": <UNIX_Epoch_time_in_milliseconds>,
      "name": "SQLException",
      "message": "open() failed because db is locked"
    }
  ]
}]

Transmit the Beacon Data

To send the beacon, you post the JSON request body to the /beacons endpoint. Again, in this cURL example, use the JSON you saved to the file testBeacon.json and replace <appKey> with your EUM App Key:

```
curl -v -X POST -d '@testBeacon.json'
https://iot-col.eum-appdynamics.com/eumcollector/iot/v1/application/<appKey>/beacons
```

If the beacons were transmitted successfully, the IoT REST API will return the HTTP Status Code 202:

```
< HTTP/1.1 202 Accepted
```

Verify the Instrumentation in the Controller UI

See Confirm the IoT Application Reported Data to the Controller to verify the instrumentation.

Correlate Business Transactions with Network Requests (Optional)

To correlate business transactions (BTs) with network requests, you need to have instrumented a business application and enabled business transactions in the Controller UI. See Correlate Business Transactions for IoT Monitoring to learn more.

The steps below show you how to get the BT response headers and use them to correlate the BT with an IoT Network Request event.

1. Make a network request that includes the AppDynamics HTTP request headers ADRUM and ADRUM_1 to one of your business applications:

```
```

2. The business application will return response headers that contain information for correlating business transaction. If you were to print these BT response headers, you would see something like the following:
3. Create a beacon file `btCorrelation.json` with the returned BT response headers (only those headers that include ADRUM_*). You assign the returned ADRUM_* response headers from the network event request to the business application to the object `responseHeaders` in the beacon as shown below.
3. Internal request:

```json
[
  {
    'deviceInfo': {
      'deviceId': '1111',
      'deviceName': 'AudiS3',
      'deviceType': 'SmartCar'
    },
    'versionInfo': {
      'hardwareVersion': '1.0',
      'firmwareVersion': '1.0',
      'softwareVersion': '1.0',
      'operatingSystemVersion': '1.0'
    },
    'networkRequestEvents': [
      {
        'url': '<url_to_business_app>',
        'statusCode': 200,
        'responseHeaders': {
          'ADRUM_0': [
            '<value_returned_from_business_app>'
          ],
          'ADRUM_1': [
            '<value_returned_from_business_app>'
          ],
          'ADRUM_2': [
            '<value_returned_from_business_app>'
          ],
          'ADRUM_3': [
            '<value_returned_from_business_app>'
          ],
        },
        'timestamp': 1525226857000,
        'duration': 0,
        'requestContentLength': 0,
        'responseContentLength': 457
      }
    ]
  }
]```

4. Send the beacon containing the BT headers to the EUM Server with a cURL command similar to the one here:

5. For a successful call, the response headers should be similar to the following:

HTTP/1.1 202 Accepted
Cache-Control: private, no-cache, no-store, must-revalidate, max-age=0, proxy-revalidate, s-maxage=0
Expires: 0
Pragma: no-cache
Vary: *
Transfer-Encoding: chunked
Via: 1.1 sjc12-dmz-wsa-5.cisco.com:80 (Cisco-WSA/X)
Connection: keep-alive

6. In the Controller UI, you should be able to view the correlated business transaction in the Device Details dialog.

Customize the IoT REST API Instrumentation (Optional)

You can further customize the IoT instrumentation using the IoT REST API. See the latest IoT REST API documentation or the previous versions listed below:

- https://docs.appdynamics.com/javadocs/iot-rest-api/4.5/4.5.0/
- https://docs.appdynamics.com/javadocs/iot-rest-api/4.5/4.5.1/
- https://docs.appdynamics.com/javadocs/iot-rest-api/4.5/4.5.2/
- https://docs.appdynamics.com/javadocs/iot-rest-api/4.5/4.5.4/

Run the Sample Python Application

The sample Python application uses the IoT REST API to send sample data for Custom, Network Request, and Error events. The Network Request events include correlated business transactions. The data mocks a smart car application, capturing usage information, network performance, and errors.

To run the sample app, follow the instructions given in the GitHub repository iot-rest-api-sample-apps.

Troubleshoot the IoT REST API Instrumentation

The sections below provide instructions for troubleshooting your IoT REST API instrumentation.

Verify Your IoT App Has Been Enabled

Using your App Key, verify that your IoT app has been enabled:
curl -v -X GET
https://iot-col.eum-appdynamics.com/eumcollector/iot/v1/application/<appKey>/enabled

If your App Key has been enabled, you should get the following response:

HTTP/1.1 200 OK
Cache-Control: private, no-cache, no-store, must-revalidate,
max-age=0, proxy-revalidate, s-maxage=0
Date: Sat, 19 Aug 2017 01:20:39 GMT
Expires: 0
Pragma: no-cache
Vary: *
Content-Length: 0
Connection: keep-alive

If the App Key does not exist:

HTTP/1.1 403 Forbidden

Validate Beacons

You can use the validate beacon endpoint (/validate-beacons) to verify that the beacon's JSON request body complies with the REST API schema.

You are not required or recommended to validate beacons before transmitting them. You should only use this endpoint in development for testing and troubleshooting.

In this cURL example, you are verifying that the JSON given in the file testBeacon.json is valid. Replace <appKey> with your EUM App Key.
curl -v -X POST -d '@testBeacon.json'
https://iot-col.eum-appdynamics.com/eumcollector/iot/v1/application/<appKey>/validate-beacons

If the JSON request body containing the beacon data is valid, the IoT Monitoring REST API will return the HTTP Status 200:

HTTP/1.1 200 OK

If the JSON request body is invalid, the IoT REST API will return the HTTP Status 422 and a response body with the description of the error message.

< HTTP/1.1 422 Unprocessable Entity
Confirm the IoT Application Reported Data to the Controller

From the **Getting Started Wizard**, you will see your device listed in the table after the IoT application has reported data to the Controller.

If you left the **Getting Started Wizard**, you can always verify the IoT application has been enabled and reported data by doing the following:

1. In the Controller UI, open **User Experience > Connected Devices**.
2. Check the list of registered connected device applications to verify that the application is registered with the Controller. You can also use view some basic information about the app such as the number of devices, total events, and network request information.
3. Start monitoring your application! See **Configure IoT Application Monitoring**.
Monitor Applications with the IoT Dashboards

On this page:
- Device Dashboard
- Network Request Dashboard
- Error Dashboard
- How to Use IoT Dashboards
- Build Custom Dashboard Widgets
- Choose the Best Graph Type

IoT Monitoring provides you with three dashboards below to view and analyze the performance and usage of your application. Each dashboard provides you with a different aspect of your application’s performance. Within each dashboard, you can view data visualized as widgets or detailed records or create custom widgets based on your chosen parameters.

The page introduces the features, functionality, and goal of each of the following dashboards:
- Devices
- Network Requests
- Errors

Device Dashboard

The Device Dashboard enables you to monitor your device status, custom information, device status, and custom device application information. You can view summary activity or use filters to drill down to view metrics for a specific device.

Using the Device Dashboard, DevOps can ensure that devices are up and running, product managers and business stakeholders can examine device activity and trends, and developers can drill down into results to identify issues affecting single or multiple devices.

Network Request Dashboard

The Network Request Dashboard enables you to view outgoing network requests, network request details, and analyze network requests.
With the **Network Requests Dashboard**, you can do the following:

- Check the availability of a device or service
- Discover slow network requests
- View failed network requests
- Analyze and sort network requests by criteria such as status code, app version, etc.
- Monitor network traffic over time

**Error Dashboard**

The **Error Dashboard** enables you to view crashes, exceptions, and custom errors, both fatal and non-fatal. DevOps can monitor activity list and notify developers of error events. The developers can select the error event to view details and download the stack trace for debugging.
With the **Error Dashboard**, you can do the following:

- Discover new errors
- Assess the scope and impact of errors, such as how many devices are affected, what type of devices are affected, and how seriously affected are the devices.
- Monitor error patterns
- Download stack traces of errors to debug code
- Discover broken devices (failed hardware)

### How to Use IoT Dashboards

Although each dashboard presents different information, they all have the same UI functionality and features. This page provides an overview of the dashboard functionality to help you get started monitoring your devices.

#### Access the IoT Dashboards

1. Open the application in which you are interested.
2. Select one of the dashboards to view:
   - Devices
   - Network Requests
   - Errors

#### View Dashboard Summaries

From each dashboard, you can view aggregated data in predefined and custom widgets. Each dashboard is in view mode by default, so data is read-only, although you can set filters, view details, and resize widgets.

The stack trace will only be available if it is reported with the IoT SDKs or through the IoT REST API.
View Details

To view details, you select a row in one of the tables and click the Details icon. This will bring up the Device Details dialog seen below. Here you can view when the three types of events (Custom Event, Network Request, and Error) occurred on the timeline or by duration. You can also click on the event bar in the waterfall for more details.

Filter Results

Click to view associated business transaction.
You can filter results by adding criteria. This enables you to focus on factors affecting availability, usage, and performance.

In the screenshot below, the criteria softwareversion and Country are added, which could potentially home in on performance issues caused by a particular software version in a particular country.

Modify Content and Presentation

To modify the dashboard, you need to be in Edit mode. You simply change the toggle to Edit, and then you'll be able to add, move, resize, remove, and even create widgets.

Build Custom Dashboard Widgets

Each dashboard has predefined widgets for basic monitoring, but because there are so many different devices and platforms, you will need to create custom widgets with the Widget Builder, which guides you through the creation of a widget. This page provides guidance for selecting graph types and using filters when you create widgets.
How to Access the Widget Builder

1. Select a dashboard.
2. Change to Edit mode.
3. From the toolbar, click Add Widget.
4. From the dropdown, click Create a Widget.

Choose the Best Graph Type

The Widget Builder allows you to choose from one of six types of graphs, you should try to use the best graph for your data set. The following is a guideline for choosing the graph type:

<table>
<thead>
<tr>
<th>Graph Type</th>
<th>Graph Properties</th>
<th>Use Case(s)</th>
<th>Example Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pie Chart</td>
<td>• Field</td>
<td>Use pie charts to compare parts of a whole.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Subfield</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Filters</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use pie charts to compare parts of a whole.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use pie charts to compare parts of a whole.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use pie charts to compare parts of a whole.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Bar Chart</strong></td>
<td><strong>Field</strong></td>
<td><strong>Subfield</strong></td>
<td><strong>Filters</strong></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Use bar charts to compare different items within the same category.</td>
<td>Examples:</td>
<td>For an inventory device, the number of items per store.</td>
<td>For a smart car, the mileage of different cars.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Time Series</strong></th>
<th><strong>Field</strong></th>
<th><strong>Filters</strong></th>
<th><strong>Name</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Use a time series to measure events over time.</td>
<td>Examples:</td>
<td>network errors</td>
<td>sales</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Histogram</strong></th>
<th><strong>Field</strong></th>
<th><strong>Filters</strong></th>
<th><strong>Name</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Use histograms to analyze and understand the distribution of large sets of numerical data.</td>
<td>Examples:</td>
<td>The distribution of network request duration.</td>
<td>For a smart home device, the distribution of temperatures.</td>
</tr>
<tr>
<td>Field</td>
<td>Filters</td>
<td>Name</td>
<td>Use a number graph to highlight an important data point. Examples:</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>------</td>
<td>---------------------------------------------------------------</td>
</tr>
</tbody>
</table>
|      |         |      | • active devices  
|      |         |      | • sales total  
|      |         |      | • network errors  |

<table>
<thead>
<tr>
<th>Table</th>
<th>Field</th>
<th>Filters</th>
<th>Name</th>
<th>Use a table to view detailed data for a list of similar items. Examples:</th>
</tr>
</thead>
</table>
|      |      |         |      | • customers  
|      |      |         |      | • network requests  
|      |      |         |      | • devices  |

<table>
<thead>
<tr>
<th>Geo</th>
<th>Field</th>
<th>Filters</th>
<th>Name</th>
<th>Use the Geo widget to analyze and understand performance across geographic areas. Examples:</th>
</tr>
</thead>
</table>
|     |       |         |      | • Sales in different countries, regions, and cities  
|     |       |         |      | • Network request durations across countries, regions, and cities  
|     |       |         |      | • Device errors in different countries, regions, and cities  |

### How to Use Filters

Filters enable you to narrow your results to meaningful information. For example, suppose you wanted to see the total number of items sold at each store, but you were particularly interested in those more expensive items. You could choose a bar graph and use the field **Device Name** and the subfield **Annual Mileage** with the filter **Location: Quebec, Canada** to view the average annual mileage of cars in Quebec, Canada, as shown in the example widget below.
Configure IoT Application Monitoring

On this page:

- Access the IoT App Configuration
- Enable/Disable IoT Monitoring
- Name IoT Network Requests
- Exclude IoT Network Requests

In addition to enabling and disabling IoT Monitoring, you can also configure the display names of network requests and exclude network requests matching given criteria from being monitored.

You can:

- Use the AppDynamics default naming rule, which you can leave as is or modify.
- Disable/modify the default naming configuration.
- Create custom include rules to override the default convention.
- Create custom exclude rules to exclude from monitoring network requests that meet certain criteria.

Access the IoT App Configuration

To access connected device configuration:

1. Open the IoT application in which you are interested.
2. From the left-hand navigation menu, click Configuration.

Enable/Disable IoT Monitoring

From the Configuration page, toggle the Connected Device Monitoring switch to ON to enable monitoring or OFF to disable monitoring.
Name IoT Network Requests

The following sections show you how to modify the default naming configuration for network requests and create include naming rules for network requests.

Access Network Requests Rules

From the Configuration page, click the Monitor tab if it’s not selected already.

Default Network Request Naming Configuration

By default, AppDynamics names network requests using:

- the hostname
- the first two segments of the URL

For example, if an application makes this HTTP request: http://myapp.com/friends/profiles/12345

The default name that is displayed in the Controller UI for that request is: myapp.com/friends/profiles

If this is adequate for your needs, you can leave the default as is. The naming rules you configure here apply to all the IoT applications that are in the same IoT App Group.

Modify the Default Naming Configuration Rule

You may want to configure a different default rule for naming your network requests to help you visualize the parts of your application more clearly. The task is similar to configuring naming rules for business transactions on the server side. Try to group logically related requests together while keeping unrelated requests in separate groups.
If the default hostname and first two segments of the URL for all your requests are identical, you might want to name the requests based on the last segments or a selection of non-contiguous segments of the URL to distinguish among requests in the network requests list.

You can also name the requests based on query parameters. For example, if the request passes an order number, you could specify that the value of the order-number query parameter is used in the network request name.

You can also base the name on a regular expression run on the URL. AppDynamics uses the Java libraries for regular expressions. For more information see:

- Javadoc: http://download.oracle.com/javase/1.5.0/docs/api/java/util/regex/Pattern.html

**Modifying the default network request naming rule**

The default configuration covers how all your requests are named if you do not customize them further.

1. From the **Network Request** tab, scroll down to the **Include Rules** section.
2. Double-click **Default Naming Configuration**.
3. In the **Include Rule** dialog, select the elements you want to use for your default network request naming.
4. Click **OK**.
5. Click **Save**.

**Create IoT Network Request Include Rules**

By default, the same request naming rule is applied to every URL that your application requests. If you want to apply different naming rules to different URLs, create include rules.

For example, if some requests call your own in-house server and others call out to a third-party API, you may want to see all the third-party API calls as one network request and use the default naming rules for the calls to your own server. You would create a custom naming rule that matches the third party calls and uses only the host in the default rule name or perhaps also include certain query parameters.

**Creating an Include Rule**

1. From the **Network Request** tab, scroll down to the **Include Rules** section.
2. Click **Add**.
3. In the **Include Rule** dialog, enter a name for the custom rule that you are creating.
4. Check the **Enabled** checkbox to enable the rule.
5. Select the checkboxes and radio buttons and enter the match criteria for AppDynamics to use to name network requests.
6. Click **OK**.

**Sample Include Rule**

The following rule creates a custom match rule for requests in which the URL contains "inventory". This rule uses the protocol, the subdomain and the third and fourth segments of the URL in the network request name.
You can temporarily cancel the application of a custom naming rule by clearing the Enabled checkbox in the custom rule configuration. In this case, the default naming rule is applied to requests that would have been named by the disabled custom rule. To remove the rule permanently, select the custom rule in the Custom Naming Rules list and click the Delete icon.

Exclude IoT Network Requests

If there are certain types of requests that you do not want to monitor, create custom exclude rules for them based on the URL and/or the application name. Excluded network requests are not reported or counted toward the network request limit of 500 requests per controller application.

Creating an exclude rule

1. From the Network Request tab, scroll down to the Exclude Rules section.
2. Click Add.
3. In the Exclude Rule dialog, enter a name for the exclude rule that you are creating.
4. Check the Enabled checkbox to enable the rule.
5. Select the check boxes and radio buttons and enter the match criteria for AppDynamics to use to name network requests.
6. Click OK.

You can temporarily cancel the application of an exclude rule by clearing the Enabled checkbox in the exclude rule configuration. To remove the rule permanently, select the exclude rule in the Exclude Rules list and click the Delete icon.

Change Priority of Rules

Rules are evaluated in the order that they appear in the include or exclude list. You can change the priority of the rules by dragging and
dropping rules towards the top (higher priority) or towards the bottom of the list (lower priority). Custom rules are always evaluated before the default naming rule, beginning with the custom rule that has the highest priority.

Your IoT app may make various kinds of network requests, and not all of them may be equally important to monitor in detail. For example, any requests to Google Analytics that your app may make are useful but probably aren’t as important to analyze as the requests it makes to your backend.

To manage the impact on your overall Events Service usage, you can create rules which specify which of these network requests should be sent on to the Event Service, either by excluding a request entirely, including a particular request or a sample of that request types by percentage, or by simply allowing the request to be sent on.

In general, the behavior follows this pattern:

- If no rules are specified, data on all network requests are sent on.
- If exclude rules are specified, and a network request satisfies a rule, that data is not sent on.
- If include rules are specified, any network request that satisfies a rule is sent on, based on sampling defined by the percentage indicated in the rule.
- If both include and exclude rules are specified, a network request that satisfies an include rule but does not satisfy an exclude rule is sent on.
IoT Monitoring Licenses

This topic describes how to interpret the details of your IoT Monitoring license information.

License Key

This is the unique identifier that AppDynamics uses to associate IoT application data to your account. From a practical perspective, you only need to know this information for troubleshooting purposes. The same key applies to Browser RUM, Browser Synthetic Monitoring, and Mobile RUM services. However, each product has its own types and metrics for allowed usage.

License Editions

The only supported license for IoT Monitoring is the Peak Edition. The Peak edition would include Analytics with the EUM product. See IoT for Connected Devices (SaaS) for details about the license.

Usage Period

The usage period for IoT is always the current month, even if you have a multi-year license.

The usage period begins and the meter resets on the first of each month at 12:00 am Pacific time.

Unique Devices

One unique device is one application instance that is embedded on a connected device (or multiple devices).

Data Retention and Storage

See IoT for Connected Devices (SaaS) for the daily storage limit and data retention policy.

Check Usage

From the License page of your Controller UI, you can view your IoT Monitoring license usage in the Connected Devices widget as shown below. The widget displays the license usage percentage, the number of licensed units, the license period, the allowed monthly unique devices, and the number of active monthly unique devices.
Connected Devices

16.8%

4,023 of 24,000 Monthly Unique Devices

Valid from 02/01/19 12:00:00 PM
Valid to 03/01/19 12:00:00 PM
Units Licensed 24
Overage Allowed
Edition IOT Peak