



Axeda® Platform

Script Developer's

Guide

Version 6.1.3
April 2011

Copyright © 2001 - 2011. Axeda Corporation. All rights reserved.

Portions of Axeda® ServiceLink protected by U.S. Pat. Nos.: 6757714, 7046134, 7117239, 7149792, and 7185014; and EU Patent Nos.: 1350367, 1305712.

Axeda Corporation
25 Forbes Boulevard, Suite 3
Foxborough, MA 02035
USA
(508) 337-9200

Axeda® ServiceLink™ and Axeda® Platform software and services (“Axeda Products”) and Questra® IDM software (“IDM Software”) are protected by contract law, copyright laws, and international treaties. Axeda Products and IDM Software are supplied under license and/or services contracts with Axeda’s customers and only users authorized under the applicable contract are permitted to access and use the Axeda Products and IDM Software. Unauthorized use and distribution may result in civil and criminal penalties.

Use, duplication, or disclosure of Axeda software by the U.S. government is subject to FAR 12.211 and 12.212 which state that Government shall acquire only the technical data and the rights in that data customarily provided to the public with a commercial item or process and commercial computer software or commercial computer software documentation shall be acquired by the Government under licenses customarily provided to the public to the extent such licenses are consistent with Federal law.

Portions of the Axeda Products include one or more open source or other third party software programs. Please refer to the [Open_Source_License_Requirements.pdf](#) included in your Axeda product(s) for important notices and licensing information related to such programs. This information is also available through the Axeda Support Site for authorized customers.

The information in this manual is subject to change without notice and does not represent a commitment on the part of Axeda Corporation. This manual and the accompanying Axeda products are copyrighted by Axeda Corporation and contain proprietary and confidential information of Axeda Corporation. All rights are reserved. Any reproduction and/or distribution without the prior written consent from Axeda Corporation is strictly prohibited, except as permitted under the contract between your company and Axeda Corporation. Please refer to the contract for details.

"Axeda" is a registered trademark and the "Axeda" logo is a trademark of Axeda Corporation. "Questra" is a registered trademark of Axeda Corporation. Axeda ServiceLink, Safe Access, Firewall-Friendly, and Maximum Support are trademarks or service marks of Axeda Corporation. Questra IDM Application Suite, Questra RemoteService, Questra SmartMonitor, Questra SoftwareDirector, Questra SoftwareCourier, Questra TotalAccess, A2B, and Asset-to-Business are trademarks of Axeda Corporation.

All third party brand or product names are trademarks or registered trademarks of their respective companies or organizations and are hereby acknowledged, including without limitation the following:

Microsoft, .Net logo, Access (database software), Active Desktop, Active Directory, Internet Explorer, the Microsoft Internet Explorer logo (graphic only), SQL Server, Terminal Services RemoteApp, Visual C++, Visual InterDev, Visual Studio, Visual Studio logo (graphic only), Win32, Windows, the Windows logo (aka the flag logo, graphic only), Windows NT, Windows start button, Windows Start logo (design), and Windows Vista are registered trademarks of Microsoft Corporation. Sun, Solaris, iPlanet, Java, and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. Axeda Corporation is independent of Sun Microsystems, Inc. Oracle and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. IBM and Cognos are registered trademarks of International Business Machines, Incorporated, in the United States and other countries. Linux is a trademark of Linus Torvalds. RED HAT and JBOSS are registered trademarks of Red Hat, Inc. and its subsidiaries in the US and other countries. SuSE Linux is a trademark of SuSE, Inc. Apache Tomcat and Tomcat are trademarks of the Apache Software Foundation. VxWorks is a registered trademark of Wind River Systems, Inc. Netscape is a registered trademark of Netscape Communications Corporation in the U.S. and other countries. Navigator is also a trademark of Netscape Communications Corporation and may be registered outside the U.S. Accelerated Technology and Nucleus are registered trademarks of Accelerated Technology, Inc. ThreadX is a registered trademark of Express Logic, Inc. NetSilicon and NET+OS are trademarks of NetSilicon, Inc. ARM is a registered trademark of ARM Limited. This product includes software developed by the OpenSSL Project for use in the OpenSSL toolkit ([http:// www.openssl.org/](http://www.openssl.org/)). Contains software developed by the University of California, Berkeley and its contributors. All other brand or product names are trademarks or registered trademarks of their respective companies or organizations and are hereby acknowledged.

Document Number: Documentation/ScriptDevelopment/6.1.3/0411

Table of Contents

Chapter 1

Using This Guide	1-1
About This Guide	1-2
Audience	1-2
Contents	1-2
Getting Help	1-3
Documentation Feedback	1-3

Chapter 2

Overview of Scripting for Axeda® Platform	2-1
Data Collection	2-2
Introduction to Axeda Platform Scripting	2-3
Axeda Platform Scripting: Under the Covers	2-4
Two types of script files	2-4
Sequence Files	2-5
Script Files	2-6

Chapter 3

Writing Scripts for Axeda Platform	3-1
Creating Axeda Scripts	3-2
Example Use for Axeda Scripts	3-2
Writing Scripts	3-3
Best Practices	3-3
Gateway considerations	3-4
Error Handling	3-5
Debugging Scripts	3-5
Configuring Scripts for User Authentication	3-6
Creating Sequence Files	3-7
Format of a Sequence File	3-8
Example Sequence File	3-9
Substitutions and Internal Directives for the Axeda Agent	3-10
Naming a Sequence File	3-10
Adding a Timeout to the Sequence File	3-10
Agent-specific string replacements	3-11
Using Script Substitution Values as Arguments	3-12
Additional directives for the Agent	3-13

Chapter 4	
Syntax for Data Source Files	4-1
The \$(parse) directive	4-2
Notations used in this chapter	4-3
<dataitemblock> elements	4-3
<alarm> elements	4-4
<dataitem> elements	4-6
<event> elements	4-7

Chapter 5	
Snapshots	5-1
Overview of Snapshots	5-2
Creating Scripts that Create Snapshots	5-4
Format for Snapshot Files (DTD)	5-5
Attributes for <snapshot>	5-7
Attributes for <node>	5-7
Attributes for <leaf>	5-8
Attributes for <dataitemblock>	5-8
Attributes for <alarm>	5-9
Attributes for <dataitem>	5-10
Attributes for <event>	5-10
Example Snapshot Files	5-11
Using the Axeda® Snapshot Viewer	5-13
Information Shown in the Axeda Snapshot Viewer	5-14
Viewing the Contents of the Snapshot	5-16
Filtering and Searching the File Contents	5-17
Snapshot Parser Errors	5-18
Comparing Snapshots	5-19

Chapter 6	
Managing Axeda Scripts from ServiceLink Applications	6-1
Managing Scripts from ServiceLink Applications	6-2
Publishing Scripts	6-2
Registering Scripts	6-2
Running Scripts	6-3
Running Scripts Locally	6-4
Running Scripts on Assets Managed by Axeda® Gateway	6-4
Stopping Scripts	6-4
Creating Dynamic Timers	6-4
Script and Timer Statuses and Results	6-6
Agent Script Status	6-6
Last Result Values	6-6
Script Status Messages	6-7

Script Tools in the Axeda Configuration Application	6-8
Viewing and Managing Scripts	6-8
Managing the Registrations for a Script	6-9
Selecting Assets for a Script	6-10
Packages to Run Scripts	6-12
Script Tools in the Axeda Service Application	6-12

Chapter 7

Axeda Builder Tools for Scripts	7-1
Actions to Run Scripts	7-2
Script Tools in Axeda Builder	7-3
Create Scripts	7-3
Create Logic Schemas to Run Scripts	7-4

Appendix A

Scripting Examples	A-1
Example Perl Scripts for Axeda Platform	A-2
Wizard Example Script: Files and Instructions	A-2
Example Sequence File - Wizard.seq	A-3
Example Script - start.pl	A-4
Example Script - snmp.pl	A-4
Example Script - files.pl	A-6
Example Script - env.pl	A-7
Example Script - registry.pl	A-7
Example Script - os.pl	A-8
Example Script - end.pl	A-9
Analyze Log Example Script: Files and Instructions	A-9
analyze.seq	A-10
copy_agent_log_file.pl	A-10
get_debug_agent_log.pl	A-10
snapshot_agent_log.pl	A-11

Index	IX-i
--------------------	-------------



Chapter 1 Using This Guide

This chapter contains the following major sections:

About This Guide explains the intended audience of this guide and summarizes the contents of each chapter contained in this guide.

Getting Help explains where to look for answers to your questions about Axeda® Platform components.

Documentation Feedback explains how you can help us improve Axeda documentation.

About This Guide

This section describes the intended audience for this guide (including assumptions about prerequisites) and presents a summary of the contents of this guide.

Audience

This guide assumes that you are an experienced software developer, with a knowledge of the Perl or other appropriate programming language for developing scripts. Further, it assumes that you are familiar with Axeda® Gateway, Axeda® Connector, Axeda® Builder, Axeda® Platform, and the Axeda® ServiceLink™ Applications.

Contents

This guide contains the following chapters and appendices:

Chapter 1, “Using This Guide”, provides a summary of the content of this guide and alternative sources of information about Axeda products.

Chapter 2, “Overview of Scripting for Axeda® Platform”, first explains the two data collection methods and describes the various tools available in the system for script management. This chapter then defines the components of an Axeda script and explains how scripts become available, where scripts run, how to create dynamic timers, where to view the data from scripts, and where and how to manage scripts

Chapter 3, “Writing Scripts for Axeda Platform”, explains the structure of a sequence file, how the file is processed, and how to use substitution and internal Axeda Gateway and Connector commands in the sequence file. It also provides the example sequence file for an Axeda script for which the entire code is provided in the last section of this chapter.

Chapter 5, “Snapshots”, defines “snapshots” for the Axeda Platform environment, explains their purpose and how snapshot data differs from data items collected by an Axeda Gateway or Connector project, and gives a brief overview of what is required for an Axeda script to create the XML file. This chapter also provides all the details for writing scripts that generate a snapshot, including examples and the snapshot DTD. Finally, it explains how to use the Snapshot Viewer to display snapshots

Chapter 7, “Axeda Builder Tools for Scripts”, explains how to access and briefly how to use the tools available in the ServiceLink Applications for creating, running, registering, and managing scripts. In addition, it explains how to use the Axeda® Builder application to create a sequence file and set up a project to run scripts.

Appendix A, Scripting Examples, provides the Perl script for the examples used in this guide.

An index completes this guide.

Getting Help

If you still have questions after reading the documentation for your Axeda product, you can contact Axeda at <http://help.axeda.com> for help or more information.

Documentation Feedback

As part of our ongoing efforts to produce effective documentation, Axeda asks that you send us any comments, additions or corrections for the documentation. Your feedback is very important to us and we will use it to improve our products and services.

Please send your comments to documentation@axeda.com. Thank you for your help.



Chapter 2 Overview of Scripting for Axeda® Platform

This chapter introduces Axeda® Platform *scripting*. The scripting feature enables users to troubleshoot problems, perform operations, analyze asset configurations, perform data collection, and more, on selected assets. In this chapter, you will learn the various tools available for script management so that you can discover different ways to employ scripting in your own systems. Refer to the following sections:

Data Collection explains the two data collection methods.

Introduction to Axeda Platform Scripting explains the benefits of collecting data using scripts and suggests ways to take advantage of them.

Axeda Platform Scripting: Under the Covers explains the components of Axeda scripting, which include Axeda sequence files that execute scripts you write using a scripting language, data source files, and snapshot files.

Data Collection

The Axeda Platform can collect data from installed systems and assets, and present that data to remote users for analysis, reporting, business logic, and troubleshooting. You may need to generate alarms, deploy updated software, or simply track the changes in data item values over time. Additionally, you may need to collect data based on a system event or on a schedule (such as at the end of a run, or at the end of daily operations). If a question arises about the current configuration and operation of an asset, you may want to capture all the system configuration information from that asset. These are some examples of the types of data that the system can collect and reasons you may need to retrieve it from your assets. The question is: how do you get that data?

Within Axeda Platform, you can collect data in two ways: through Axeda® Gateway and Axeda® Connector-based data items and through Axeda scripting. Each method has its unique purposes and strengths. When you are interested mainly in changes to data values, collecting data items provides a reliable way of getting that information. You can use this data in live displays, trend charts, and calculations for usage, Key Performance Indicators (KPIs), or Service Level Agreement (SLA) analysis. Stored in the Axeda Platform database, this data is available through the Current Data Item Values and Historical Data Item Values pages of the Axeda® Service application. Further, you can configure business logic (called “rules”) that take actions based on specific values for selected data items. You can also configure asset conditions to be set (or unset) based on values of data items, based on the results of data expressions that use received values for selected data items, or based on alarms triggered by specific values for selected data items. Although collecting data items is fast, efficient, and reliable, it does not always provide the flexibility for every data collection need.

The inherent flexibility available with Axeda Platform scripts enables you to collect:

- Large amounts of data immediately (“blocks” of data)
- Configuration information and other parameters hidden in environment variables, database tables, SNMP, COM interfaces, and files
- A different set of information from the various assets in your system

An Axeda script can store data, alarms, and events in two types of XML files, a *snapshot* file and a *data source* file. Using special script directives that the Axeda Agents can interpret and run, you can perform the following actions:

- parse an XML file for data items and write the values found to data items in the Agent project
- parse the XML file for alarms and trigger or modify the alarms
- parse the XML file for events and send events to the Axeda Enterprise server
- upload the alarms to the Enterprise server
- upload snapshot, dependency, log, or error files to the Enterprise server

Introduction to Axeda Platform Scripting

Axeda Platform supports scripts that can retrieve information from an asset, automate repetitive processes, and perform complex operations or data manipulations. When retrieving information, Axeda scripts can store the information in XML snapshot and data source files for processing by the Axeda Agent and in the XML snapshot files for viewing from the Axeda Service application. Scripting thereby enables users of the Axeda® Applications to perform almost any operation on a remote asset that a local operator can perform.

Examples of some of the actions you can perform using Axeda Platform scripts include:

- automating a set of system administration steps
- capturing and parsing complex data structures or logs for upload to customer service personnel
- performing repairs by running calibration utilities, cleaning up disk space, and deleting out-of-date log files.

Unlike collecting data items, running scripts can be based on events or as instructed by a user. Scripts can save the data to XML snapshot files and use metadata instructions to describe or format the data into a hierarchy for readability and presentation. Each snapshot can contain information about many data points, and the data in the snapshots is saved within the file rather than inserted in a database. Scripts can also save data to the simpler XML data source files, from which the Axeda Agents can retrieve data item values, blocks of data item values, alarms, and events, and process them.

To use data items as a means of data collection, you need to configure those data items (using Axeda® Builder) within the Agent project. To use scripts for data collection, you need to write scripts (using your preferred scripting language) that perform the operations you require, write script *sequence* files that call your scripts and perform other operations (such as parsing and uploading files), and set up Script definitions using Axeda Builder or the applications at the Axeda Enterprise server. Using Axeda Builder, you can configure script definitions within the project at design time. Unlike data items, scripts are not necessarily bound to a single model and its configuration. You can write scripts and run them on multiple models of assets. In fact, with scripts, assets do not need to have the same set of data. Using a script, you can collect a dynamic set of data from multiple components and assets in your system.

Axeda Platform Scripting: Under the Covers

Axeda Platform scripts can be installed on assets (as part of installing the Axeda Agent at the manufacturer, for example), or created on the fly and “published” to the assets from the Axeda® Applications. You can download a script to a single asset (for example, to solve a specific problem) or publish a script for use on multiple assets.

Once scripts are registered with assets, the Axeda Agents on the assets can execute the script on demand or at regularly scheduled intervals (based on a timer). To run a script on demand, you can use the Axeda Service application or the Axeda Software Management application. In Axeda Service, you display the Asset dashboard for the target asset and then select the Run action for the script in the Scripts module. In Axeda Software Management, you configure a package that contains the script and an instruction to run the script, and then deploy the package to the Axeda Agent running on the asset. Either way, when the Agent next contacts the Enterprise server, the request to run the script is sent to the Agent. Once it has executed the script, the Axeda Agent sends the results to the Enterprise server. For scripts that collect data, the results typically include a snapshot file, which appears in the Uploaded Files module of the Asset dashboard. You can click the name of the file to display its contents in the Snapshot Viewer. Alternatively, results might include changed data item values, alarms, or events collected from a data source file. For changed data item values, script results might also include results of logic schema actions based on the new data item values.

When you create an Axeda script, it is available for use on the related Axeda Agent only. You can publish the script so that it is available for deployment (download and execute) to other assets of the same model, or for configuration as a “Run Script” action.

Two types of script files

Axeda Platform scripting uses two types of input files: *sequence* files and *script* files. Think of the sequence file as a shell script or batch file that the Axeda Agent runs. The sequence file contains commands that instruct the Agent to execute script files and to perform specific actions. These specific actions are internal to the Agent and referred to as *directives*. At runtime, the Agent uses the order in which script files and/or directives appear in the sequence file as the order in which to execute them.

Script files are the scripts you write using the scripting language that is supported on the machines that are running the Axeda Agent. On Linux and UNIX machines, Perl is a commonly used scripting language; examples of Perl scripts can be found in Appendix A, . On Windows machines, VBScript and CScript are commonly used scripting languages that you may find useful for your purposes.

The next two sections describe the two types of Axeda script files in more detail.

Sequence Files

A sequence file provides the commands for the Axeda Agent to run. It works in a similar manner to shell scripts in the UNIX environment or batch files in a Windows environment. When you select to run an Axeda script on an asset, it is the sequence file that you select to run.

The sequence file, in turn, starts any scripts identified in the sequence file and available on that asset. The sequence file can also tell the Axeda Agent to run external programs or to run directives that are specific to Axeda Agents. The sequence file coordinates the interaction of scripts, external programs, and Axeda Agent directives.

The content of a sequence file has multiple possibilities. The simplest sequence file might contain one script file to execute or one directive that uploads a file from the asset. Another sequence file might contain several commands that run scripts. Still another might contain a few directives for the Axeda Agent. The most complex sequence file contains commands to run scripts and/or start external programs, and directives for the Axeda Agent.

For example, in a sequence file, you might run a script that collects data from the Registry of an asset and creates a snapshot file with that information, and then run the `$(upload)` directive to tell the Axeda Agent to upload that snapshot file to the Axeda Enterprise server. Once uploaded to the Axeda Enterprise server, the snapshot file is available to users of the Axeda Service and Software Management applications.

A sequence file can have three main sections, `[REGISTER]`, `[RUN]`, and `[UNREGISTER]`; of these, typically only the `[RUN]` section is used. For sequence files created using a text editor (instead of Axeda Builder or the scripting wizard of the Axeda Applications), it is also possible that a sequence file has only lines. In any case, each line of the sequence file is a command that the Axeda Agent runs in the order in which it finds the command.

When specifying script files to run within the sequence file, you list the scripts in the order in which you want to run them, in the `[RUN]` section. If you are using directives, you list them in the order in which you want to run them, in the `[RUN]` section. If you are mixing directives and running scripts (or external programs), put them in the order in which you want them executed. When the Axeda Agent receives the request from the Axeda Enterprise server to start an Axeda script, it executes the `[RUN]` lines of the sequence file in sequential order (or all lines, if individual sections are not defined) and sends a status message back to the server.

You can create sequence files several ways: in Axeda Builder; in the Axeda Service or Axeda® Configuration applications (using the Script wizard); as part of configuring a package in the Axeda Software Management application that runs a script; or outside of Axeda Platform in a separate text editor. When created in Axeda Builder, the sequence files are saved, by default, with an `.seq` file name extension. You can save the file to another extension, if necessary. As long as the sequence file contains the correct information in the correct format, the Axeda Agent will be able to run it.

Script Files

Using a scripting language supported by your assets, you create script files outside the Axeda Platform system (for example, in a text editor or a programming language editor). Then, you either add them to the server in order to deploy to an asset, or you include them in the installation of the Axeda Agent during project development. Script files are called and run by the commands defined in the sequence file. These scripts can be written in any scripting language supported by the related asset, including Perl, VBScript, and CScript.

An Axeda Agent sends its list of registered scripts to the server each time it starts or as instructed from the Axeda Applications. This communication ensures that the server knows the scripts currently registered with the asset. Scripts are registered by name, and each must have a unique, case-sensitive name within the related model. If two scripts for a single model have the same name, they are assumed to be the same and the script registered or created last overwrites the original script.

If the script was configured to save information or script results to an XML file, the Axeda Agent creates that file and, when directed to do so, parses it and/or uploads it to the Enterprise server. You can view uploaded *snapshot* files and text files from the Uploaded Files module on the Asset dashboard of the Axeda Service application, or from the Uploaded Files page of the Axeda Software Management application, as long as they conform to the XML Snapshot Document Type Definition (DTD) described later in this book. Figure 2-1 shows an example of an Asset dashboard, with one file listed in the Uploaded Files module.

The screenshot displays the Axeda Service application interface for an asset named '100xk'. The dashboard is organized into several sections:

- Location:** Customer: Default Customer, Location: Default Location.
- Serial number:** 100xk
- Model:** SP_1W6Mon2
- Status:** Good
- Registration:** 10/20/06 1:31 PM
- Last contact:** 10/20/06 1:55 PM (2 hours 43 minutes ago)
- Ping rate:** 20 seconds
- Time zone:** Greenwich Mean Time

The **Recent Actions** section shows a list of completed package deployments:

Time	Status	Details
10/20/06 1:54 PM	Complete	Package Deployed [file = .\win32_configuration_snapshot.xml, device relative path = false, delete after upload = false]
10/20/06 1:51 PM	Complete	Package Deployed [file = .\win32_configuration_snapshot.xml, device relative path = false, delete after upload = false]
10/20/06 1:34 PM	Complete	Package Deployed [file = .\win32_configuration_snapshot.xml, device relative path = false, delete after upload = false]
10/20/06 1:31 PM	Complete	Package Deployed [file = .\win32_configuration_snapshot.xml, device relative path = false, delete after upload = false]

The **Alarms** section shows one active alarm:

Time	Alarm Name	Status
10/20/06 1:55 PM	High CPU Condition Value	(Acknowledge)

The **Uploaded Files** section (highlighted with a red box) shows one file:

Time	File Name
10/20/06 1:54 PM	.\win32_configuration_snapshot.xml

The **Audit Log** section shows a list of events:

Time	Event
10/20/06 1:54 PM	Successfully uploaded file(s): .\win32_configuration_snapshot.xml
10/20/06 1:54 PM	Agent command (id=drmsrver#30, Deployed Package): file = .\win32_configuration_snapshot.xml, device relative path = false, delete after upload = false.
10/20/06 1:54 PM	Device 100xk restarted
10/20/06 1:51 PM	Successfully uploaded file(s): .\win32_configuration_snapshot.xml
10/20/06 1:51 PM	Agent command (id=drmsrver#28, Deployed Package): file = .\win32_configuration_snapshot.xml, device relative path = false, delete after upload = false.

The **Data** section shows system metrics:

Time	Metric	Value
10/20/06 1:54 PM	CPU Utilization	99
10/20/06 1:54 PM	Disk Description C:	N/A
10/20/06 1:54 PM	Disk Percent Free C:	0
10/20/06 1:54 PM	Disk Space Available C:	33344736
10/20/06 1:54 PM	Disk Space Total C:	N/A

Figure 2-1. Service application, Asset dashboard, showing snapshot from asset

To view the snapshot, users can click the name of the file. The Axeda Snapshot Viewer opens, displaying the content of the file in the hierarchy set up by the XML tags. Figure 2-2 shows an example of a snapshot file in the Axeda Snapshot Viewer.

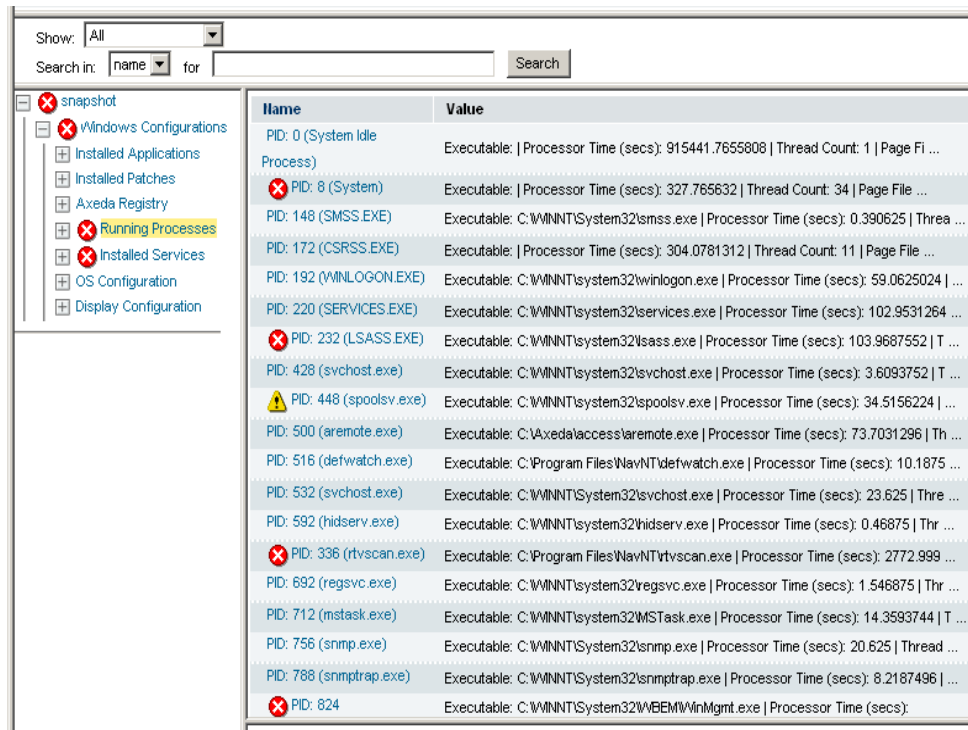


Figure 2-2. Axeda Snapshot Viewer showing contents of snapshot

For example, you can create a sequence file (named *wizard.seq*) that identifies several scripts (in this case, Perl-based). These scripts retrieve asset and SNMP information, format that information as an XML snapshot, and then send that snapshot back to the Axeda Enterprise server. In this example, the Axeda script consists of one sequence file, *wizard.seq*, and the following Perl scripts:

- ◆ *start.pl* — prints asset information, *snapshot.xml* version, and the time the snapshot was created
- ◆ *snmp.pl* — creates an SNMP session, gets SNMP information from the assets connected to this gateway, and returns a session and an error
- ◆ *env.pl* — retrieves information about the asset's environment
- ◆ *end.pl* — ends printing results to the *snapshot.xml*

The resulting *snapshot.xml* is accessible from the Uploaded Files module on the Asset dashboard of the Service application. This example is explained in full, in Appendix A, .



Chapter 3 Writing Scripts for Axeda Platform

This chapter provides important information for writing Axeda scripts in the following sections:

Creating Axeda Scripts provides a brief overview of writing Axeda scripts.

Writing Scripts explains the requirements for the scripts written in your chosen scripting language and called by the sequence file.

Creating Sequence Files explains the structure of a sequence file, how the file is processed, and how to use substitution and internal Axeda Agent commands in the sequence file.

Creating Axeda Scripts

By now you know that *scripts* in the Axeda Platform environment actually consist of a *sequence* file specifically formatted for Axeda Platform operations, and one or more *script* files, written using the scripting language that your assets support. You can create sequence files using Axeda Builder, the Script wizard in the Axeda Service or Axeda Configuration applications, or a text-based editor (to create them separately from an Axeda application). You create the script files (Perl, VBScript, CScript, or other scripting language supported by your assets) outside of the Axeda Platform tools and then call them in the sequence files. In the sequence file you can use only script files, only Axeda directives, or mix the script files and Agent directives to achieve the results you require. Keep in mind that when selecting a script to run from the Axeda Service or Axeda Configuration applications, it is the name of the sequence file that you select, not the name of a script file.

Example Use for Axeda Scripts

Creating Axeda scripts is a useful alternative to writing drivers when you are interested in only one or two values and then only under certain conditions. Using the scripting language supported by your assets, you can write scripts that retrieve a value from a file or other location on an asset and write the value to an output file (typically a data source file, or if the file must be available for viewing, a snapshot file). The Axeda Agents provide a `$(parse)` directive that you can use in the sequence file to retrieve that value. The `$(parse)` directive looks for data items in the output file, retrieves their values, and writes the values to corresponding data items in the Agent project. This directive can operate on both parse and snapshot files.

You can use the retrieved value in many ways. For example, you might configure the Agent project to trigger alarms, events, or other actions based on the value of the data item. Alternatively, you might configure the Agent project to upload the data item to the Enterprise server and then use the value to trigger business logic on the Enterprise server or to set an asset condition on the Enterprise server. You can also use the value of the data item to coordinate the actions of a script with the actions inside the Agent. Whatever uses you can devise for specific pieces of information (the values of data items), the Axeda Agents and the Axeda Applications provide ways for you to implement them. Scripting and the Agent `$(parse)` directive provide the ways for you to retrieve the values.

Writing Scripts

Axeda Agents read the sequence file first. When the sequence file contains the command `start` and the name of a script file, the Agents can run the named scripts. Typically, you would write the script files first and *then* the sequence file that calls them. If you are installing the script files with the Axeda Agent, make sure that the Agent can find the scripts after reading the sequence file. Either include complete paths to the scripts in the sequence file or store the scripts in the same directory as the sequence file on the asset.

You can write the scripts in any scripting language that is supported by the asset on which the Axeda Agents are running. For Linux and UNIX platforms, the Perl language is commonly used and well suited to Axeda Platform scripting. For Windows platforms, VBScript and CScript are commonly used. Whatever scripting language you choose, the related asset must also support it. Axeda Platform does not require any set structure or syntax for these scripts and does *not* validate or debug the scripts. This guide includes some sample Perl scripts in Appendix A, .

Best Practices

Using your scripting language, write a series of scripts to generate an XML file. For modularity and reuse, each script should collect data from a specific source or category and then send its output into the same XML file. Once the XML file has been created, you can use the directives of the Axeda Agents to parse the XML file for specific pieces of information (values of data items), for alarms, and for events and/or to upload the XML file to the Enterprise server for viewing. After retrieving them from the XML file, the Axeda Agent passes values to data items configured in its project, modifies alarms and/or sends them to the Enterprise server, and sends any events it finds to the Enterprise server.

After writing the scripts and the sequence file, you can configure the Agent project to process the data item values and/or modified alarms as appropriate to your requirements. If appropriate, you configure business logic using the Axeda Applications to process data item values, alarms, and events.

The generated XML file can be a *snapshot* file or a *data source file*. If all you want to do is pass data, alarms and events into the Agent project, a *data source file* is sufficient. If you also want to send the content of the output file to the Enterprise server for viewing, create a *snapshot* file. The Axeda Agents can parse either type of file, but only a valid snapshot file can be displayed by the Axeda Snapshot Viewer.

For the Axeda Snapshot Viewer to display it, the *snapshot* file must conform to the XML snapshot Document Type Definition (DTD), which is maintained on the Enterprise server. , *Snapshots*, lists the content of this DTD and explains its elements and their syntax in detail. A *data source file* is an XML file that requires only a subset of the elements of the Snapshot DTD. The section, "*Substitutions and Internal Directives for the Axeda Agent*" on page 3-10 of this chapter, explains the syntax for the Agent directives (including `$(parse)`) as well as the XML elements and attributes for a data source file.

An Axeda script can call a sequence of programs that process the XML file. These programs can parse the XML file and modify it to add knowledge, such as checking for inconsistencies. The modifications usually either flag elements that may be problems or add elements with error descriptions or alarms.

Note: *When generating snapshot files in XML format, problems can occur if certain characters are found by the XML parser. You must write scripts so that they substitute HTML codes for the following problematic characters: & (substitute with &); "or" (substitute with "); < (substitute with <); > (substitute with >).*

For information on creating sequence scripts, see "*Creating Sequence Files*" on page 3-7. For information on viewing snapshots, see "*Using the Axeda® Snapshot Viewer*" on page 5-13.

Gateway considerations

When you are running scripts from a Gateway project that will collect data from managed assets, keep in mind that the script you are writing in Perl or VBScript, for example, must be able to *communicate* with the managed assets that are the source of the data. If the script responsible for collecting data and generating an output file has no provisions for communicating with remote machines, all generated data in the output file will pertain to the *local* machine (that is, the gateway asset).

The sequence file can refer to the address of the managed asset when invoking the script. Typically, a script that generates an output file accepts an extra argument that identifies the target asset. The argument is usually specified as `$(device.address)` in the sequence file for a script that will run on a managed asset. Even if you set up a sequence file with the asset address, if the Perl or VB script does not expect the argument, it always assumes the local host (the gateway asset).

Error Handling

If a program in an Axeda script fails (such as exiting with an error code or timing out), the Script Manager creates an event. The event can be tied to a notification at the Axeda Enterprise server. Following a program failure, the Axeda script stops.

If a program within an Axeda script has an error (such as failure to log in through Telnet), it should output XML with an `<event>` tag to generate an event within Axeda Platform.

Debugging Scripts

Axeda Agents can use the Script Manager's XML file to report debugging messages for script statuses. An example XML file follows:

```
<?xml version="1.0" standalone="yes"?>
<PersistedData moduleName="xgScript" TerseType="1">
<v>1</v>
<Debug>0</Debug>
</PersistedData>
```

The `<Debug>` element defaults to 0 (off) in Axeda Builder.

To enable script debugging to the Kernel log

This procedure provides two ways to enable script debugging.

1. To enable script debugging by editing an XML file, follow these steps:
 - a) In the project files, locate and open *EScriptManager.xml* (Connector project) or *xgScriptManager.xml* (Gateway project)
 - b) Locate the `<Debug>` element and set the value to 1 (on).
2. To enable script debugging using Axeda Builder, follow these steps:
 - a) Open the project in Axeda Builder.
 - b) For a Connector project, select **Settings** in the Project Window. For a Gateway project, select **Configuration** in the Project Window.
 - c) Double-click the **Scripts** icon to open the Script Settings dialog box.
 - d) Select the **Debug** check box.
 - e) Click **OK** to save the change.
 - f) From the **File** menu select **Save All**.
 - g) If you have not enabled the **Auto download on save** option, download the project files to the asset.

Configuring Scripts for User Authentication

To ensure that only selected individuals have access to run a script, you can direct the script to authenticate user information through the Authentication Helper program (*EAuthHelper.exe*). When the Authentication Helper program runs as directed by a script, it performs the following functions in the order shown:

1. Spawns the specified communication program (for example, Telnet) and connects the pipes to the program's standard input and output.
2. Performs password authentication using the specified "chat-script"; `expect` sequences are received from the communication program and `send` sequences are sent to the communication program.
3. Relays standard input and standard output of the communication program after the authentication phase.

The call to Authentication Helper requires the `user@host` argument. For example:

```
spawn('EAuthHelper dhenry@111.211.21.111')
```

Authentication Helper retrieves the encrypted password and parameters from the password database file for the specified user and host, and then authenticates the user information for the remote system.

Two optional parameters are available for Authentication Helper:

- `-p protocol_name` — Parameter for the name of the protocol to use, followed by the protocol name. For example, `-p telnet`
- `-c "chat script"` — Chat script parameter, followed by the chat script name in quotation marks

If no `-p` argument is provided, the default protocol is `telnet`.

A chat script tells Authentication Helper which text to expect for the user name and password, and which information to send in response to those requests. If no `-c` argument is provided, the default chat script format is:

```
ogin: %u\n assword: %p\n
```

where Authentication Helper expects the text "ogin" (from "login") and sends back the user name in response, and then expects the text "assword" (from "password") and sends back the password in response. Notice that the first character of each word is omitted in the default

argument so that the case of the first letter does not prevent Authentication Helper from finding the expected text. In addition to %u (user name) and %p (password), these special sequences are supported: \n (new line character), \e (null string), and \s (space).

Note: *The Windows Telnet and Windows FTP communication programs do not work with Authentication Helper; third-party Telnet and FTP protocols (such as cygwin) should be used instead. Authentication Helper works only with communication programs that use standard input and output streams to accept user input and deliver program output.*

For more detailed information about how to modify your Perl scripts to run the Authentication Helper, go to <http://help.axeda.com>.

Creating Sequence Files

A sequence file contains commands that run scripts in the order in which they appear in the file. A sequence file can also contain directives, which are commands that are specific to the Axeda Agents. The sequence file is a text-based file in which each line is a separate command to the Axeda Agent.

The same sequence file can contain both external scripts and internal Axeda Agent directives. The content and format of the sequence file controls the interaction between the external scripts, the output files they may create, and Axeda Agent directives that operate on the output files. For example, the sequence file can extract the asset's model and serial numbers, and then run a script that extracts the registry settings of the asset and writes them to a snapshot XML file. The sequence file might then also contain the Agent directive that uploads that file to the Enterprise server for viewing through the Snapshot Viewer (Service application).

Notes: *Make sure assets that will run scripts contain support for the scripting language in which the related scripts are written. For example, the Linux operating system comes with the Perl engine and can run Perl scripts. If an asset does not support a specific programming language, you can use the DOS batch language to run batch scripts (for example, .bat).*

Format of a Sequence File

Typically, a sequence file consists of at least one and possibly three sections: **REGISTER**, **RUN**, and **UNREGISTER**. You do not need to define the sections in any particular order, although it is good practice to list them in the order given below.

1. **[REGISTER]** — When the Axeda Enterprise server sends a “register script” command, the Axeda Agent executes all commands in this section. The Axeda Agent verifies that the script name and path are accurate, and then modifies its configuration to reflect the newly registered script. In this section, the paths specified for the scripts to register must be relative to the Agent’s path on the asset. Finally, the Axeda Agent sends an updated list of its registered scripts to the Enterprise server.
2. **[RUN]** — When the Axeda Enterprise server sends a “run script” command, the Axeda Agent executes any commands in this section. If there are no sections, all commands in the sequence file are run, in sequential order.
3. **[UNREGISTER]** — When the Axeda Enterprise server sends an “unregister script” command, the Axeda Agent executes all commands in this section, deletes the script file at the registered path, and then sends an updated list of its registered scripts to the Enterprise server.

Note: *Each time an Axeda Agent starts up, it sends its list of registered scripts to the Axeda Enterprise server.*

Example Sequence File

The following example sequence file is configured to execute Perl scripts, write the results of the scripts to the *snapshot.xml* file, and then parse the *snapshot.xml* file without uploading. This example could also be rewritten to generate a *parse XML* file, using any name and the XML extension.

```
[REGISTER]
# comment line, you can create directories, move files here

[RUN]
perl start.pl ${device.modelNumber} ${device.serialNumber} > snapshot.xml
perl registry.pl >> snapshot.xml
perl env.pl >> snapshot.xml
perl config.pl >> snapshot.xml
rsh ${device.address} fcstat | fcstat.pl >> snapshot.xml
perl end.pl >> snapshot.xml
perl analyze.pl snapshot.xml
# comment line, now parse the file
$(parse snapshot.xml)

[UNREGISTER]
# comment line, clean up directories, files no longer needed
```

If an Axeda Agent is instructed to run this script, it reads each line in the **[RUN]** section of the sequence file. If there is no defined **[RUN]** section, the Axeda Agent reads and executes all lines in the file, in sequential order. The Axeda Agent passes most commands to the asset's operating system, like a batch file. It performs substitutions on its known set of parameters on each line as it processes. In the preceding example sequence file, the Axeda Agent reads the first line as:

```
perl start.pl X13 32671 > snapshot.xml
```

where X13 is the model number for the asset and 32671 is the serial number of the asset.

This example does not have any commands for registering and unregistering this script.

Substitutions and Internal Directives for the Axeda Agent

The Axeda Agent can perform parameter substitutions on specific strings (case insensitive) defined in the sequence file, replacing them with information from the asset. Table 3-1 on page 3-11 provides the details on these strings. In addition, the sequence file can include commands, called *directives*, that are internal to the Axeda Agent and are *not* passed to the operating system. These directives include the `timeout` directive (described in the section below, "[Adding a Timeout to the Sequence File](#)" on page 3-10), the `parse` directive, and the `upload` directive (these last two are described in Table 3-2 on page 3-13).

Naming a Sequence File

The name of your sequence file is registered with the Axeda Enterprise server within the Axeda Agent's scripting registration message. Each sequence file is identified by a unique (per-model), case-sensitive name. Two sequence files with the same name are assumed to be duplicates of the same script.

Adding a Timeout to the Sequence File

To prevent a script from executing without end (for example, the rare "runaway" Perl script), you can add the following directive to the sequence file.

```
#{timeout = x}
```

where 'x' is the number of seconds before the Axeda Agent will time out the process.

The `timeout` directive defines the maximum number of seconds that a process is allowed to run before it is terminated by the Axeda Agent. You can add this directive to the `[RUN]` section of your sequence file. You can use the directive alone or for specified processes. If you use the directive alone, each process defined after the directive can time out.

The following is an example of using the `timeout` directive alone, where all processes defined after it (for example, Perl processes) will run for 21 seconds before being timed out:

```
[RUN]
#{timeout = 21}
perl startup.pl ${device.modelNumber} ${device.serialNumber}
```

The following is an example of using the `timeout` directive to set the timeout for a specific script process (`startup.pl`, in this example) at 21 seconds:

```
[RUN]
perl startup.pl ${device.modelNumber} ${device.serialNumber}#{timeout = 21}
```

If the process is terminated, the following message appears on the console for the Axeda Agent:

```
INFO      EScriptManager: Script [script name] timeout.  
Terminated process: perl startup.pl model xyz, serial number abc
```

where:

[*script name*] is the name defined for the sequence file (the name of an Axeda script is the name of the sequence file).

perl startup.pl is the name of the process being run.

At this point, the running of the script is terminated and status information for the script is updated to reflect the timeout.

Agent-specific string replacements

Axeda scripts support parameter substitution to integrate Axeda Agent information with scripts. At runtime, the Axeda Agent searches for the strings shown in Table 3-1 (case insensitive), substituting the appropriate values for the parameters.

Table 3-1.String replacements

Symbol	Meaning
#{device.address}	<i>For Axeda Gateway, SNMP-managed assets.</i> The IP address of the asset. Typically, SNMP or other protocols can discover IP addresses.
#{device.dataItems(<i>data_item_name</i>)}	Current value of the data item specified (in place of <i>data_item_name</i>).
#{device.modelNumber}	Model number for this asset.
#{device.password}	The password for network access (telnet or ftp).
#{device.serialNumber}	Serial number for this asset.
#{device.username}	The user name for network access (telnet or ftp).
#{device.community}	<i>For Axeda Gateway, SNMP-managed assets.</i> The SNMP community for an SNMP-discovered asset.
# in the first column	The text that follows is a comment and ignored.

Using Script Substitution Values as Arguments

You can use the supported script substitution strings as a way to pass arguments, for example to other applications on the asset. For instance, you can use the model substitution string to pass the model name as an argument. This substitution is useful for scripts that will run on gateway assets, where the scripts need to communicate with the managed assets in order to collect data, alarms, and events from them.

When using the script substitution strings in this way, it is important to remember that the target program will use the Axeda Agent string *verbatim*. If the string is comprised of two words separated by a space, the target program may see these as two parameters instead of as one parameter joined by a space. In this case, you would need to enclose the script substitution parameter in quotation marks to identify the replacement string as a single value.

For example, for an asset model, *ACME Model*, and asset serial number, *Asset XYZ*, in the following script command:

```
Sample.pl ${device.model} ${device.serial}
```

the script strings are replaced correctly, generating the following snapshot XML:

```
<Snapshot version="1.0" time="time" model="ACME Model" device="Asset XYZ" />
```

To pass the model as a single argument to an application, *MyApplication*, that is designed to read spaces as tokens, you need to enclose the script substitution strings in quotation marks.

- ◆ In this example, `MyApplication.exe "${device.model}"`, the script command would correctly translate to: `MyApplication.exe ACME Model`, where `ACME Model` is seen as one argument.
- ◆ In this example, `MyApplication.exe ${device.model}`, the script command would incorrectly translate to: `MyApplication.exe ACME Model` where `ACME` and `Model` are seen as two separate arguments.

Additional directives for the Agent

Your sequence file can also include the `parse` and `upload` directives, which are internal to the Axeda Agent and not run by the operating system, as described in Table 3-2 on page 3-13 . You can use these directives after generating a data source file or a snapshot file to process alarms, data items, or events in the file. For example, you may want to write the value specified for a data item in a data source file to the data item in the project.

Table 3-2. Internal Agent directives

Syntax	Meaning
<code>\$(parse filename)</code>	The Axeda Agent parses the specified file for data items, alarms, and events. Chapter 4, , explains the actions that the Agent takes when it finds these elements. It also explains and provides examples of the XML syntax for the files that the Agent will parse.
<code>\$(upload [async] [uncompressed] filename [hint or type])</code>	The Axeda Agent uploads the specified file(s) to the Axeda Enterprise server. This directive has the following arguments, in addition to <i>filename</i> : async — When uploading multiple files, the Agent should start the upload of one file and not wait until that upload completes to start the upload of the next file. If you do not use this argument, the Agent uploads the files one at a time, waiting until one upload completes before starting the next uncompressed — When you use this argument, the Agent will skip the usual compression and upload the file. If you do not use this argument, the Agent runs its compression utility before uploading the file. hint or type — For this argument, specify the type of file for the Enterprise server: <code>snapshot</code> or <code>dependencies</code> , <code>log</code> or <code>error</code> . NOTE: If you are using both <code>[async]</code> and <code>[uncompressed]</code> , the order in which you specify them does not matter.



Chapter 4 Syntax for Data Source Files

This chapter explains the syntax you need to use when creating a data source file for the Axeda Agent to process (using the `$(parse)` directive). It contains the following sections:

The `$(parse)` directive explains the difference between the data source file and a snapshot file and then explains the main elements that the Axeda Agent is looking for when it parses a data source file.

Notations used in this chapter defines the XML syntax used to describe the elements that the Agent can find in a data source file.

`<dataitemblock>` elements provides a prototype (of sorts), definitions and possible values for attributes, and an example for the `<dataitemblock>` element.

`<alarm>` elements provides a prototype, definitions and possible values for attributes, and an example for the `<alarm>` element.

`<dataitem>` elements provides a prototype, definitions and possible values for attributes, and an example for the `<dataitem>` element.

The \$(parse) directive

For you to use the \$(parse) directive, scripts must write the data, alarms, and events that they collect to an output file, using an XML-based syntax that the Agent is designed to interpret. This output file can contain just data item, data item block, alarm, and event elements; in this case, it is a *data source* file. Alternatively, this file can contain all the XML encoding needed to display the information as a *snapshot*. If you want to create snapshots, continue to the next chapter, specifically to the section, "*Creating Scripts that Create Snapshots*" on page 5-4. If you want to create data source files, continue with this chapter.

If the sequence file contains a \$(parse) directive, the Axeda Agent reads the specified file and parses the XML, searching for the following elements:

- ◆ **<alarm>** elements — The Axeda Agent modifies the alarm, based on the values of the attributes. If specified, the Axeda Agent also uploads the alarm and any files specified to be uploaded with the alarm.
- ◆ **<dataitemblock>** elements — This element is a container for <dataitem> elements. When it encounters this element, the Axeda Agent treats the <dataitem> elements within it as a block. You may find this element useful when you have an expression that evaluates several data items and you want the expression evaluated once, for the new values of all the data items.
- ◆ **<dataitem>** elements — The Axeda Agent imports the value specified in the element into the named data item of its project. Note that the Axeda Agent does not send the data items it finds while parsing to the Axeda Enterprise server. You must configure a Data Logger in the project that sends the data items to the server. For example, you may want to configure a Data Logger that sends the data items set by a script upon data change, in real time. Refer to the online help for Axeda Builder if you need assistance configuring Data Loggers.
- ◆ **<event>** elements — The Axeda Agent sends each event to the Axeda Enterprise server.

For the Agent to process the information contained in the file, you need to follow an XML-based syntax for each of these elements.

Notations used in this chapter

The next few sections provide the syntax of the four elements, using the following notations:

- ◆ `<element_name>` — The type of element to be written to the file. For example, `<dataitem>`. Most elements have attributes. Elements can also have sub-elements. Note that all attributes appear after the name of the element, within the angle brackets. Separate the attribute `name="value"` pairs with a single space.
- ◆ `attribute="value"` — For elements that have attributes (required or optional), the attributes and values are written as `name="value"` pairs. The values must always be enclosed in double quotation marks.
- ◆ `attribute="true | false"` — For attributes that have an enumerated set of possible values, the possible values are shown in this **boldface** type, separated by a vertical bar.
- ◆ `name="some_string"` — For attributes whose values are variable, the values are shown in this *italic* type, with words separated by an underscore. You do NOT have to use an underscore.
- ◆ `</element_name>` — For best practices, all elements should have a closing tag. The tag is denoted by the forward slash after the opening angle bracket. The element name must match the element name in the main (or opening) tag. A shorthand for the closing tag is to include the forward slash immediately before the closing angle bracket. The `<event>` element allows this shorthand.

Each element described in the next few sections provides a prototype (of sorts), definitions and possible values for attributes, and examples.

<dataitemblock> elements

One `<dataitemblock>` element can contain one or more `<dataitem>` elements. This element has no additional attributes.

Prototype

```
<dataitemblock>  
  ...some <dataitem> elements  
</dataitemblock>
```

For example, your script needs to write a `<dataitemblock>` to the output file, using the following syntax and the values collected by your script, whether literals, as shown here, or as variables from the script):

```
<dataitemblock>
<dataitem name="ScriptDigital" type="digital" quality="good" utc="2007-08-07T08:10:30+500">1</dataitem>
<dataitem name="ScriptString" type="string" quality="good" utc="2007-08-07T08:10:30+500">Sun June 3 01:08:35 2007</dataitem>
</dataitemblock>
```

<alarm> elements

When the Axeda Agent finds `<alarm>` elements, it updates the definition of the matching Alarm Style in the project with the values given for the attributes in the `<alarm>` element. When it encounters the `<upload>` sub-element, the Agent sends the alarm to the Axeda Enterprise server, using the information given in the `<hint>` sub-element of `<upload>`.

Prototype

```
<alarm name="some_string" desc="some_string" severity="some_integer"
active="true | false" ack="true | false" utc="yyyy-mm-ddThh:mm:ss.ms|time-
zone_offset"
<upload hint="snapshot | dependency | log | error">
  <file name="path.filename" delete="n | y" pathDeviceRelative="n | y">
</upload>
</alarm>
```

Where:

- ◆ `ack` — Boolean value, identifies whether the alarm is acknowledged (`true`) or unacknowledged (`false`) on the asset.
- ◆ `active` — Boolean value; whether the alarm is active (`true`) or inactive (`false`) on the asset
- ◆ `desc` — description of the alarm
- ◆ `name` — a unique identifier for the alarm. This name must match the name of an Alarm Style in the project that the Axeda Agent is running.
- ◆ `severity` — current severity value for the alarm. When the Axeda Agent encounters this attribute and a value in the file, it updates the alarm definition in the project with the value.
- ◆ `utc` — timestamp of the alarm item, in UTC format. Notice the required “T” character to signal the start of the time.

- ◆ `upload` — identifies files to upload from the Agent to the Axeda Enterprise server in response to an alarm generated by the Script Manager while running the script. This element is the only way that the Axeda Enterprise server will be able to link the uploaded files to the specific script-based alarm that generated them. The `upload` attribute is a sub element of the `alarm` attribute; its attributes are as follows:

- `hint` — type of file uploaded: `snapshot`, `dependency`, `log`, or `error`.

Below the `hint` sub-element, use a `<file>` element to identify all files of the same `hint` type (for example, all snapshots) to upload. The `<file>` element includes these three attributes:

- `name` — path and name of file(s) to upload; for example, `"/tmp/logs/*.xml"`
- `delete` — whether to delete the file after uploading it; valid values are `"y"` (yes, delete) or `"n"` (no, do not delete)
- `pathDeviceRelative` — for Axeda Gateway, whether the file path is relative to the Axeda Gateway installation path (where `xGate.exe` is installed); valid values are `"y"` (yes, relative) or `"n"` (no, not relative)

An example of an `upload` sub element follows:

```
<upload hint="snapshot">
  <file name="/tmp/logs/*.xml" delete="n" pathDeviceRelative="y">
</upload>
```

This example causes the Axeda Agent to upload all snapshot files in the directory, `/tmp/logs`. The Agent will *not* delete the files from the asset after uploading them. The path to the files is relative to the installation path of the Axeda Agent.

An example of an `<alarm>` element follows:

```
<alarm name="alarm1" desc="alarm description" severity="222" active="true"
ack="false" />
```

<dataitem> Elements

When the Axeda Agent finds <dataitem> elements, it writes the value given for the data item to the corresponding data item in the project it is running. The string in the name attribute for the <dataitem> element must match the name given to a data item in the project for the value to be written.

Prototype

```
<dataitem name="some_string" quality="good | bad | uncertain" type="analog  
| digital | string" utc="yyyy-mm-ddThh:mm:ss.ms|time-  
zone_offset">value_to_be_written_to_Agents_project</dataitem>
```

Where:

- ◆ name — The unique identifier of the data item. For example, name="LaunchStatus".
- ◆ quality — The quality of the data item; valid values are "good", "bad", "uncertain". For example, quality="good".
- ◆ type — The type of data in this data item; valid values are "analog", "digital", "string". For example, type="analog".
- ◆ utc — The timestamp of the data item, in UTC format. For example, utc="2007-08-06T09:17:53.140-5:00". Notice the required "T" character to signal the start of the time.

Note: *If you omit the optional "quality" attribute for a <dataitem> element, the quality is assumed to be good.*

An example of a <dataitem> element follows:

```
<dataitem name="LaunchStatus" type="analog" quality="good"  
utc="2007-08-06T10:22:34.140-5:00">4</dataitem>
```

When it finds this data item element, the Axeda Agent will look for a data item in the project with the name, "LaunchStatus", and the type, "analog". It will set the data item value to 4 and assign the UTC timestamp given in the attribute to the data item.

<event> elements

When it finds <event> elements, the Axeda Agent sends the event information to the Axeda Enterprise server. The attributes of the element provide the name of the event, a message concerning what occurred, its severity, and its timestamp.

Prototype

```
<event name="some_string" message="string_description_of_event"
severity="some_integer" utc="yyyy-mm-ddTThh:mm:ss.ms|time-zone_offset">
</event>
```

Where:

- ◆ message — ASCII string describing the event
- ◆ name — name defined in the project for the event
- ◆ severity — the level of severity of the event
- ◆ utc — timestamp of the event, in UTC format. Notice the required “**T**” character to signal the start of the time.

An example of an <event> element follows:

```
<event name="event1" message="event message" severity="111" utc="2007-08-
07T08:10:30+500" />
```

For an Axeda script to generate a snapshot for display in the Axeda Service application, you need to ensure that the script properly creates the XML file for the Snapshot Viewer. This chapter provides an overview of snapshots and then provides instructions and syntax information, in the following sections:

Overview of Snapshots defines a snapshot for the Axeda Platform environment, explains their purpose in an Axeda Platform system, explains how snapshot data differs from data items collected by an Axeda Agent project, and gives a brief overview of what is required for an Axeda script to create the XML file.

Creating Scripts that Create Snapshots, provides all the details for writing scripts that generate a snapshot, including examples and the snapshot DTD.

Using the Axeda® Snapshot Viewer explains how to use the Snapshot Viewer to display snapshots. It also provides information about parser errors that may occur if the XML in the snapshot does not match the DTD.

Comparing Snapshots explains how to use the File Comparison utility available from the Uploaded Files module on an Asset dashboard to view changes from one snapshot to another.

Overview of Snapshots

A snapshot is a set of data from an asset, collected at a single instant in time, thereby providing a view of the asset activity or configuration at that moment. While this information may not change often, it can help to diagnose problems or confirm configurations.

Snapshots have three distinct uses in the Axeda Platform environment:

- ◆ Provide a source of asset data. The Axeda Agents can parse a snapshot file to find alarms, events, and data items, and then take appropriate actions.
- ◆ Show a hierarchical representation of asset data.
- ◆ Provide dependencies for a package created using the Axeda Software Management application.

A single snapshot can serve one or more of these purposes.

Snapshot data differs from standard asset data sent to the Axeda Enterprise server, as follows:

- ◆ Data items are reported as they change; rules are applied to them by the Axeda Agent or the Axeda Enterprise server. Alarm conditions can be applied to data items; data items can be presented in trend charts, manipulated in various calculations (for example, Key Performance Indicators), and stored in the Axeda Platform database.
- ◆ Snapshot data is collected by a script, which may be scheduled to run periodically or may be run on demand. The data for a snapshot is stored in XML format. A snapshot can be large, including many data points, and is stored as an XML file. Database constraints do not apply to snapshots.

Snapshot XML documents are uploaded to the server, where Axeda Applications users can select them for viewing from the following locations

- ◆ Axeda Service application, Uploaded Files module on the Asset dashboard
- ◆ Axeda Software Management application, View Uploaded Files From Assets page

To create a *snapshot.xml* in the correct format for parsing by the Snapshot Viewer, your script needs to apply the XML elements defined in the snapshot DTD, which is stored on the Axeda Enterprise server. The Axeda Agent creates the snapshot file when running the script, using the XML elements from the script(s).

Users of the Axeda Service and Axeda Software Management applications can view a snapshot in the Axeda Snapshot Viewer, which presents a browser-based explorer tool for searching, filtering, and navigating the contents of the snapshot. The Axeda Snapshot Viewer can display XML files only if their XML elements conform to the definitions in the snapshot DTD. The actual file created does not need to be named *snapshot.xml*, but it does need to be created correctly.

Figure 5-1 shows an example of the Axeda Snapshot Viewer as it first appears when you select a snapshot to display. For more information on the Snapshot Viewer, refer to *"Using the Axeda® Snapshot Viewer"* on page 5-13 and to the online help for the application.

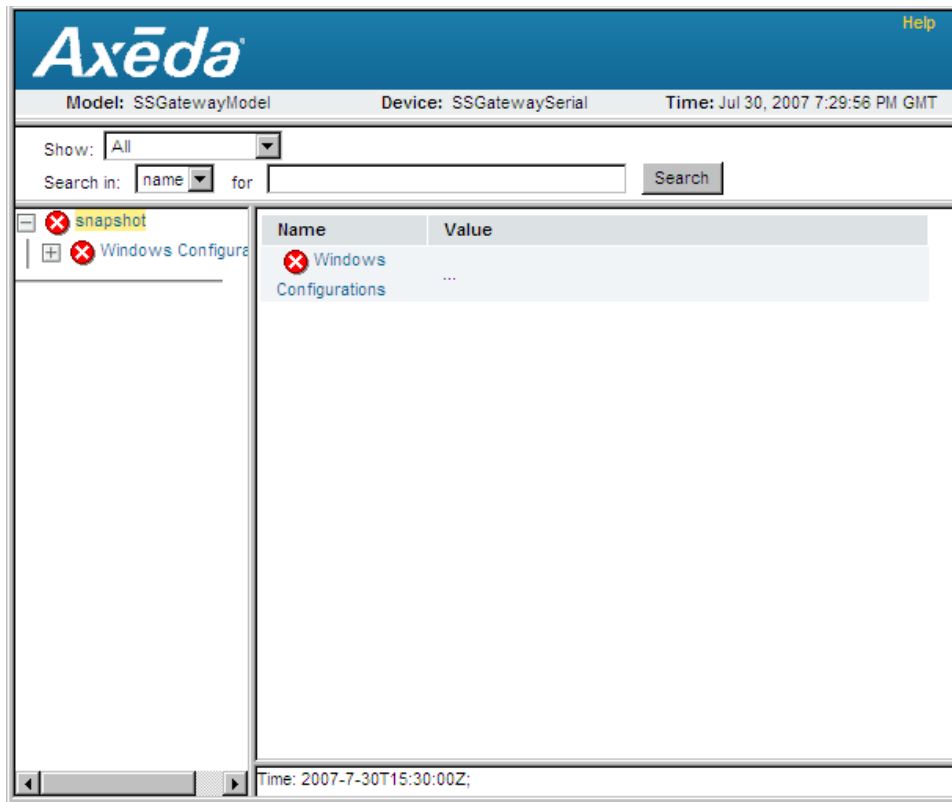


Figure 5-1. Axeda Snapshot Viewer

Creating Scripts that Create Snapshots

As defined by the DTD, the root element of the snapshot XML file is `<snapshot>`. The `<snapshot>` element contains one or more `<node>` elements. A `<node>` element creates a branch of the tree and can contain `<leaf>` and `<dataitemblock>` elements along with additional `<node>` elements. Containing names and values, a `<leaf>` element defines the parameters for the snapshot. Together, these elements create a hierarchical framework for the script data results.

A `<dataitemblock>` element typically contains multiple `<dataitem>` elements that you want to send as a block. For example, when two or more data items are used in evaluating an expression, sending the items as a block ensures that the expression is evaluated once with the *set* of values. Otherwise, the Enterprise server evaluates the expression each time it receives a new value for one of the data items.

In addition to `<leaf>` and `<dataitemblock>` elements, `<node>` elements may also contain `<alarm>`, `<dataitem>`, and `<event>` elements to provide the specified information from the asset to the Axeda Agent for parsing. All elements have associated attributes, some of which have required supported values. For example, the following definition:

```
<dataitem name="tank" type="analog" quality = "good">50</dataitem>
```

sets the value of a data item named `tank`, to 50. The type of this data item is `analog` and the quality is `good`. If a script writes this element to the `snapshot.xml` file, the Axeda Agent can parse the snapshot and set the value of the data item in the project. As long as its project is configured to do so, the Agent can then upload the data item to the Enterprise server.

Tip *To configure the project to upload the data item, configure a Data Logger to be uploaded to the Enterprise server on data change. Refer to the documentation for Axeda Builder if you need assistance.*

Together, the `<node>`, `<leaf>`, `<dataitemblock>`, `<alarm>`, `<event>`, and `<dataitem>` elements create the hierarchical representation of the script data results. Each XML file will have one `<snapshot>` element; the `<snapshot>` can contain multiple `<node>` elements; each `<node>` can contain multiple `<node>`, `<leaf>`, `<dataitemblock>`, `<alarm>`, `<event>`, and `<dataitem>` elements. The next section shows this hierarchy in the XML Document Type Definition (DTD) for snapshots. Immediately following the DTD section are individual sections that describe the attributes for each of these elements and then another section that presents examples of using the elements and their attributes to create a `snapshot.xml` file.

Format for Snapshot Files (DTD)

The XML Document Type Definition (DTD) for snapshot files is stored on the Axeda Enterprise server in the directory, *bea/user_projects/domains/drm/applications/drm/WEB-INF*. The Axeda Snapshot Viewer uses this DTD to validate the XML files uploaded from Axeda Agents. The XML in a snapshot generated by a script must pass the validation before the Snapshot Viewer can display it.

To assist you in reading this DTD, here are some simplified definitions:

- ◆ **!ELEMENT** — denotes the beginning of the definition of an element. The first element in the DTD is the “root” element and must be the first tag in the associated XML file (*snapshot.xml*). The name of the element follows immediately. In parentheses are the names of other elements that this element can contain (called “parent” or “child” elements).

For example, `<!ELEMENT Snapshot (node+)>` defines the root element, `<snapshot>`, and says that it can contain multiple `<node>` elements. The `<node>` element is an example of a parent element; its definition, `<!ELEMENT node ((leaf*, dataitemblock*, node*) | alarm* | dataitem* | event*)>` says that a `<node>` element can contain multiple `<leaf>`, `<dataitemblock>`, *or* `<node>` elements that can be either parent or child elements *and* multiple `<alarm>`, `<dataitem>`, and `<event>` child elements.

- ◆ **!ATTLIST** — denotes the beginning of a list of attributes for the element (whose name immediately follows **!ATTLIST**). For example, `!ATTLIST Snapshot` says that the list of attributes for the `<snapshot>` element is about to start. The first word in the left column is the name of the attribute, which must be used exactly as shown. For a `<snapshot>` element, the first attribute is called `device`.

Tip When specifying values for attributes, keep in mind that the XML rule is to *ALWAYS* enclose the values in double quotation marks. For example, the attribute name for a `<dataitem>` element should be written as follows:
`name="gauge206"`

- ◆ **CDATA** — denotes the type of data that this attribute can contain. **CDATA** denotes *character* data, which means that the parser expects letters and numbers in the text that is between the start and end tag of the element.
- ◆ **#PCDATA** — denotes *parsed* character data. Character data is the text found between the start tag and the end tag of an element. The `<dataitem>` element contains **#PCDATA**, which allows you to write a value to a data item by having the Axeda Agent parse the *snapshot.xml* file.
- ◆ **#IMPLIED** — denotes that the attribute is not required.
- ◆ **#REQUIRED** — denotes that the attribute is required. If it is not present, the parser will reject the snapshot file and the Snapshot Viewer will not display it.

For examples of elements and attributes, refer to *"Example Snapshot Files"* on page 5-11.

The body of each element is a string. The string must be encoded as valid XML. For example, it cannot contain any characters that are reserved in XML, such as an ampersand. If an ampersand occurs in the body, it should be replaced by the character entity reference for an ampersand or by the word “and.” The character entity reference for an ampersand is &. The string may represent a name, a text file, a formula, or a number. In these cases, the string is formatted for viewing by its script. If a snapshot needs to include binary data, that data should be uploaded as a separate file rather than included within the snapshot.

The XML structure for a snapshot, as defined by the DTD, follows.

```
<!-- The DTD for the Snapshot configuration XML document -->
<!--
<!DOCTYPE Snapshot [
-->
  <!ELEMENT Snapshot (node+)>
  <!ATTLIST Snapshot
    device CDATA #IMPLIED
    model CDATA #IMPLIED
    time CDATA #IMPLIED
    version CDATA #IMPLIED>

  <!ELEMENT node ((leaf*,dataitemblock*,node*)|alarm*|dataitem*|event*)>
  <!ATTLIST node
    name CDATA #REQUIRED
    status CDATA #IMPLIED
    desc CDATA #IMPLIED>

  <!ELEMENT alarm (upload*)>
  <!ATTLIST alarm
    ack CDATA #IMPLIED
    active CDATA #IMPLIED
    desc CDATA #IMPLIED
    name CDATA #IMPLIED
    severity CDATA #IMPLIED
    utc CDATA #IMPLIED>

  <!ELEMENT dataitem (#PCDATA)>
  <!ATTLIST dataitem
    name CDATA #REQUIRED
    quality CDATA #IMPLIED
    type CDATA #REQUIRED
    utc CDATA #IMPLIED>

  <!ELEMENT event >
  <!ATTLIST event
    message CDATA #IMPLIED
    name CDATA #REQUIRED
    severity CDATA #IMPLIED
    utc CDATA #IMPLIED>

  <!ELEMENT leaf (#PCDATA)>
  <!ATTLIST leaf
    name CDATA #REQUIRED
    status CDATA #IMPLIED
    units CDATA #IMPLIED
    min CDATA #IMPLIED
    max CDATA #IMPLIED
    url CDATA #IMPLIED
    time CDATA #IMPLIED
    file CDATA #IMPLIED
    type CDATA #IMPLIED
    desc CDATA #IMPLIED>

  <!ELEMENT dataitemblock (dataitem+)>
```

```

<!ELEMENT upload (file+)>
<!ATTLIST upload
  hint      CDATA #IMPLIED>

<!ELEMENT file >
<!ATTLIST file
  name      CDATA #REQUIRED
  delete    CDATA #IMPLIED
  deviceRelative CDATA #IMPLIED>

<!--
]>
-->

```

Attributes for <snapshot>

The <snapshot> element is the root element and can contain one or more <node> elements. Although none of them are required (#IMPLIED), you may want to specify the following attributes for the <snapshot> element:

- ◆ `device` — The name or serial number of the asset.
- ◆ `model` — The name of the asset model.
- ◆ `time` — The date and time that the snapshot is created (the date and time are taken from the asset's operating system).
- ◆ `version` — The version number of the XML snapshot.

Attributes for <node>

One <node> element can contain one or more <node>, <leaf>, <dataitemblock>, <alarm>, <event>, and <dataitem> elements. The attributes for a <node> element are

- ◆ `name` — A name that you want to assign to the node. As shown in the DTD, this attribute is required (#REQUIRED).
- ◆ `status` — The condition of the node; the snapshot statuses, “ok,” “info,” “warning,” and “error” statuses are predefined in the configuration file for the Axeda Enterprise server (*DRMConfig.properties*, Snapshot Status Configuration section). You can define other statuses in this configuration file. Refer to the *Axeda® Enterprise Server Installation and Maintenance Guide*.
- ◆ `description` — A few words describing the node

Attributes for <leaf>

A <leaf> element is a set of name/value pairs. As shown in the DTD listing for this element, it has the following attributes:

- ◆ `name` — A name that you want to assign to the leaf. This attribute is required (#REQUIRED) and is the only one required for this element.
- ◆ `status` — The current status of the leaf; the snapshot statuses, “ok,” “info,” “warning,” and “error” statuses are predefined in the configuration file for the Axeda Enterprise server (*DRMConfig.properties*, Snapshot Status Configuration section). You can define other statuses in this configuration file. Refer to the *Axeda® Enterprise Server Installation and Maintenance Guide*.
- ◆ `units` — The unit of measure for the value of this leaf item.
- ◆ `min` — The minimum value for this leaf item. This element exists purely to inform a reader of the snapshot what the minimum value is; the information appears below the value pane of the Snapshot Viewer.
- ◆ `max` — The maximum value for this leaf item. This element exists purely to inform a reader of the snapshot what the maximum value is; the information appears below the value pane of the Snapshot Viewer.
- ◆ `url` — A Web link to additional information for the leaf. For example, the URL might point to a document that explains how to configure or tune the entity represented by the leaf. When displayed in the Snapshot Viewer, the URL is a live link.
- ◆ `time` — The date and time that this leaf was last created or updated.
- ◆ `type` — The data type for this leaf item; options for this attribute include `hex`, `octal`, `integer`, `float`, `text`, `html`, and `mime`.
- ◆ `desc` — A description of the leaf.

Attributes for <dataitemblock>

One <dataitemblock> element can contain one or more <dataitem> elements. As you can see in the DTD listing for this element, it has no additional attributes:

```
<!ELEMENT dataitemblock (dataitem+)>
```

Attributes for <alarm>

When the Axeda Agent parses a snapshot file for <alarm> elements, it updates the definition of the matching Alarm Style in the project with the values given for the attributes in the alarm element. When it encounters the <upload> sub-element, the Agent sends the alarm to the Axeda Enterprise server, using the information given in the <hint> sub-element of <upload>.

- ◆ `ack` — Boolean value, identifies whether the alarm is acknowledged (`true`) or unacknowledged (`false`) on the asset.
- ◆ `active` — Boolean value; whether the alarm is active (`true`) or inactive (`false`) on the asset
- ◆ `desc` — description of the alarm
- ◆ `name` — a unique identifier for the alarm. This name must match the name of an Alarm Style in the project that the Axeda Agent is running.
- ◆ `severity` — current severity value for the alarm. When the Axeda Agent encounters this attribute and a value in the snapshot file, it updates the alarm definition in the project with the value.
- ◆ `utc` — timestamp of the alarm item, in UTC format
- ◆ `upload` — identifies files to upload from the Agent to the Axeda Enterprise server in response to an alarm generated by the Script Manager while running the script. This element is the only way that the Axeda Enterprise server will be able to link the uploaded files to the specific script-based alarm that generated them. The upload attribute is a sub element of the alarm attribute; its attributes are as follows:
 - `hint` — type of file uploaded: `snapshot`, `dependency`, `log`, or `error`.

Below the `hint` sub-element, use a <file> sub element to identify all files of the same hint type (for example, all snapshots) to upload. The <file> element includes these three attributes:

- `name` — path and name of file(s) to upload; for example, `"/tmp/logs/*.xml"`
- `delete` — whether to delete the file after uploading it; valid values are `"y"` (yes, delete) or `"n"` (no, do not delete)
- `pathDeviceRelative` — for Axeda Gateway, whether the file path is relative to the Axeda Gateway installation path (where `xGate.exe` is installed); valid values are `"y"` (yes, relative) or `"n"` (no, not relative)

An example of an `upload` sub element follows:

```
<upload hint="snapshot">
  <file name="/tmp/logs/*.xml" delete="n" pathDeviceRelative="y">
</upload>
```

This example causes the Axeda Agent to upload all snapshot files in the directory, `/tmp/logs`. The Agent will *not* delete the files from the asset after uploading them. The path to the files is relative to the installation path of the Axeda Agent.

Attributes for <dataitem>

When the Axeda Agent parses a snapshot file for <dataitem> elements, it writes the value given for the data item to the corresponding data item in the project it is running. The string in the name attribute for the <dataitem> element must match the name given to a data item in the project for the value to be written.

- ◆ `name` — The unique identifier of the data item. For example, `name="LaunchStatus"`.
- ◆ `quality` — The quality of the data item; valid values are “good”, “bad”, “uncertain”. For example, `quality="good"`.
- ◆ `type` — The type of data in this data item; valid values are “analog”, “digital”, “string”. For example, `type="analog"`.
- ◆ `utc` — The timestamp of the data item, in UTC format. For example, `utc="2007-08-06T09:17:53.140-5:00"`

An example of a data item element follows:

```
<dataitem name="LaunchStatus" type="analog" quality="good"
utc="2007-08-06T10:22:34.140-5:00">4</dataitem>
```

When it finds this data item element, the Axeda Agent will look for a data item in the project with the name, “LaunchStatus”, and the type, “analog”. It will set the data item value to 4 and assign the UTC timestamp given in the attribute to the data item.

Attributes for <event>

When it parses a snapshot file for <event> elements, the Axeda Agent sends the event information to the Axeda Enterprise server. The attributes of the element provide the name of the event, a message concerning what occurred, its severity, and its timestamp, as follows:

- ◆ `message` — ASCII string describing the event
- ◆ `name` — name define in the project for the event
- ◆ `severity` — the level of severity of the event
- ◆ `utc` — timestamp of the event, in UTC format

Example Snapshot Files

This example snapshot will show information for two nodes, Registry and PerlScript, in the Axeda Snapshot Viewer.

```
<?xml version='1.0' encoding='UTF-8' ?>
<Snapshot version='1.0' model='ModelXYZ' device='asset123' >
  <node name="Registry">
    <node name="ActivePerl">
      <leaf name="CurrentVersion">805</leaf>
      <node name="805">
        <leaf name="">C:\Perl\</leaf>
        <node name="Help">
          <leaf name="">C:\Perl\html\index.html</leaf>
        </node>
      </node>
    </node>
  </node>
  <node name="PerlScript">
    <node name="1.0">
      <leaf name="NoCaseCompare">0x00000001</leaf>
      <leaf name="EnabledZones">0x00000010</leaf>
      <leaf name="EnableEventLogMsgs">0x00000000</leaf>
    </node>
  </node>
</Snapshot>
```

The following snapshot example shows how to format individual <dataitem> elements, an <alarm> element, and an <event> element:

```
<Snapshot>
  <dataitem name="ScriptAnalog" type="analog" quality="good" utc="2007-08-07T08:10:30+500">99</dataitem>
  <dataitem name="ScriptDigital" type="digital" quality="good" utc="2007-08-07T08:10:30+500">1</dataitem>
  <dataitem name="ScriptString" type="string" quality="good" utc="2007-08-07T08:10:30+500">Sun June 3 01:08:35 2003</dataitem>
  <event name="event1" message="event message" severity="111" utc="2007-08-07T08:10:30+500" />
  <alarm name="alarm1" desc="alarm description" severity="222" active="true" ack="false" />
</Snapshot>
```

The following snapshot example shows how you can format not only individual data items but also a block of data items:

```
<Snapshot>
  <dataitemblock>
    <dataitem name="LaunchStatus" type="analog" quality="good"
      utc="2007-08-10T09:17:48.140-05:00">1</dataitem>
    <dataitem name="LaunchStatusString" type="string" quality="good"
      utc="2007-08-10T09:17:48.140-05:00">canceled</dataitem>
    <dataitem name="DiskFlashRetCode" type="analog" quality="good"
      utc="2007-08-10T09:17:48.140-05:00">0</dataitem>
  </dataitemblock>
</Snapshot>
```

Using the Axeda® Snapshot Viewer

The Axeda Snapshot Viewer shows the content of snapshots (XML documents) created by scripts running on assets. These files may or may not be named *snapshot.xml*, but they will be defined with a type of *snapshot*, as shown in the list of uploaded snapshot files. The Axeda Snapshot Viewer appears when you select to view a snapshot that was uploaded to the Axeda Enterprise server. Uploaded snapshot files are listed in the Uploaded Files module on the Asset dashboard of the Axeda Service application, and in the View Uploaded Files From Assets list of the Axeda Software Management application. Figure 5-2 shows an example of the Snapshot Viewer after the nodes in the left navigation pane have been expanded. The rest of this section describes the areas of the Snapshot Viewer.

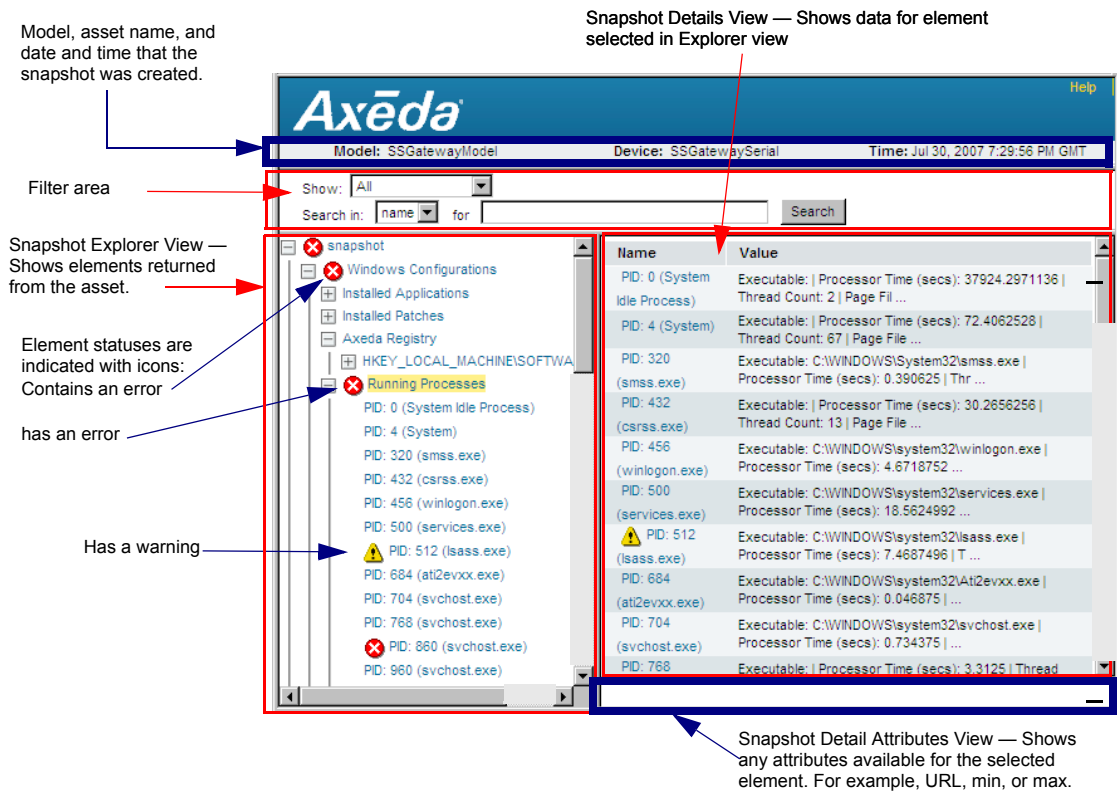


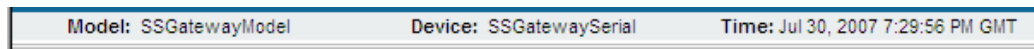
Figure 5-2. Viewing information in the Axeda Snapshot Viewer

Note: The Axeda Snapshot Viewer can show snapshot XML files only if their XML elements match the definitions in the snapshot DTD. If you attempt to open a file that is not created according the snapshot.dtd, an error message appears.

Information Shown in the Axeda Snapshot Viewer

At the top of the Axeda Snapshot Viewer window, the following information is displayed:

- ◆ **Model** — the model name associated with the asset; this name can be inserted into the script from the Axeda Agent running the script (if the script uses known substitution parameters for Axeda Agents)
- ◆ **Asset** — the name or serial number of the asset; this name can be inserted into the script from the Axeda Agent running the script (if the script uses known substitution parameters for Axeda Agents)
- ◆ **Time** — the date and time that the snapshot was created



Below the asset and time information are filter tools for selecting the contents of the file you want to see in the Axeda Snapshot Viewer.

- ◆ **Show** — an enumerated list, containing the following display options: **All**, **Not Success**, **Warnings & Errors**, and **Errors only**
- ◆ **Search in** — an enumerated list, containing the display options, **name** and **value**
- ◆ **for** — a text field in which you can type a specific string or numeric value to search for
- ◆ **Search** button — click this button to filter the snapshot details shown below



The actual contents of the file appear below the filter tools, in the snapshot Explorer view and snapshot Details view panes, shown in Figure 5-3. The information shown in the Details view depends on the filters applied as well as the current selection in the Explorer view.

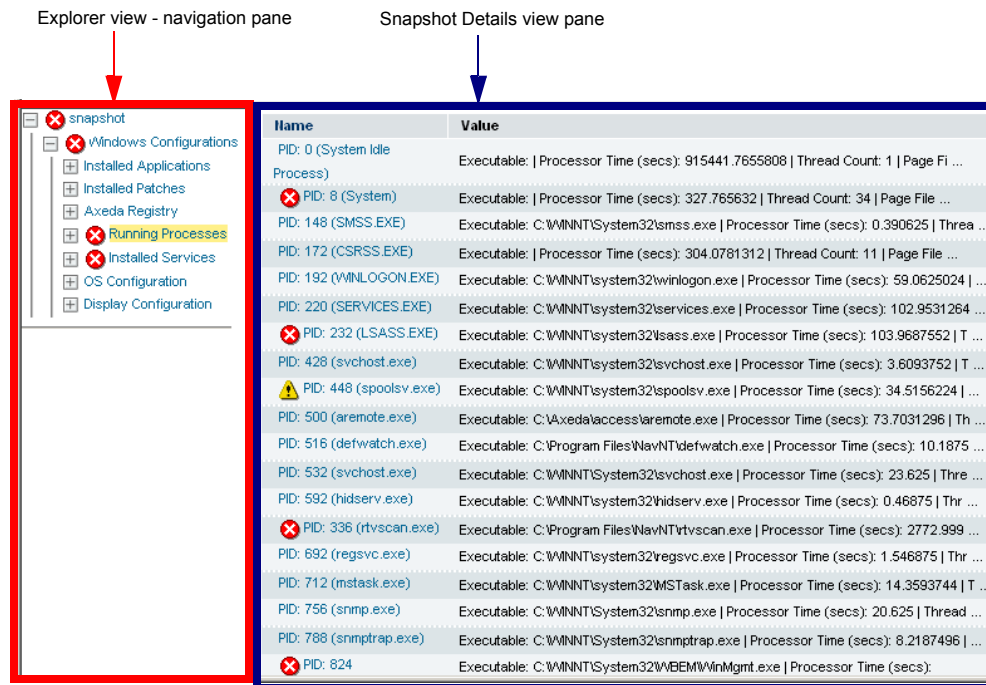


Figure 5-3. Snapshot Explorer view and Details view

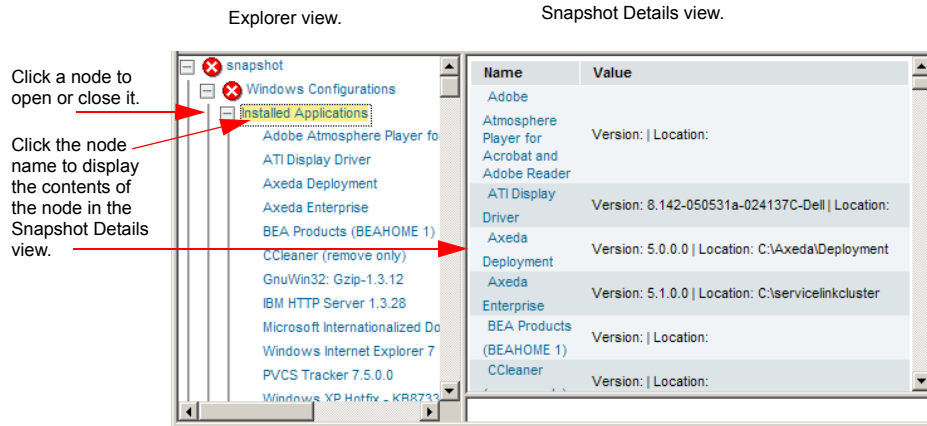
Below the snapshot Details view is another pane (snapshot Details Attributes view, shown in the following figure) that can show any attributes defined for the selected element, such as a URL (which, if clicked, will open in a new browser window), minimum or maximum values for the selected element, and type of units for the selected element.



Note: *The Axeda Snapshot Viewer window session will time out if you have not used it for a period of time (20 minutes is the default set in the Axeda Enterprise server properties). If an Axeda Snapshot Viewer session times out, you will need to log back in to the applications and then reselect any filters you want for viewing the snapshot file.*

Viewing the Contents of the Snapshot

The snapshot Explorer view (left side) shows nodes and their contents, as defined in the XML. For example, a node could be a directory or subdirectory on the asset, and the contents of that node could be files on the asset, configuration properties for the asset, or data items configured in the Agent project.



Clicking a node opens or closes it. When open, the contents of that node, either more nodes or actual content items, appear. If you click a node name, the contents of that node appear in the snapshot Details view (right side). The Details view shows the name of each item and its value. Below this pane is the snapshot Details Attributes view, which can show any attributes defined for the selected element, such as a URL, minimum or maximum values for selected element, and type of units for the selected element. The name for an element selected from the Snapshot Explorer view appears in the Snapshot Details view; any additional attributes defined for that element appear in the Snapshot Detail Attributes view.

Clicking a node item in either view shows any attributes defined for that item. If you select the node item from the Explorer view, the Details view shows the contents and attributes for the item. The attributes are names and values for the selected node item, such as the name and version of the asset's operating system, path and file name of the selected item, or programming code for a selected script node item.

Filtering and Searching the File Contents

You can filter the file contents by element status (**Show** list) to view only elements in error status or in error and warning status. Further, you can search the contents of the entire file by name or value (**Search in** list), and then search for specific strings or numbers in the name or value.

To view file elements by status

1. From the **Show** list, select the status for the type of information you want to view.
2. Click **Search** to view all elements that have the corresponding status

OR

Filter the displayed elements further by defining the type of elements to view, either values or names (see the next procedure, *To view file elements by name or value*).

Only elements or information with statuses of the same or greater significance than the selected status appear in the Snapshot Explorer view and Snapshot Details view.

For example, if you select **Warnings & Errors**, all elements defined with warning or error status are shown. If you select **Errors only**, only elements defined with an Error status are shown.

These statuses are defined by the script developer who determines what conditions of asset elements are considered errors or warnings and need to be flagged as such in the snapshot.

To view file elements by name or value

Define the status of the elements you want to view, if applicable (see the preceding procedure, *To view file elements by status*).

1. From the **Search in** list, select the type of elements you want to view, as follows:

- **name** — search for file elements by name
- **value** — search for file elements by value

2. To search for an exact or partial string or number, type it in the **for** text box.

You do not need wildcards for partial values or names; the Snapshot Viewer will return ALL values or names containing the string or number you enter here.

3. Click **Search**.

The Explorer and Details views update to show all elements that match the string with any portion of the specified search, either name or value.

Snapshot Parser Errors

The parser checks the PCDATA in a snapshot.xml file for entities and markup; it does NOT check CDATA (the values for attributes). If a script is retrieving information from the Registry of a machine, for example, it writes the values it finds as PCDATA for the elements that represent each piece of information. The <dataitem> and <leaf> elements provide the tags in which to write this information in the snapshot file. If the project does not define a data item for a piece of information, use the <leaf> element to store it in the XML file.

If the parser finds an error in the XML, the Snapshot Viewer cannot display the snapshot. However, the parser will return an error message that tells you the line number in the file where it encountered the error and explains the problem. For example:

```
line 41. parser expected entity reference
```

Strictly speaking, this message tells you that the parser encountered a reserved character that is used to signal the start of an entity reference; however, the characters that followed did not match any entity reference known to the parser. For example, the parser may have encountered an ampersand in line 41 that was not followed by a character entity reference. Continuing the example, the ampersand may have been followed by a space (as in the phrase, Search & Destroy). The ampersand is a reserved character in XML that tells a parser that an entity reference that it will have to translate into a character (or other entity) is coming next. For example, the character entity reference " tells the XML parser to write a double quotation mark. The following table lists the predefined entities in XML:

Table 0-1 Predefined Entities

This Entity Reference	Represents
<	less than sign (<), indicating the start of an XML tag
>	greater than sign (>), indicating the end of an XML tag
&	ampersand (&), indicating the start of an entity reference
"	double quotation mark (")
'	apostrophe (')

Important! Keep in mind that once the Axeda Agent sends a snapshot to the Enterprise server, you cannot edit the snapshot XML file manually to correct an error detected by the XML parser. To avoid parser errors caused by content that contains characters that are reserved in XML, be sure that your script checks the content it has captured for these characters. Refer to HTML or XML references for lists of reserved characters.

Comparing Snapshots

In the Uploaded Files module of an Asset dashboard (Axeda Service application) is a link, **File Comparison**, which allows you to compare two snapshots (and two TXT files).

To compare two snapshots

1. Display the Asset dashboard for the asset whose snapshots you want to compare.
2. In the Uploaded Files module, click the **File Comparison** link, as shown in Figure 5-4.



Figure 5-4. Uploaded Files module on Asset dashboard

In a separate browser window, you are prompted to select the files to compare (Figure 5-5):

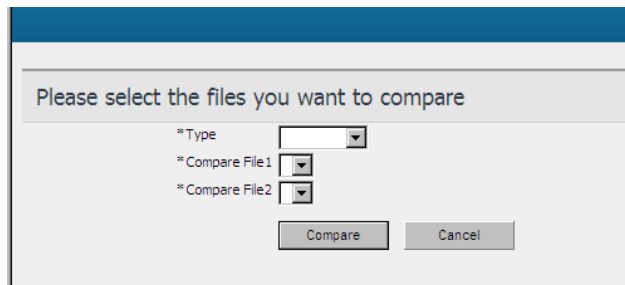


Figure 5-5. Select the files to compare

3. From the **Type** list, select the type of file, **Snapshot**. (The other option is **Text** files, for comparing two TXT files.) If you select Snapshot, the system displays only files of that type in the File 1 and File 2 lists. If you select Text, the system displays all files that have been uploaded for the asset. However, only files that are in a TXT format can be compared when you select Text.
4. From the **File 1** list, select a snapshot.XML file that you want to compare.

- From the **File 2** list, select the snapshot XML file to which you want to compare File 1. This list is the same as **File 1**, so be careful to select a different file name.
- Click **Compare**. The results appear in the same browser window as the prompt. Figure 5-6 shows an example of the results of comparing two snapshots:

The screenshot shows the Axeda web interface with the following details:

- Header:** Axeda logo, Help link, and buttons for Export, Back, and Close.
- File Information:**
 - File 1: ./win32_configuration_snapshot.xml_07-31-2007_23:56:59
 - File 2: ./win32_configuration_snapshot.xml_07-31-2007_22:53:12
- Legend:** Added (green), Modified (yellow), Removed (orange).
- Table:**

Parameter	Type	File 1	File 2
PID: 3412 (wmpirvse.exe)	Added		Executable: Processor Time (secs): 0.1875 Thread Count: 7 Page File Size: 2220032 Page Faults: 1640 Working Set Size: 5971968
PID: 2804 (xGate.exe)	Added		Executable: c:\Axeda\Gateway\Gate.exe Processor Time (secs): 0.640625 Thread Count: 14 Page File Size: 3395584 Page Faults: 1068 Working Set Size: 2015232
PID: 1964 (wmpirvse.exe)	Added		Executable: Processor Time (secs): 0.453125 Thread Count: 6 Page File Size: 1933312 Page Faults: 1257 Working Set Size: 4960256
PID: 4060 (cscrip.exe)	Added		Executable: C:\WINDOWS\system32\cscrip.exe Processor Time (secs): 0.4375 Thread Count: 6 Page File Size: 4308992 Page Faults: 1740 Working Set Size: 6828032
PID: 1780 (xGate.exe)	Removed	Executable: c:\Axeda\Gateway\Gate.exe Processor Time (secs): 0.4375 Thread Count: 14 Page File Size: 3350528 Page Faults: 1897 Working Set Size: 1916928	
PID: 3656 (wmpirvse.exe)	Removed	Executable: Processor Time (secs): 0.125 Thread Count: 7 Page File Size: 2220032 Page Faults: 1640 Working Set Size: 5971968	
PID: 3380 (wmpirvse.exe)	Removed	Executable: Processor Time (secs): 0.390625 Thread Count: 6 Page File Size: 1933312 Page Faults: 1256 Working Set Size: 4956160	
PID: 2272 (cscrip.exe)	Removed	Executable: C:\WINDOWS\system32\cscrip.exe Processor Time (secs): 0.40625 Thread Count: 6 Page File Size: 4308992 Page Faults: 1735 Working Set Size: 6807552	
PID: 3676 (cidaemon.exe)	Modified	Executable: C:\WINDOWS\system32\cidaemon.exe Processor Time (secs): 194.6406272 Thread Count: 4 Page File Size: 6385664 Page Faults: 782342 Working Set Size: 462848	Executable: C:\WINDOWS\system32\cidaemon.exe Processor Time (secs): 192.54668 Thread Count: 4 Page File Size: 6397952 Page Faults: 775073 Working Set Size: 603808
PID: 704 (svchost.exe)	Modified	Executable: C:\WINDOWS\system32\svchost.exe Processor Time (secs): 2.125 Thread Count: 22 Page File Size: 3072000 Page Faults: 5256 Working Set Size: 5701632	Executable: C:\WINDOWS\system32\svchost.exe Processor Time (secs): 2.048875 Thread Count: 20 Page File Size: 3031040 Page Faults: 5109 Working Set Size: 5681152

Figure 5-6. Comparison results

If either of the selected files is empty, the system returns an error. If you select the same file name for **File 1** and **File 2**, the system also returns an error. In either event, select a different file, and click **Compare** again.

The results include the following information:

- File 1 and File 2 — the paths and names of the snapshot files you selected
- Added, Removed, and Modified sections — the names and values of attributes that were added, removed, or modified from one snapshot to the other. Different colors distinguish the records. By default, the settings are: green for Added records, yellow for Modified records, and orange for Deleted records.

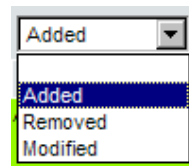
A snapshot comparison can span multiple pages. To help you sort through the information for the snapshots, filters are available for each of the column headings.

7. For snapshot files, you can

- Sort information by Parameter, Type (Added, Removed, or Modified), File 1, or File 2, by clicking the column heading.
- Search for information, using the Simple filters:

a) In the text box above a column heading, type all or part of a **Parameter** name or **File** name. You can use a wildcard if you do not know the exact name to type.

b) To view only a certain type of information, select one from the **Type** list:



c) Click **Filter**. When it refreshes, the page displays only the items corresponding to the filters you typed. (Refer to Figure 5-7 on page 5-22 for an example.)

The following figure shows an example of filtering by the **Added** Type:

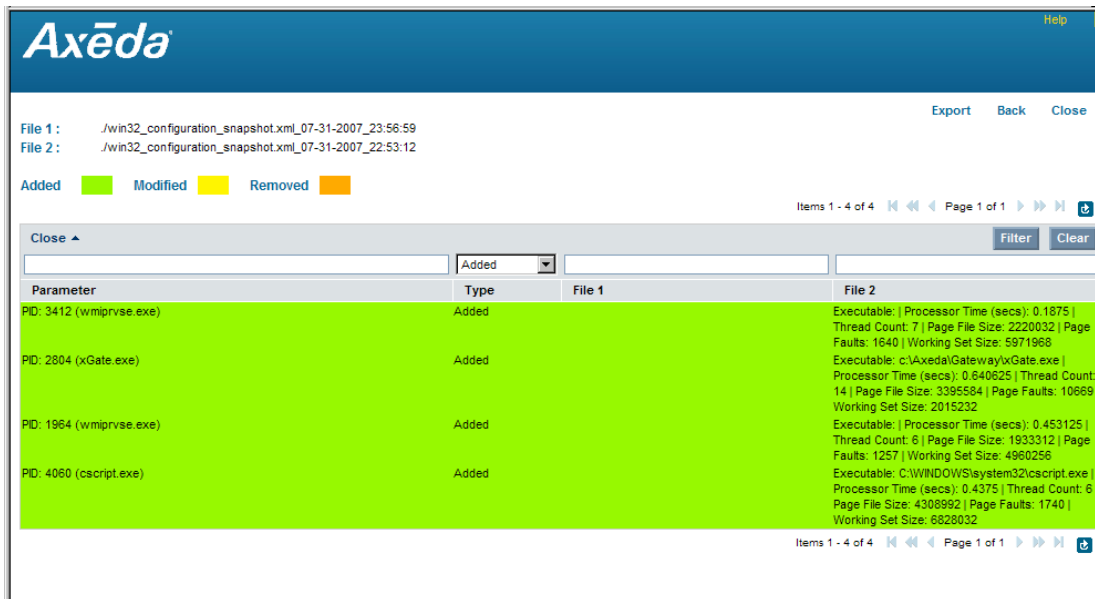
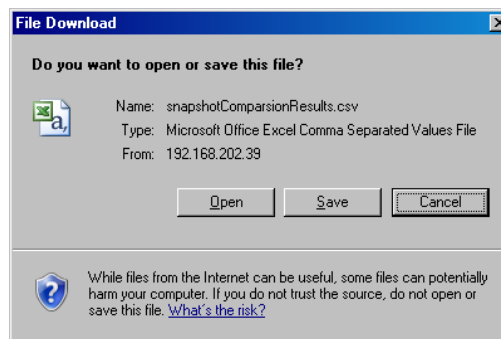


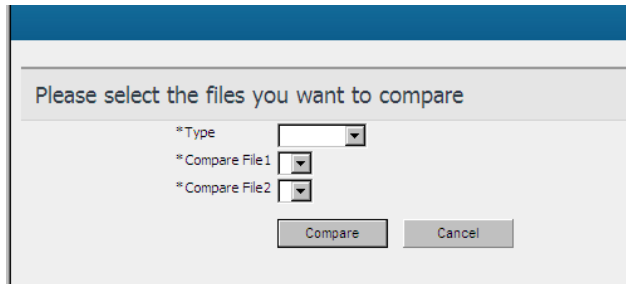
Figure 5-7. Filtering comparison results

- Use different filtering criteria: click **Clear**, and then select your new filtering criteria and click **Filter**.
- Export comparison results to a comma separated file (*.csv) by clicking the **Export** link. The File Download dialog box for your system appears:



Click **Save** to save the file to your local hard drive.

- ♦ Run another comparison by clicking the **Select Files** link to display the initial File Comparison screen:



The screenshot shows a dialog box titled "Please select the files you want to compare". It contains three dropdown menus: "*Type", "*Compare File1", and "*Compare File2". Below the dropdowns are two buttons: "Compare" and "Cancel".

Select the Type of files, then select the files to compare, and click **Compare**.

- ♦ Exit the utility and close the window by clicking **Close**.



Chapter 6 Managing Axeda Scripts from ServiceLink Applications

Axeda® ServiceLink™ Applications contain tools for managing scripts that are run by Axeda® Gateway and Axeda® Connector. These tools are available in the Axeda® Service, Axeda® Configuration, and Axeda® Software Management applications. Using Axeda Configuration, you can create script actions and rules that will run scripts in response to a defined condition. From the Asset dashboard in Axeda Service, you can run a registered scripts on the asset whenever necessary. The Axeda Software Management application enables you to create packages that, when downloaded to assets, run scripts either on a schedule or on demand.

Managing Scripts from ServiceLink Applications explains how sequence files and scripts become available, where they run, where to view the data that scripts collect, and where and how to manage sequence files and scripts.

Creating Dynamic Timers explains how to use the Axeda Service application to create dynamic timers so that you can run scripts based on a schedule.

Script and Timer Statuses and Results explains where and how to view the status and results of running sequence files. It also defines all the possible statuses.

Script Tools in the Axeda Configuration Application explains how to use the Axeda Configuration application to view and manage scripts, including registering scripts and assigning scripts to assets.

Packages to Run Scripts explains how to use a package (Axeda Software Management application) to download and run Axeda scripts on assets.

Script Tools in the Axeda Service Application explains how to use the Axeda Service application to manage Axeda scripts.

Managing Scripts from ServiceLink Applications

Using the Axeda Configuration and Axeda Service applications, you can manage scripts for your Axeda Platform system. You can import new scripts, run existing scripts, create dynamic timers for running scripts on a schedule, and publish imported scripts to make them available to other assets. Using the Axeda Software Management application, you can create a package that downloads a script to assets, where the Axeda Agents run the script.

Note: *Scripts included in the Axeda Agent's project configuration are registered automatically for that Axeda Agent and cannot be published for use by other assets. The script exists solely on that asset and can be run only on that asset.*

Publishing Scripts

Publishing a script makes it available for registration and use on other assets of the same model, or for configuration as Run Script actions. Note that you cannot un-publish a script, except to delete it from the related assets. In addition, you can delete scripts imported to the Enterprise server only, not scripts installed directly on the assets.

Registering Scripts

Registering an Axeda script on an asset is the process of downloading all files for the script (sequence and script files) from the Axeda Enterprise server to that asset. An Axeda Agent cannot run a new script until it has been registered on that asset, in one of the following ways:

- ◆ Downloaded in a package created using the Axeda Software Management application.
- ◆ Selected directly for registration from the **Manage the registrations for script** `<script_name>` page in the Axeda Configuration application.
- ◆ Automatically registered when imported using the **Script manager wizard** from the Axeda Configuration application or from the Scripts page for the asset in the Axeda Service application.

Scripts can be installed directly on the asset, as part of installing the Axeda Agent. These scripts are automatically registered when the Axeda Agent registers its scripts with the Axeda Enterprise server.

When an asset first comes online to the Axeda Enterprise server, it *registers* its scripts. The server overwrites its script information for the asset with the newly registered information. In this manner, script registration ensures that the server reflects accurate script information for the related asset. The asset registers its scripts when it comes online, and any time an applications user manually requests that asset to register or unregister scripts from the Scripts page of the Axeda Service application or as configured in a package.

Running Scripts

An Axeda Agent will start a script on its asset, as directed by the configuration of its project (a logic schema, for example) or as directed by the Axeda Enterprise server. An Axeda Agent can run only the scripts it has, that is, scripts that were either included in the Axeda Agent's project configuration or sent to the Axeda Agent from the Axeda Enterprise server. Any script an Axeda Agent needs to run must be actively registered on that Axeda Agent. This rule applies even to Axeda Agent- installed scripts that may have been unregistered on an Axeda Agent. Unregistered scripts need to be re-registered with that Axeda Agent if it is to run those scripts.

The Axeda Agent updates the Axeda Enterprise server with its list of scripts as part of asset and script registration. This automatic update ensures that the Axeda Enterprise server accurately reflects that information to the scripting user.

From the applications, there are several ways to request the Axeda Agent to run a script:

- ◆ From the Axeda Configuration application or Axeda Service application, select to run a script already registered on an asset.
- ◆ From the Axeda Software Management application, deploy a package configured to download and run scripts on the asset.
- ◆ From the Axeda Configuration application, create a Run Script action for a published script available to an asset.

Note: *If the script is not registered on that asset, the action fails on the asset and error information is sent to the Audit log (Axeda® Administration application) and the asset's own error log (Asset dashboard in Axeda Service application).*

If an error occurs while the script is running, the Axeda Agent reports the error to the Axeda Enterprise server. You can see this information in the Axeda Service application, which shows the line number and command on which the script generated an error. The Axeda Service application reports all script status information from the Axeda Enterprise server and from the asset. You can use this information to determine if a script is awaiting registration, running on the asset, canceled from the Axeda Enterprise server, generated an error (and the line that caused the error), and more.

Running Scripts Locally

In addition to starting scripts remotely, Axeda Platform supports running any script command locally (at the asset) or through a Telnet connection to the asset. The output of the commands can be parsed to create XML. Examples of additional uses of scripts include running an asset utility that generates a report and uploads the report to the Axeda Enterprise server, or performing maintenance that cleans up processes that are no longer running and temporary files that are no longer needed, based on a condition being true.

Running Scripts on Assets Managed by Axeda® Gateway

Scripts can be run on an asset that is running Axeda Gateway, and on the assets managed by (and connected to) the gateway asset.

Stopping Scripts

From the Axeda Configuration or Axeda Service applications, you can send a “Stop” or “Cancel” command that instructs the Axeda Agent to stop a script that it is currently running on an asset. When the Agent next contacts the server, it receives this command and stops the running script.

Creating Dynamic Timers

You can assign scripts to *dynamic timers*; that is, timers created in the Axeda Service application for the purpose of running scripts based on a schedule. When a dynamic timer is registered with an asset, the Axeda Agent runs the configured scripts based on the related timer’s schedule. (You can configure Axeda Agent projects with their own timers: dynamic timers are executed in the same manner as the project timers and do not conflict with project timers on an asset.)

You can configure dynamic timers to initiate almost any asset operation that can be done locally. Examples include running an asset calibration session, automating a set of system administration steps, or capturing and parsing complex data structures or logs for upload to customer service personnel.

Dynamic timers can be scheduled to execute at regular intervals (relative or absolute timer), or initiated on demand from the Timers pages on the Asset dashboard of the Axeda Service application. Figure 6-1 shows two dynamic timers created for asset snapshot_sn1: *Before shutdown* and *On Startup*. These timers run the associated scripts each day, one at 6:00 PM and the other 8:00 AM, respectively.

Service : snapshot_sn1

Jump to: Timers

Status	Name	Description	Next run	Creator	Actions
Waiting to register	Before Shutdown	Every day at 6:00:00 PM EST	Aug 15, 2007 6:00:00 PM EST	devuser	Send to other devices Stop
Waiting to register	On Startup	Every day at 8:00:00 AM EST	Aug 16, 2007 8:00:00 AM EST	devuser	Send to other devices Stop

Scripts

- UploadSnapshot Run
- ParseSnapshot Run
- ParseFred Run

Available Scripts

Agent Triggers

- OnTimerParse1
- OnTimerUploadSnap

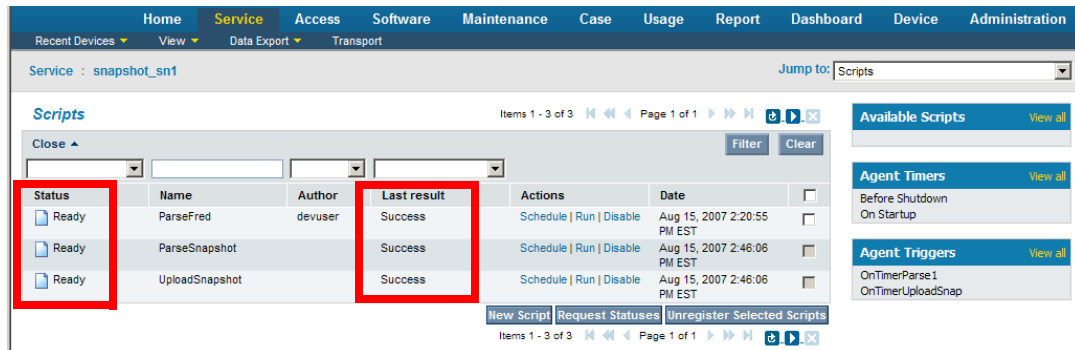
Figure 6-1. Example of two dynamic timers created for asset snapshot_sn1

When scripts are assigned to a dynamic timer, a *dynamic script* results. A dynamic script is comprised of a timer configured to execute one or more scripts. From the Axeda Service application you create the timers, assign their scripts, and then register them with the related assets.

Script and Timer Statuses and Results

The Axeda Agent sends the status of its scripts and timers to the Axeda Enterprise server when a script starts or exits (successfully or not). This status information is shown in the Scripts page of the Axeda Service application. From this application you can also request an update of the script status for an asset; the Axeda Agent will send the status when it next contacts the server.

The Axeda Service application displays two different status values for each script associated with an asset. The first is the current state of the script (Ready, Not Running, Waiting to run, and so forth), and the second is most recent result of running the script. The latter status corresponds to the date and time displayed for that script result. Figure 6-2 highlights these two statuses.



Status	Name	Author	Last result	Actions	Date
Ready	ParseFred	devuser	Success	Schedule Run Disable	Aug 15, 2007 2:20:55 PM EST
Ready	ParseSnapshot		Success	Schedule Run Disable	Aug 15, 2007 2:46:06 PM EST
Ready	UploadSnapshot		Success	Schedule Run Disable	Aug 15, 2007 2:46:06 PM EST

Figure 6-2. Viewing script statuses, as reported from the server and asset.

Agent Script Status

When it receives a command to run a script, the asset starts the script and sends status information back to the server; this status includes the current line number and command being run (line of the Sequence file). If the script errors, it sends the error as an event to the server and stops running. For example, you can configure the script error event to trigger a notification either to an operator at that asset or to the service technician for that asset.

Last Result Values

The possible values for the **Last Result** status are:

- ◆ **Success**
- ◆ **Error/Failed** (including the line number in the script that was processing when the error occurred). For example, “*Timed out at line: 6*” or “*Failure at line 11: \$(parse diskUsageIntermediate.xml)*”.

- ◆ **Canceled** (including the line number in the script that was running when the script was canceled)
- ◆ **Interrupted** (meaning that while the script was running, the asset was restarted)
- ◆ **None** (meaning that the script has never been run)

Script Status Messages

The second type of status is the script's status, based on either information received from the Axeda Agent or activity on the Axeda Enterprise server. This information appears in the Status column of the Script table.

Possible messages from the Axeda Agent include

- ◆ **Running**, and the line number and command currently running.
- ◆ **Not Running**
- ◆ **Error registering** or **Error unregistering**, on the asset.

Possible messages from the Axeda Enterprise server include

- ◆ **Ready** – not currently running; enabled and ready to be run.
- ◆ **Disabled** – will not run on the asset for any manual or automatic configurations (timers, triggers, and so forth) until it is again enabled and returned to the Ready status.
- ◆ **Waiting to run** or **Waiting to cancel** – the script was selected to be run/canceled, but has not yet been run/canceled on the asset because, for example, the asset has not yet contacted the server and received that message.
- ◆ **Waiting to disable** or **Waiting to enable** - the script was selected to be disabled/enabled, but is not yet disabled/enabled because, for example, the asset has not yet contacted the server and received that message.
- ◆ **Waiting to register** or **Waiting to unregister** - the script was selected to be unregistered/registered on the asset, but has not yet been unregistered/registered because, for example, the asset has not yet contacted the server and received that message.

Script Tools in the Axeda Configuration Application

Using the Axeda Configuration application, you can:

- ◆ View all scripts available to you (per your permissions) and assign them to assets
- ◆ Assign and register scripts to selected assets
- ◆ Manage script registrations for assets
- ◆ Run, stop, and cancel scripts

The rest of this section explains how to perform these tasks. To learn how to use actions or packages to run scripts, continue to the sections, "[Actions to Run Scripts](#)" on page 7-2 and "[Packages to Run Scripts](#)" on page 6-12.

Viewing and Managing Scripts

The View and manage scripts page shows all scripts configured in the database or available on the registered assets, for the selected search criteria. This tool is available by selecting the **Scripts** option on the **View** menu. Figure 6-3 shows an example of the View and manage scripts page.

Name	Description	Model	Author	Actions	
Diagnostic Snapshot		PC	n/a	Registrations	<input type="checkbox"/>
FredSnapshot		FireFox_Model1	n/a	Registrations	<input type="checkbox"/>
ParseFred		Snapshot_mm	devuser	Registrations Publish Copy	<input type="checkbox"/>
ParseSnapshot		Snapshot_mm	n/a	Registrations	<input type="checkbox"/>
UploadSnapshot		Snapshot_mm	n/a	Registrations	<input type="checkbox"/>

Figure 6-3. View and manage scripts page

The Scripts table shows the following information for each script:

- ◆ **Name** - of the related script. Server-based scripts are shown in linkable blue text; asset-based scripts are shown in black text
- ◆ **Description** – of the script
- ◆ **Model** – to which this script is assigned
- ◆ **Author** – name of the user who created this script

From the View and manage scripts page, you can:

- ◆ Select to filter the list of scripts based on search criteria; for example, find all scripts created by a selected user (author) or for a selected model.
- ◆ View more information for scripts created in the server. You cannot view and edit information for scripts created in the Axeda Agent's project. Server-based scripts are shown in linkable blue text; asset-based scripts are shown in black text.
- ◆ View and manage the registrations for scripts; for example, you can specify the assets on which you want to register or re-register a script or, for each asset, you can select to run or cancel scripts, or enable or disable the scripts' operations on an asset.
- ◆ Publish a script for use in other assets of this model; this option is available in this page only if the script is not yet published.
- ◆ Copy server-created scripts for use with other models
- ◆ Redeploy or re-register a script to all assets; this option is available only for server-based scripts that were already registered on one or more assets.
- ◆ Delete scripts from the server; you cannot delete asset-based scripts.

Tip To unregister asset-based scripts, download a software management package that contains an Axeda Agent instruction to unregister the specified scripts.

Managing the Registrations for a Script

When you click the [Registrations](#) link (under **Actions**) for a script on the View and manage scripts page, the Manage the registrations of script `<script_name>` page appears, as shown in Figure 6-4.

Device	Status	Last Result	Result Date	Actions
snapshot_sn1	Ready	Success	Aug 15, 2007 2:20:55 PM EST	Run Disable

Figure 6-4. Managing script registrations for all assets

This page shows all assets on which a script is currently registered. If no assets are shown here, you may first need to publish the script, which you can do from the **View and manage scripts** page. The **Registered Assets** table shows the following information for all assets on which the script is already registered or is in the process of registering:

- ◆ **Asset** – name of the asset on which the script is registered
- ◆ **Status** – current status for this script for this asset

- ◆ **Last Result** – result status for the last time this script ran on this asset; blank if the script has not yet run on the asset
- ◆ **Result Date** – date and time related to the last result received from the asset

From this page, you can:

- ◆ Run the script on selected assets
- ◆ Register or unregister the related script on selected assets. The script will be deleted on the assets (at the registered paths), and the Axeda Agents on the assets will run any unregister commands defined in the script.
- ◆ Cancel a running script
- ◆ Disable or enable the script on selected assets

For each action or operation you perform, the Status field is updated with the latest information. For example, if you select to run a script, the Status column shows “Waiting to run”. The next time it contacts the server, the asset receives this action and runs the script.

Selecting Assets for a Script

After adding a new script through the Axeda Configuration application, you need to select assets on which to register (and run) the script.

To select assets on which you want to register a script

Note: *Before you can register a script on other assets, you must first Publish the script (select Publish in the Actions column on the View and manage scripts page).*

1. From the **View and manage scripts** page, click **Registrations** in the Actions column for the script that you want to register. The Manage the registrations of script <script_name> page appears.
2. To start the wizard, click **Register More Assets**. This button appears only for scripts that can be registered on additional assets. If you do not see this button, then you need to return to the View and manage scripts page and select **Publish** in the **Actions** column.

The **Choose assets for script** wizard starts, showing the assets on which the related script is registered and able to be run, or the assets on which the script was unregistered and cannot be run.

The wizard information for a script is presented in three pages, as follows:

1. Choose assets for script `<script_name>` – select the assets on which this script will be registered
2. Choose assets for script `<script_name>` – select the assets on which this script will be re-registered
3. Confirmation for script `<script_name>` – select to register or re-register the script from the selected assets, or change the script registration selections first, or cancel the script registration selections altogether.

Warning! *When moving backward and forward through the wizard, use the wizard's **Back** and **Next** buttons. Do NOT use your browser's Back and Forward tools.*

Step 1: Choose assets for script wizard – Unregistered Assets

The **Available unregistered assets** table lists all assets for this model on which this script is not yet registered. Once you select assets from the **Available** table and click an **Add** button, the **Selected unregistered assets** table lists the assets on which this script will be registered when you finish this wizard.

Step 2: Choose assets for script wizard - Registered assets

The **Available registered assets** table lists all assets for this model on which this script is already registered. Once you select assets from the **Available** table and click an **Add** button, the **Selected registered assets** table lists the assets on which this script will be re-registered when you finish this wizard.

Step 3: Choose assets for script wizard – Confirmation

This page lists all the asset script registrations or re-registrations selected in this wizard. When you confirm the settings as shown and click **Finish**, the server queues the registration settings. Then, the next time that the asset(s) contact the server, the asset(s) will register or unregister the script, based on the actions sent from the server.

Packages to Run Scripts

Another way to run a script on an asset is using a package that is configured to download and run that script on selected assets. From the Software Management application, you can create a new package for a selected model by selecting **Package** on the **New** menu. After selecting the model for the package, you need to specify the instructions that define the operations or actions contained in this package: select a Run Script instruction and then select the script to run.

When this package is deployed, the Axeda Agent runs all commands specified in the Sequence file's [RUN] section, or the entire script if [RUN] is not defined. If there are multiple scripts to run, you can specify a wait period between the executions.

Script Tools in the Axeda Service Application

Using the Axeda Service application, you can

- Assign and register scripts to a selected asset
- View scripts available for a model (and per your permissions)
- Create and manage dynamic timers
- Run, stop, cancel, register, and unregister scripts for a selected asset
- View the Agent triggers configured to run scripts

In addition to the tools that are similar to those of the Axeda ConfigurationAxeda Configuration application, the Axeda Service application provides tools for managing and controlling scripts in runtime for a selected asset. The Scripts module on the Asset dashboard of the Axeda Service application (see Figure 6-5) shows scripts available to be run on the asset.

Organizations : Default Customer : Default Location : 100xk Jump to: Dashboard

Location [Edit](#)

Customer: Default Customer
Location: Default Location

Contacts [View all](#) | [Manage](#)

Notes [View all](#) | [Add](#)

Charts

Displays

Rules [View all](#)

Properties [Edit](#)

100xk

Serial number: 100xk
Model: EP_VinMon2
Status: ● Good

Registration: 10/20/06 1:31 PM
Last contact: 10/20/06 1:55 PM (2 hours 43 minutes ago)
Ping rate: 20 seconds
Time zone: Greenwich Mean Time

Recent Actions [View all](#)

10/20/06 1:54 PM	Complete	Package Deployed [file = .\win32_configuration_snapshot.xml, device relative path = false, delete after upload = false]
10/20/06 1:51 PM	Complete	Package Deployed [file = .\win32_configuration_snapshot.xml, device relative path = false, delete after upload = false]
10/20/06 1:34 PM	Complete	Package Deployed [file = .\win32_configuration_snapshot.xml, device relative path = false, delete after upload = false]
10/20/06 1:31 PM	Complete	Package Deployed [file = .\win32_configuration_snapshot.xml, device relative path = false, delete after upload = false]

Alarms [Current Alarms](#) | [Historical Alarms](#) | [Acknowledge All](#)

10/20/06 1:55 PM	High CPU Condition Value	(Acknowledge)
------------------	--------------------------	---------------

Uploaded Files [View all](#)

10/20/06 1:54 PM	./win32_configuration_snapshot.xml
------------------	------------------------------------

Audit Log [View all](#)

10/20/06 1:54 PM	Successfully uploaded file(s): ./win32_configuration_snapshot.xml
10/20/06 1:54 PM	Agent command (id=drmserver#30, Deployed Package): file = .\win32_configuration_snapshot.xml, device relative path = false, delete after upload = false.
10/20/06 1:54 PM	Device 100xk restarted
10/20/06 1:51 PM	Successfully uploaded file(s): ./win32_configuration_snapshot.xml
10/20/06 1:51 PM	Agent command (id=drmserver#28, Deployed Package): file = .\win32_configuration_snapshot.xml, device relative path = false, delete after upload = false.

Data [Current Data](#) | [Historical Data](#)

10/20/06 1:54 PM	CPU Utilization: 99
10/20/06 1:54 PM	Disk Description C: N/A
10/20/06 1:54 PM	Disk Percent Free C: 0
10/20/06 1:54 PM	Disk Space Available C: 33344736
10/20/06 1:54 PM	Disk Space Total C: N/A

Tasks

[Edit this device](#)

[Add this device to watch list](#)

[E-mail device link](#)

Actions [View all](#)

Scripts [View all](#)

Windows Applications Snapshot	Run
Windows Configuration Snapshot	Run
Windows Display Settings Snapshot	Run
Windows Patches Snapshot	Run
Windows Processes Snapshot	Run

Remote Sessions [View all](#) | [Download All Clients](#)

[Remote Browser](#)

[Desktop](#)

Deployed Packages [View all](#)

Maintenance Windows [View all](#)

Figure 6-5. Asset dashboard, Scripts module

You can select a script to run from the Asset dashboard. The next time it contacts the server, the asset receives this command and runs the script. If you do not see a script that you want to run, click **View all** in the Scripts module. The Scripts page appears, showing all registered scripts and available scripts, timers, and triggers available or configured for the asset. Figure 6-6 shows an example of this page.

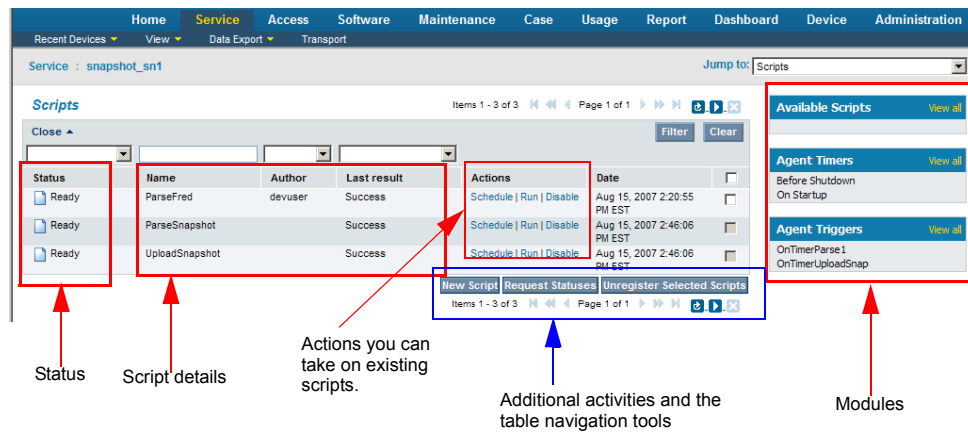


Figure 6-6. Viewing all scripts available to or registered with an asset

From the Scripts page, you can:

- ◆ View the details of a registered script
- ◆ Request the statuses of a registered script
- ◆ Create new scripts
- ◆ Schedule the scripts for timer-based execution
- ◆ Re-register or un-register a script from the asset
- ◆ Run a selected script
- ◆ Disable a script
- ◆ Stop a running script on the asset
- ◆ View timers configured for this asset; you can then select to create new timers and manage timers for the asset. Note that any timers listed are NOT those that are configured in the project that the Axeda Agent is running.
- ◆ View triggers for the asset; these are the logic schemas (configured in the Axeda Agent project) that trigger Run Script actions.

(Refer to “Script Tools in the Axeda Configuration Application” on page 6-8 for information on similar script functionality available in the Axeda Configuration application.)

On the right side of this page are three modules:

- ◆ The **Available Scripts** module shows scripts that are registered with the asset but not yet published to that asset. Select **View all** to see all scripts for this asset. From this page you can deploy selected scripts to the asset.
- ◆ The **Agent Timers** module shows any timers created using this application for the asset. Select **View all** to see all timers for this asset. From this page, you can select to create new timers or delete timers for this asset, add scripts to a selected timer, or send a timer to other assets of the model.
- ◆ The **Agent Triggers** module shows triggers configured for the Axeda Agent on the asset. Select **View all** to see all triggers for this asset. The Triggers table shows Run Script actions configured in the Logic Schema of an Axeda Agent project running on the asset. This table is blank if no Run Script actions are configured in the project.



Chapter 7 Axeda Builder Tools for Scripts

Axeda Builder also provides script tools. You can refer to the online help for Axeda Builder when setting up your project. This chapter explains how to access the Builder script tools in the following sections:

Actions to Run Scripts explains how to use the Run Script action to run Axeda scripts on assets where the scripts are registered.

Script Tools in Axeda Builder explains how to use the Axeda Builder application to create a sequence file and set up a project to run scripts.

For detailed procedures, refer to the online help for the Axeda Configuration, Axeda Service, Axeda Software Management, and Axeda Builder applications.

Actions to Run Scripts

Run Script actions enable the Axeda Enterprise server to run a script on an asset. Only published scripts and those scripts available to a selected model are available for configuration as asset actions.

When the rule associated with the Run Script action evaluates to true, the Axeda Enterprise server sends a command to the Axeda Agent on the related asset(s) to start this script. This script must already exist on the asset. (This action does not first deploy the script to the asset; it assumes the script exists on the asset already.)

To check whether a script is available and registered on an asset before attempting to start the script on the asset, look for the script in the Scripts module or Scripts page from the Asset dashboard of the Axeda Service application. If the script does not already exist on the asset, you can use the tools in the Axeda Service application to deploy the script to the asset. You can also download the script to the asset using a package created using the Axeda Software Management application.

Tips

- *In the Rules module on the Asset dashboard of the Axeda Service application, you can view and enable or disable the rules configured for a specific asset.*
- *The Audit log module on the Asset dashboard of the Axeda Service application shows a log of activity for the selected asset, including registration, actions run, alarms acknowledged, statuses, usage value setting, and more. Also, if you have privileges to the Axeda Administration application, you can search the Audit log for asset, Axeda Enterprise server, and user activity.*

You can create new Run Script actions using the Action wizard, available by selecting **Actions** on the **New** menu in the Axeda Configuration application. Run Script actions perform operations on assets and thus are categorized as “Asset update actions” in the Action wizard. When creating a Run Script action, you select the model for the script and the script to run for the action.

Notes: *All scripts available to the selected model appear, whether or not they actually exist on an asset of the selected model. Before you need to run this script, make sure it already exists and is registered on the asset. You can do this by viewing the list of scripts registered on an asset in the Scripts module on the Asset dashboard in the Axeda Service application or on the Scripts pages.*

If you do not see a script, you may need to create it first in the Scripts module on the Asset dashboard of the Axeda Service application or as a package.

The Run Script action appears in the Actions table. You can assign it for running on assets.

Script Tools in Axeda Builder

You can use Axeda Builder to configure script definitions and logic schemas that run scripts on the Axeda Agent assets. To trigger Run Script actions in logic schemas, you may also want to configure timers in Axeda Builder. In addition, if the script will parse a snapshot file that contains values for data items, then you need to use Axeda Builder to create a data logger that will send any changed data item values to the Axeda Enterprise server.

Create Scripts

Using Axeda Builder, you create Axeda scripts (sequence files and related scripts) from the Script Definition dialog box, shown in Figure 7-1:

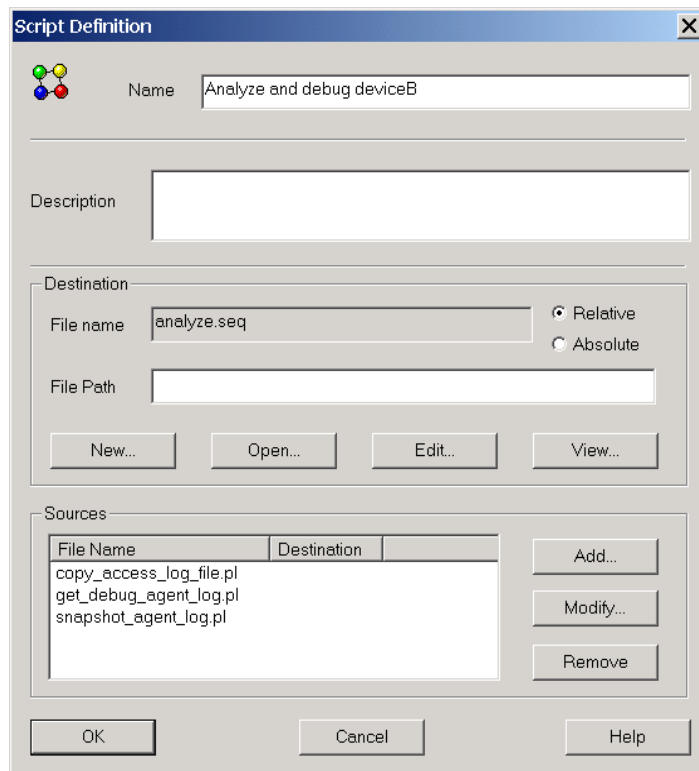


Figure 7-1. Creating a script definition for a project in Axeda Builder

When this project is saved and downloaded from Axeda Builder, the Sequence file and related scripts are downloaded to the asset along with other project files.

If the sequence file selected for a script definition is located on the Axeda Builder computer, then, when it downloads project files to an asset, Builder copies the *.seq file to the Axeda Connector program directory (for standalone Connector projects) or to a *Scripts for model* subdirectory of the *Gateway\DefaultProject\model* directory (for Gateway projects) by default. The script files (Perl or other scripting language), as well as any absolute sequence files identified in script definitions, must be available on the asset.

You can configure the script definition to point to an Absolute or Relative path for the sequence file on the asset. If you choose Relative, Builder downloads the Sequence file to the File Path specified, relative to the program directory for the Axeda Agent. If you choose Absolute, Builder downloads the Sequence file to the File Path specified, without regard to the Agent program directory. If you leave the default selection (Relative) and the File Path field empty, Builder downloads the Sequence file to the default location.

Create Logic Schemas to Run Scripts

To configure the Axeda Agent to run a script defined in the project, you can use Axeda Builder to define a Logic Schema that has a trigger and the action, *Run Script*. For example, you can define a Logic Schema that directs the Axeda Agent to run a script based on a timer that fires every time the Axeda Agent starts up, as follows:

1. In the Project window of Axeda Builder, right-click **Logic Schemas**, and select **Add New Schema**.
2. In the Logic Schema dialog box, type a **Name** for this Logic Schema. In this example, the Name is `OnTimerParse1`.
3. In the Trigger tab of the Logic Schema dialog box, select **Timer** as the type of trigger.
4. When the list of Timers configured in the project becomes available, select the name of the Timer to use. In this example, the name of the Timer is `ParseSnapTimer`.

Figure 7-2 shows the Trigger tab of the Logic Schema dialog box with this example:

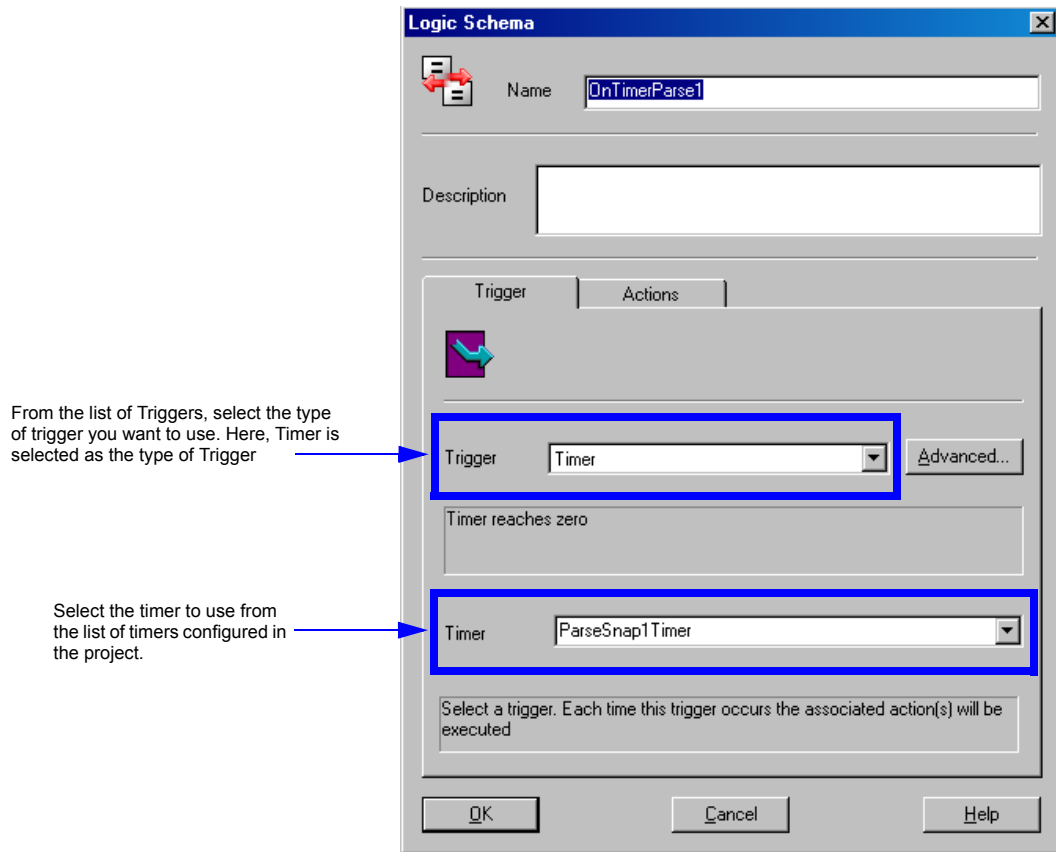


Figure 7-2. Defining the Trigger for a Logic Schema

5. Click **Actions** to display the tab.
6. From the list of **Actions**, select **Run Script**.

- When the list of **Scripts** (definitions) configured in the project becomes available, select the name of the Script definition that you want to run. Figure 7-3 shows an example of the Actions tab for this example.

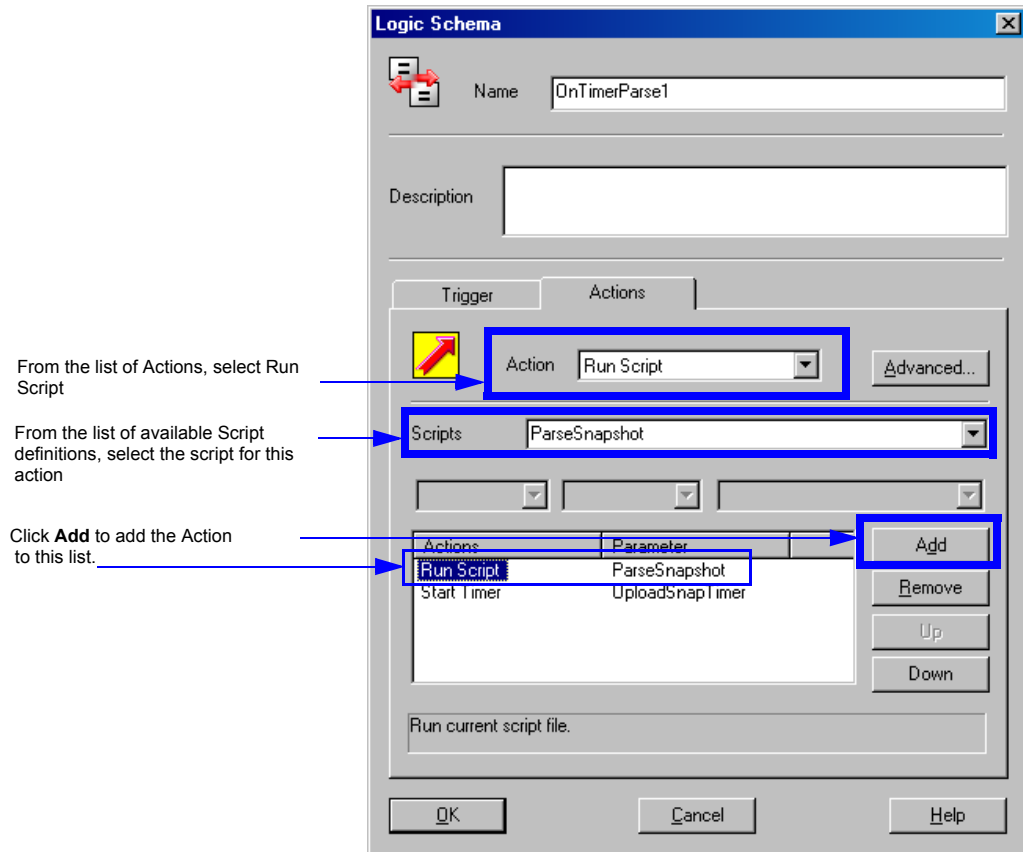


Figure 7-3. Adding the Run Script action

- Click **OK** to save the Logic Schema.

The new Logic Schema, including the name of the trigger and the name of the selected script appear in the Logic Schemas window along with other schemas configured for the project, as shown in Figure 7-4.

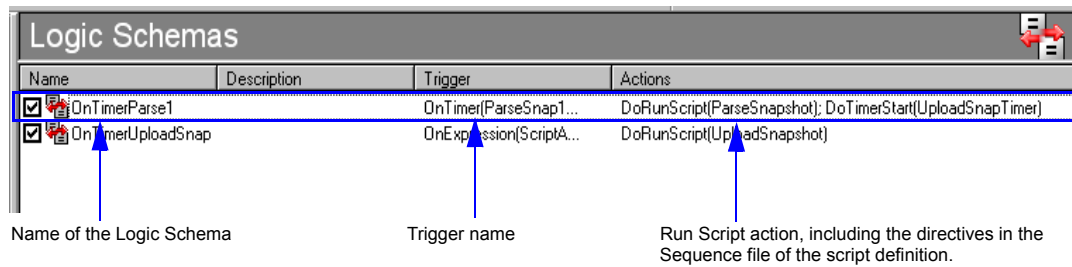


Figure 7-4. Logic Schemas window — all logic schemas currently configured in the project

The Actions column of the Logic Schemas window shows the directives used in the Sequence file of the Script definition.

When it runs its configured scripts or scripts sent from the Axeda Enterprise server, the Axeda Agent sends the status of those scripts to the Axeda Enterprise server for display in the Audit log of the Service Application. You can view script status in the **Audit** module of an Asset dashboard. You can also view script status by clicking **View All** in the Scripts module of the Asset dashboard.

The Axeda Agent can send the status of its scripts and timers to the Axeda Enterprise server when a script starts or when it exits (whether it exits successfully or not). If the user running the Axeda Service application requests an update of the script status for an asset, the Axeda Agent sends its status when it next contacts the server.

Each time it registers with or contacts the server, the Axeda Agent provides a list of all scripts and timers currently defined in the Axeda Agent’s configuration. In this manner, the Axeda Enterprise server and the Axeda Agent remain in sync with regard to available, published, registered, disabled, and removed scripts.

If you need more information about the pages of the Axeda Service and Axeda Configuration applications, refer to the online help for each page of these applications.



Appendix A Scripting Examples

This appendix provides two examples of Axeda scripts, showing the code for the sequence files as well as the Perl code for the scripts called by the sequence files.

Example Perl Scripts for Axeda Platform

The following scripts are examples of typical operations you may want to perform on an asset. These scripts were created in Perl 5.x. It is assumed that you have experience with programming in Perl. For more information about the Perl programming language, refer to the CPAN (Comprehensive Perl Archive Network) Web site, <http://www.perl.com/CPAN/>.

Wizard Example Script: Files and Instructions

This *script* includes the following files:

- ◆ **wizard.seq** — The sequence file for this script. Although Axeda Builder creates Sequence files with *.seq* file name extensions, *.seq* is not required for script processing. When a script is created, whether from the Axeda Enterprise server or Axeda Builder, the Sequence file is the file registered as the script and the file name extension is not important. The *wizard.seq* file contains more than one section, although only the [RUN] section is used.
- ◆ The following Perl scripts are run according to the sequence and attributes defined in *wizard.seq* when the Axeda Agent starts the script:
 - **start.pl** — prints asset information, version of the snapshot, and the time the snapshot was created
 - **snmp.pl** — creates an SNMP session, gets SNMP information from the assets connected to this Axeda Gateway, and returns a session and an error
 - **file.pl** — retrieves a list of files from the asset
 - **env.pl** — retrieves information about the asset's environment
 - **registry.pl** — retrieves information from the asset's operating system registry
 - **os.pl** — retrieves information about the operating system on the asset
 - **end.pl** — ends with printing results to the snapshot file

When this example script runs on an asset, it creates a *snapshot.xml* with the asset information gathered from the script results.

Example Sequence File - Wizard.seq

```
# DoRegisterScript action will run scripts in this section
[REGISTER]
# no instructions defined
# add commands to run upon installation or registration

# DoRunScript action will run scripts in this section
[RUN]
# run start.pl; Agent provides model and serial numbers;
# send script results to snapshot.xml
perl start.pl ${device.modelNumber} ${device.serialNumber} > snapshot.xml

# run snmp.pl on server "snmpsrv";
# send script results to snapshot.xml
perl snmp.pl snmpsrv >> snapshot.xml

# run file.pl env.pl registry.pl;
# send script results to snapshot.xml
perl file.pl env.pl registry.pl >> snapshot.xml

# run env.pl; send script results to snapshot.xml
perl env.pl >> snapshot.xml

# run os.pl; send script results to snapshot.xml
perl os.pl >> snapshot.xml

# run registry.pl; send script results to snapshot.xml
perl registry.pl >> snapshot.xml

# run end.pl; send script results to snapshot.xml
perl end.pl >> snapshot.xml
# upload the snapshot.xml to the server, using the Software Management
# hint word in square brackets, [snapshot], to indicate to the
# server that the uploaded file is a snapshot
$(upload snapshot.xml [snapshot])

# DoUnregisterScript action will run scripts in this section
[UNREGISTER]
# no instructions defined; add commands to run upon script uninstall or
unregister
```

Example Script - start.pl

```
#!/usr/bin/perl
$model = $ARGV[0]; # for this model
$device = $ARGV[1]; # for this asset

# extract information from the asset: time, model, asset
print qq{<Snapshot version="1.0" time="time" model="$model"
device="$device">\n}
```

Example Script - snmp.pl

```
#!/usr/local/bin/perl
use XML::Writer;
use strict;
use Net::SNMP qw(snmp_dispatcher oid_lex_sort);

my $writer = new XML::Writer(NEWLINES => 1);
$writer->startTag("node", "name" => "SNMP");

# Create the SNMP session; return a session and error
my ($session, $error) = Net::SNMP->session(
    -hostname => $ARGV[0] || 'localhost',
    -community => $ARGV[1] || 'public',
    -port => $ARGV[2] || 161,
    -version => 'snmpv2c'
);

# Determine if the session was created
if (!defined($session)) {
    printf("ERROR: %s.\n", $error);
    exit 1;
}

$writer->startTag("node", "name" => "Information");
getOID('1.3.6.1.2.1.1.1.0', 'Description');
getOID('1.3.6.1.2.1.1.5.0', 'Name');
getOID('1.3.6.1.4.1.77.1.4.1.0', 'Domain');
getOID('1.3.6.1.2.1.1.3.0', 'Sysuptime');
getOID('1.3.6.1.2.1.25.2.3.1.3.1', 'Drive 1');
getOID('1.3.6.1.2.1.25.2.3.1.3.2', 'Drive 2');
getOID('1.3.6.1.2.1.25.2.3.1.3.3', 'Drive 3');
getOID('1.3.6.1.2.1.25.3.2.1.3.7', 'Keyboard');
getOID('1.3.6.1.2.1.25.3.2.1.3.8', 'Mouse');
getOID('1.3.6.1.2.1.25.6.3.1.2.1', 'Version');
$writer->endTag;

$writer->startTag("node", "name" => "Processes");
getTable('1.3.6.1.2.1.25.4.2.1.2');
$writer->endTag;

$writer->startTag("node", "name" => "10892");
getTable('1.3.6.1.4.1.674.10892.1.100');
```

```

$writer->endTag;

$writer->startTag("node", "name" => "POST");
getTable('1.3.6.1.4.1.674.10892.1.300.30.1.5.1');
$writer->endTag;

$writer->startTag("node", "name" => "Event Log");
getTable('1.3.6.1.4.1.674.10892.1.300.40.1.5');
$writer->endTag;

# Get some tables
$writer->startTag("node", "name" => "ifTable");
getTable('1.3.6.1.2.1.2.2');
$writer->endTag;

$writer->startTag("node", "name" => "Another");
getTable('1.3.6.1.2.1.1');
$writer->endTag;

$session->close;

$writer->endTag;
$writer->end();
# =====
sub getOID {
    my $oid = $_[0];
    my $name = $_[1];
    my $result;
    if (defined($result = $session->get_request(-varbindlist => [$oid]))) {
#         printf("%s -> %s\n", $name, $result->{$oid});
        $writer->startTag("leaf", "name" => $_[1]);
        $writer->characters($result->{$oid});
        $writer->endTag;
    }
}
# =====
sub getTable {
    my ($oid) = @_;
    my $result;
    my $length = length($oid)+1;
    my $status = "error";
    if (defined($result = $session->get_table(-baseoid => $oid))) {
        foreach (oid_lex_sort(keys(%{$result}))) {
            my $value = $result->{$_};

```

```

        chomp($value);
        $value =~ s/\x00//g;
#printf("%s => %s\n", substr($_,$length), $value);
        if($value =~ /failure/){
=> "error"); $writer->startTag("leaf", "name" => substr($_,$length), "status"
        } else {
            $writer->startTag("leaf", "name" => substr($_,$length));
        }
        $writer->characters($value);
        $writer->endTag;
    }
    print "\n";
} else {
# If the session was bad, the error is printed and the script
# exits; this may be an error at the asset
    printf("ERROR: %s.\n\n", $session->error());
}
}
}

```

Example Script - files.pl

```

#!/usr/bin/perl
# Output each file in the arglist
use XML::Writer;

my $writer = new XML::Writer();
$writer->startTag("node", "name" => "Files");

# create node Files for snapshot.xml
foreach my $filename (@ARGV) {
    open MYHANDLE, $filename or die;
    $writer->startTag("leaf", "name" => $filename);
    my $line;
    while($line = <MYHANDLE>){
        $writer->characters($line);
    }
    $writer->endTag;
}
$writer->endTag;
$writer->end();

```

Example Script - env.pl

```
#!/usr/bin/perl
use XML::Writer;

my $writer = new XML::Writer();
$writer->startTag("node", "name" => "Environment");
# create node Environment for snapshot.xml
foreach (sort keys %ENV)
{
    $writer->startTag("leaf", "name" => $_);
    $writer->characters($ENV{$_});
    $writer->endTag;
}
$writer->endTag;
$writer->end();
```

Example Script - registry.pl

```
#!/usr/bin/perl
use Win32::TieRegistry( Delimiter=>"#", ArrayValues=>0 );
use XML::Writer;
$pound= $Registry->Delimiter("|");
$BaseKey = $Registry->{"|LMachine|SOFTWARE|Axeda|"} || die "Could not open
key";

my $writer = new XML::Writer();
$writer->startTag("node", "name" => "Registry");
# creates node Registry for snapshot.xml
subkeys($BaseKey);
$writer->endTag;
$writer->end();
sub subkeys {
    my ($key) = @_;
    # print this key's values
    my @value_names = $key->ValueNames;
    foreach my $value (@value_names){
        my $valueData = $key->GetValue($value);
        $writer->startTag("leaf", "name" => $value);
        $writer->characters($valueData);
        $writer->endTag;
    }
    # recurse for all subkeys
    my @subkey_names = $key->SubKeyNames;
```

```

    foreach my $name (@subkey_names){
        $writer->startTag("node", "name" => $name); # creates sub-nodes for
each name element returned from script
        my $subkey = $key->{$name};
        subkeys ($subkey);
        $writer->endTag;
    }
}

```

Example Script - os.pl

```

#!/usr/bin/perl
use XML::Writer;

my $writer = new XML::Writer();
$writer->startTag("node", "name" => "OS");
# creates node, OS, for the snapshot.xml

$writer->startTag("leaf", "name" => "Version");
# creates leaf, Version, for the snapshot.xml
open(PS, "ver |");
while (<PS>) {
    chomp;
    if($_ =~ /Version/){
        $writer->characters($_);
    }
}
$writer->endTag;

$writer->startTag("leaf", "name" => "Hostname");
# creates leaf, Hostname, for the snapshot.xml
open(PS, "hostname |");
while (<PS>) {
    chomp;
    $writer->characters($_);
}
$writer->endTag;

$writer->startTag("leaf", "name" => "tracert home");

# creates leaf, tracert home, for the snapshot.xml
open(PS, "tracert www.axeda.com |");
while (<PS>) {
    chomp;

```



```

        $writer->characters($_);
    }
    $writer->endTag;

    $writer->startTag("leaf", "name" => "Route");
    # creates leaf, Route, for the snapshot.xml
    open(PS, "route print |");
    while (<PS>) {
        chomp;
        $writer->characters($_);
    }
    $writer->endTag;

    $writer->endTag;
    $writer->end();

```

Example Script - end.pl

```

#!/usr/bin/perl
print qq{</Snapshot>} # prints all script results to the snapshot

```

Analyze Log Example Script: Files and Instructions

This script includes the following files:

- ◆ **analyze.seq** — This is the sequence file for this script. This script runs the `copy_agent_log_file` and `get_debug_agent_log` scripts and saves their results to `.log` and `.txt` files, respectively. Then, the script runs the `snapshot_agent_log` script, gets the model and serial numbers for the asset and saves the results in the `agent_log_snapshot` (XML format) file. Finally, the script uploads the XML-formatted file created with information from the asset and defines a **hint** of `[snapshot]` to identify the file contents and structure to the Axeda Service application and Snapshot Viewer for online viewing.

Only the commands in the `[RUN]` section of the `analyze.seq` file are run.

- ◆ The following Perl scripts are run according to the sequence and attributes defined in the `analyze.seq` file when the Axeda Agent starts the script:
 - **copy_agent_log_file.pl** — retrieves the Axeda Agent's log file (either `ekernel.log` for Axeda Connector assets or `xgate.log` for Axeda Gateway assets)
 - **get_debug_agent_log.pl** — reads the Axeda Agent log file, locates agent messages with a message type of `DEBUG` and prints those messages to a file, `axedaagent.log`.

- **snapshot_agent_log.pl** — creates *agent_log_snapshot.xml*, the XML snapshot file, with the following information from the asset: version, time, model, and asset/serial number. The file will contain information resulting from running this script, including debug messages from the asset.

analyze.seq

```
[RUN]
perl copy_agent_log_file.pl > axedaagent.log
perl get_debug_agent_log.pl > debug_agent_log.txt
perl snapshot_agent_log.pl ${device.modelNumber} ${device.serialNumber} >
agent_log_snapshot.xml
$(upload agent_log_snapshot.xml [snapshot])
```

copy_agent_log_file.pl

```
#!/usr/bin/perl
# Axeda Systems

$connector = "ekernel.log";
$gateway = ".././../xgate.log";

open(FILE, $connector);
while ($line = <FILE>) {
    chomp($line);
    $line =~ s/\0//g;
    $line =~ s/ÿ//g;
    $line =~ s/p//g;
    $line =~ s/\n//g;
    print $line;
    print "\n";
}
close(FILE);

open(FILE, $gateway);
while ($line = <FILE>) {
    chomp($line);
    $line =~ s/\0//g;
    $line =~ s/ÿ//g;
    $line =~ s/p//g;
    $line =~ s/\n//g;
    print $line;
    print "\n";
}
close(FILE);
```

get_debug_agent_log.pl

```
#!/usr/bin/perl
# Axeda Corporation
use XML::Writer;
my $log;
$match = "DEBUG";
```

```

open(FILE, "axedaagent.log");
while ($line = <FILE>) {
    chomp($line);
    $line =~ s/\0//g;
    $line =~ s/\n//g;
    if ($line =~ /$match/) { # look for lines with message type DEBUG
        $log = $line;
        print $log;
        print "\n";
    }
}
close(FILE);

```

snapshot_agent_log.pl

```

#!/usr/bin/perl
# Axeda Corporation

use XML::Writer;
use TimeUtils;

$model = $ARGV[0];
$device = $ARGV[1];

$time = TimeUtils::getLocalTime();

my $writer = new XML::Writer();
$writer->startTag("Snapshot",
    "version" => "1.0",
    "time" => "$time",
    "model" => "$model",
    "device" => "$device");

$writer->startTag("node", "name" => "Axeda Agent Log");
importFile("debug_agent_log.txt", "Debug Entries");
importFile("agentagent.log", "Log Entries");

$writer->endTag;
$writer->endTag;
$writer->end();

# =====
sub importFile {
    my $filename = $_[0];
    my $title = $_[1];
    open (MYHANDLE, $filename) or die;
    $writer->startTag("leaf", "name" => $title);
    while(my $line = <MYHANDLE>){
        $writer->characters($line);
    }
    close (MYHANDLE);
    $writer->endTag;
}

```

Index

Symbols

- # in the first column 3-11
- <alarm> element
 - <upload> sub element 4-5
 - attribute definitions 4-4, 5-9
 - data source files 4-2
 - DTD declaration 5-6
 - example 5-11
 - syntax 4-4
- <dataitem> element
 - attribute definitions 4-6, 5-10
 - data source files 4-2
 - DTD declaration 5-6
 - syntax 4-6
- <dataitemgroup> element 5-8
 - data source files 4-2
 - DTD declaration 5-6
 - example 5-12
 - syntax 4-3
- <event> element
 - attribute definitions 4-7, 5-10
 - data source files 4-2
 - DTD declaration 5-6
 - example 5-11
 - syntax 4-7
- <file> sub element of <upload>
 - attributes 5-9
 - DTD declaration 5-7
- <leaf> element
 - attributes 5-8
 - DTD declaration 5-6
- <node> element
 - attributes 5-7
 - DTD declaration 5-6
- <snapshot> element
 - attribute definitions 5-7
 - attributes 5-7
 - DTD declaration 5-6
- <upload> sub element of <alarm>
 - DTD declaration 5-7

- <upload> sub element of <alarm>
 - example 5-9
- <upload> sub element of <alarm>
 - syntax 4-5
 - `\${device.address}` 3-11
 - `\${device.community}` 3-11
 - `\${device.dataItems(data_item_name)}` 3-11
 - `\${device.modelNumber}` 3-11
 - `\${device.password}` 3-11
 - `\${device.serialNumber}` 3-11
 - `\${device.username}` 3-11

A

- Action wizard (Axeda Configuration) 7-2
- actions for logic schemas 7-5
- Agent directives 3-13
- Agent Timers (Axeda Service) 6-15
- Agent Triggers (Axeda Service) 6-15
- arguments, using parameter substitution values as 3-12
- Asset dashboard 5-19, 6-14
- asset data vs snapshot data 5-2
- asset password, retrieving 3-11
- assets for a script, selecting 6-10
- async argument for upload directive 3-13
- attributes
 - <alarm> element 5-9
 - <dataitem> element 5-10
 - <event> element 5-10
 - <leaf> element 5-8
 - <node> element 5-7
 - <snapshot> 5-7
 - <snapshot> element 5-7
 - viewing in Snapshot Viewer 5-16
- Authentication Helper and scripts 3-6
- Available Scripts (Axeda Service) 6-15
- Axeda Builder
 - enabling script debugging 3-5
- Axeda Builder, script tools 7-3
- Axeda Configuration
 - stopping scripts 6-4
 - View and manage scripts page 6-8
- Axeda Gateway

- relative path attribute for alarms 4-5
- retrieving IP addresses of SNMP assets 3-11
- running scripts 6-4
- script considerations 3-4
- SNMP community, retrieving for assets 3-11
- Axeda Service 5-19
 - script tools 6-12
 - stopping scripts 6-4

B

- best practices for scripts 3-3

C

- chat script (Authentication Helper) 3-6
- comment symbol 3-11
- comparing snapshots 5-19
- correcting parser errors 5-18
- creating scripts, Axeda Builder 7-3

D

- data collection 2-2
- data item values, retrieving 3-11
- Data Loggers 5-4
- data source file 4-2
- debugging scripts 3-5
- Details Attributes view (Snapshot Viewer) 5-15
- Details view (Snapshot Viewer) 5-15
- directives, Agent 3-13
- Document Type Definition (DTD) for snapshots 5-5–5-6
- documentation for Axeda products 1-3
- dynamic script, definition 6-5
- dynamic timers 6-4

E

- entity references (table) 5-18
- error handling for scripts 3-5
- examples
 - analyze log A-9
 - block of data items 5-12
 - environment information (Perl) A-7
 - extract information from asset A-4
 - files (Perl) A-6
 - operating system info (Perl) A-8

- Perl scripts A-2
- printing to snapshot file (Perl) A-9
- registry information (Perl) A-7
- Sequence file A-3
- sequence files 3-9
- snapshot files 5-11
- SNMP (Perl) A-4
- Explorer view (Snapshot Viewer) 5-15–5-16
- exporting results of file comparison 5-22

F

- File Comparison link (Asset dashboard) 5-19
- file compression for uploads 3-13
- filtering comparison results 5-21

H

- hint argument for upload directive 3-13

I

- IP addresses, retrieving 3-11

L

- Last Result status 6-6
- logic schemas to run scripts 7-4

M

- managing scripts from Axeda Configuration and Service 6-2
- model numbers, passing with string replacement 3-12
- model numbers, retrieving 3-11

N

- navigating Snapshot Viewer 5-16

O

- overview of Axeda Platform scripting 2-3

P

- packages, using to run scripts 6-12
- parameter substitution 3-10
- parameter substitution, table 3-11

- parameters for Authentication Helper 3-6
- parser errors 5-18
- parsing XML output files 3-13
- password authentication for scripts 3-6
- pathDeviceRelative (attribute for <alarm>) 4-5
- Perl script examples A-2
- publishing scripts 6-2
- publishing scripts (Axeda Configuration) 6-9

Q

- quality attribute of <dataitem> element 4-6

R

- registering scripts 6-2
- registration of scripts 6-9
- Run Script actions 7-2
- running scripts 6-3
- running scripts locally or through Telnet 6-4

S

- script files, overview 2-6
- script status 6-6
- script status messages 6-7
- scripting overview 2-3
- scripts
 - debugging 3-5
 - Gateway considerations 3-4
 - managing with Axeda Configuration 6-8
 - registrations 6-9
 - writing 3-3
- Scripts module (Asset dashboard) 6-14
- searching file comparison results 5-21
- searching in Snapshot Viewer 5-17
- sequence files
 - creating 3-7
 - examples 3-9
 - format 3-8
 - naming 3-10
 - overview 2-5
 - parameter substitution 3-10
 - timeouts, adding 3-10
 - where stored 7-4
- sequence files, creating in Axeda Builder 7-3
- serial numbers, retrieving 3-11
- Snapshot Viewer 5-3, 5-13–5-18

- snapshots
 - comparing 5-19–5-23
 - creating with scripts 5-4
 - definition 5-2
 - DTD 5-5–5-6
 - editing 5-18
 - examples 5-11
 - filtering content 5-17
 - parsing 3-13
 - searching 5-17
 - uploading 3-13
 - uses 5-2
 - viewing 5-2
 - viewing contents 5-16
 - XML elements, defined 5-4
- SNMP example (Perl) A-4
- sorting file comparison results 5-21
- status of running scripts, viewing 7-7
- stopping scripts 6-4
- string replacements 3-11
- substitution (parameters) 3-10
- substitutions (parameter) 3-11
- supported scripting languages 2-6
- syntax for data source file 4-2

T

- timeout directive 3-10
- timers, status 6-6
- triggers for logic schemas 7-5
- troubleshooting
 - Authentication Helper not working with Windows
 - Telnet and FTP 3-7
 - entity reference expected 5-18
 - Parser errors (Snapshot Viewer) 5-18
 - runaway scripts, preventing 3-10
 - snapshot file not displaying in Viewer 5-13
 - Snapshot Viewer timed out 5-15
- type argument for upload directive 3-13
- types of script files 2-4

U

- uncompressed (argument for upload directive) 3-13
- uploading multiple files 3-13
- uploading XML output files 3-13

user name for network access, retrieving 3-11
utc attribute 4-6

V

valid XML for snapshots 5-6
viewing registered scripts for an asset 6-14

W

writing scripts 3-3

X

XML definitions 5-5
XML elements for snapshots 5-4
XML entity references (table) 5-18
XML for snapshots 5-5