TIBCO Hawk® Database Adapter

User's Guide

Version 6.2.1
July 2022

Contents

Introduction

This section gives an overview of TIBCO Hawk® Database Adapter, using various underlying message transport mechanisms to communicate with the TIBCO® Operational Intelligence Hawk® RedTail system.

Note: TIBCO® Operational Intelligence Hawk® RedTail has been used to represent TIBCO® Operational Intelligence Hawk® RedTail, TIBCO® Operational Intelligence Hawk® RedTail - Container Edition, TIBCO® Operational Intelligence Hawk®, and TIBCO Hawk®. All processes and procedures applicable to TIBCO® Operational Intelligence Hawk® RedTail are also applicable to TIBCO® Operational Intelligence Hawk® RedTail - Container Edition, TIBCO® Operational Intelligence Hawk®, and TIBCO Hawk®, unless explicitly stated otherwise.

Product Overview

Nearly every enterprise relies on relational database technology for managing critical business information. For organizing, storing, and retrieving data, relational databases are now a cornerstone of the business computing environment. Database technology is so versatile and widely used that it is common to find products from multiple vendors installed at a single site, each with a specialized role. One type of RDBMS might support an enterprise application package such as SAP, while another might store large numbers of daily transactions, or content for an external web site. While a heterogeneous database environment has advantages, it also presents challenges for convenient access, integration, and monitoring.

Using the adapter you can integrate one or more relational databases with your existing TIBCO® OI Hawk® RedTail monitoring environment. This ability to link a database with other systems and applications is valuable because database tools, while robust and feature-rich, assume that the database server is completely isolated from other

environments. For example, the machine supporting an Oracle server might be running low on disk space, memory, or both. TIBCO OI Hawk RedTail can detect the resource problem, and using the adapter, automatically run a SQL script to correct the problem. Or, according to the logic defined in a rulebase, running an operating system script might be the best way to free more resources. The DBA can also be notified which script was run.

The ability to monitor and manage database servers is one significant benefit of the adapter. Another benefit is a flexible design that provides TIBCO OI Hawk RedTail users with transparent database access. The adapter includes a wrapper for packaging SQL statements as Hawk microagent methods, which the local Hawk agent uses to access a database. Custom methods are easy to build, and can contain almost any type of SQL statements, including stored procedures and functions. Invoking an adapter method executes the associated SQL statement.

Unlike database management tools, the adapter is not limited to administrative tasks. As a method can execute almost any SQL statement, using custom adapter methods you can access any content stored in a database, such as noncritical application data. For example, an application that processes inventory information stores shipment data in a relational database. Every Friday a report is generated that calculates weekly totals and must complete by 7pm. If an invalid record in the database causes the report generation to fail, a rulebase can detect this condition. A custom adapter method can remove the invalid record and store it in a table or file for later analysis. In addition to regenerating the report, TIBCO OI Hawk RedTail can also notify the correct person of the error.

Multiple instances of the adapter can run on a single machine, which also runs a Hawk agent. Each adapter might communicate with a database from a different vendor. Data retrieved from these diverse sources is processed by the same local agent. This design provides the potential to integrate isolated enterprise data stores. For example, data in a Sybase sales order entry system could be monitored by one adapter, and the results could cause a stored procedure to execute in an Oracle inventory system. In this scenario, the Oracle database is accessed by a separate instance of the adapter running on the same machine.

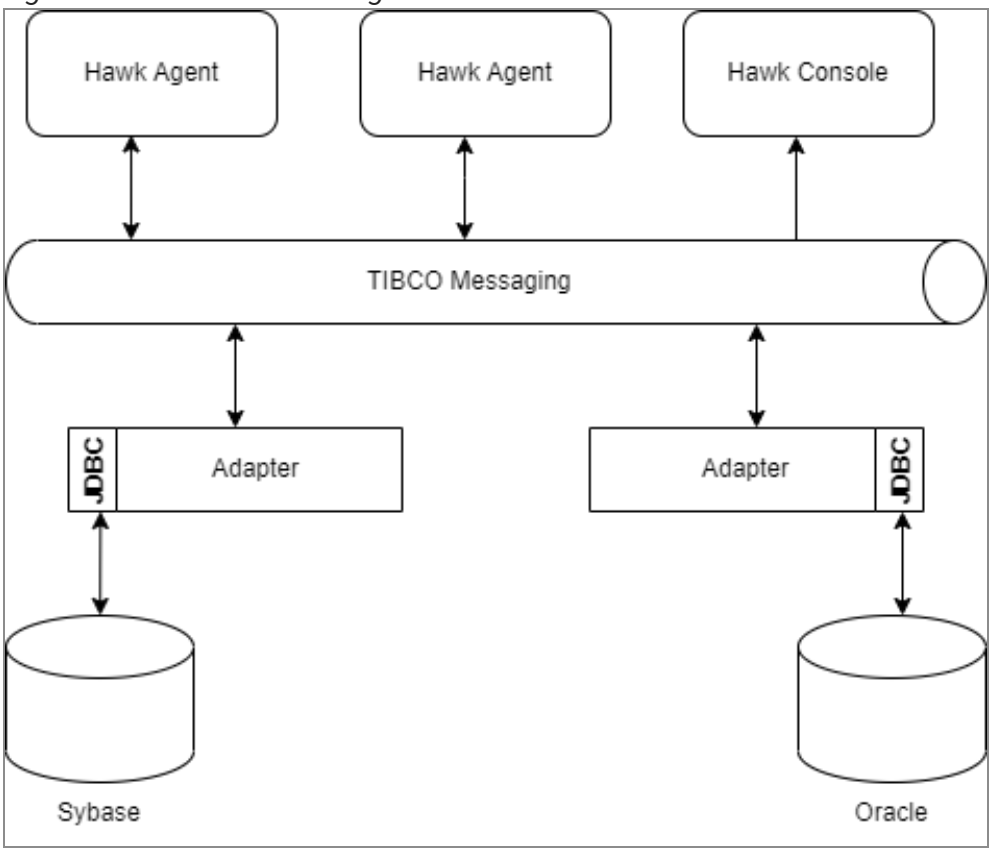
The typical enterprise computing environment consists of many distributed applications and systems, each performing a different function. The challenge is to provide a framework that allows information to flow quickly and seamlessly to and from each part of the system. The TIBCO OI Hawk RedTail system is flexible enough to support these variations, both internally and through components like the adapter that are designed to communicate with external environments.

Architecture

Hawk® Database Adapter is a TIBCO OI Hawk RedTail alert subscriber. Like the Hawk Console, it receives messages published using subject-based addressing. Multiple instances of the adapter can run on the same network to achieve fault tolerance, load-balancing, or both.

The adapter is also a Hawk microagent. Like other microagents, it uses AMI to communicate directly with the Hawk agent on the local node. The adapter microagent exposes an interface with methods and arguments so you can manage its activities from TIBCO OI Hawk RedTail. For more information, see Using TIBCO Hawk Database Adapter Methods.

Figure 1: Architecture in Target Environments



The Hawk® Database Adapter provides capabilities to externally instrument various RDBMS using Hawk AMI API and uses JDBC to communicate (through a pure Type - 4 JDBC driver) with the database listener. A Type - 4 JDBC driver is both database-independent and platform-independent, and provides performance that is comparable to a native connection.

The database that is connected through such a JDBC driver is exposed as a Hawk microagent to the local Hawk agent. It is instrumented with HAWK AMI API. Various system metadata level information is configured to be available by means of various microagent methods out-of-the-box. These methods may vary by the RDBMS type and version and are consolidated in a simple easy-to-understand and easy-to-augment XML file. They can contain almost any type of SQL statements - DML, DDL, DCL, Stored Procedure, Adhoc Query, and so on - that can be tested by dynamically invoking them before use. Users can develop their own SQL and embed them in additional microagent methods.

The database performs all enforcement of syntax rules, such as correct data types for INSERT and UPDATE. If an error occurs during the dynamic method invocation, database error messages are returned through TIBCO Hawk Console. If an error occurs during rulebase processing, database error information is written to the Hawk agent log.

Configuration details for the adapter are stored in an XML file that is flexible and easy to modify. The file contains XML elements and attributes that define Transport level configurations and database access parameters, as well as custom adapter microagents and methods. Syntax rules are defined in an external DTD, so a third-party validating parser can be used to validate the XML before it is used.

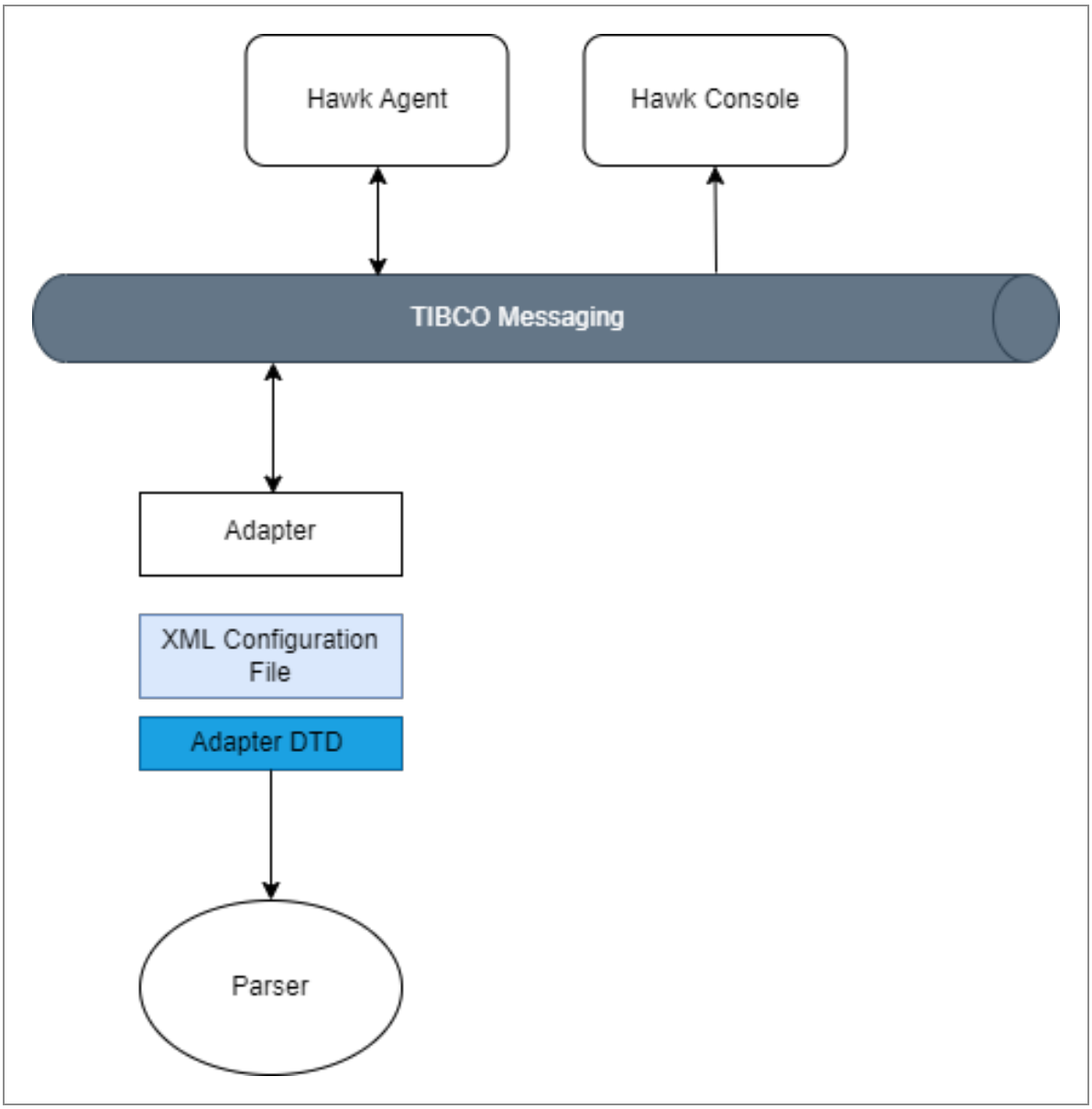


Figure 2: Configuration XML File Parsing

When the adapter is started, it parses and loads the configuration file. Using values specified in the file, the adapter establishes a database connection and then dynamically builds microagents and methods. For more information on configuring the adapter, see Getting Started. For a description of the DTD, see TIBCO Hawk Database Adapter Document Type Definition (DTD).

An advantage of exposing such database level instrumentation mechanisms, as Hawk microagent, is that these methods can be used in a single Hawk rulebase or various Hawk rulebases to create and enhance database level SLA monitoring. All results returned by

such Hawk database adapter methods running on the same machine can be accessed by the local agent. Using this design, data returned by one instance of the adapter to be used as the data source of another rule. The Hawk agent can then apply logic to the results and possibly take action. The action can be another method invocation, such as executing an SQL statement against a different database.

In addition to custom microagents and methods defined in the configuration file, the adapter is also represented by a Hawk microagent. Like other microagents, it uses AMI to communicate directly with the Hawk agent on the local node. The adapter microagent exposes an interface with methods and arguments so you can manage its activities from Hawk Console. For more information, see Using TIBCO Hawk Database Adapter Methods. For more information about Hawk agents, microagents, and Hawk Console, see the *TIBCO® Operational Intelligence Hawk® RedTail Installation, Configuration, and Administration* guide.

Functional Components

This section provides a general overview of each product component used by the adapter.

TIBCO Hawk

TIBCO OI Hawk RedTail is a tool for monitoring distributed systems and applications based on the TIBCO Rendezvous, TCP Transport for TIBCO Hawk, TIBCO or TIBCO Enterprise Messaging Service messaging system. The TIBCO OI Hawk RedTail monitoring solution is highly flexible, scalable, and easy to implement.

In a TIBCO OI Hawk RedTail environment, agents on each local computer perform the monitoring work. Hawk agents use microagents to represent and interact with managed objects, such as an application or operating system component. Microagents use methods to extract monitoring information and carry out specified tasks internally, in the managed object. In this way, a total decoupling of the management data from the management rules or policies can be done. Microagent methods can be accessed and invoked from TIBCO OI Hawk RedTail, TIBCO Administrator, or Hawk Console API applications. They are also used with the agent's powerful rules engine in autonomous monitoring.

Monitoring logic is stored in rulebases on each agent. Rulebases contain rules, which are user-defined statements about a managed object. Administrators can easily construct rules that carry out specific tests and actions. No special syntax or scripting is required. Rules

can then be grouped and distributed to multiple agents with similar monitoring requirements.

The Hawk Console or TIBCO Administrator are various console applications available to monitor managed objects within your infrastructure network. Each monitored computer is represented by an icon in the main window. Alert messages published by the Hawk agents on TIBCO Messaging software are also displayed. All users can view the same managed objects without complicated configuration schemes, and each user can customize the interface without affecting others. For more information, see *TIBCO® Operational Intelligence Hawk® RedTail User's Guide.*

TIBCO Hawk Transports

TIBCO Rendezvous

Using TIBCO Rendezvous, you can exchange data among applications across a network. It provides software support for network data transport and network data representation. TIBCO Rendezvous supports many hardware and software platforms, so applications running on many different kinds of computers on a network can communicate seamlessly.

TIBCO OI Hawk RedTail uses TIBCO Rendezvous for all network and intra machine communications between TIBCO OI Hawk RedTail product components. Both broadcast and multicast network environments are supported. Connectivity across wide-area networks is provided by the TIBCO Rendezvous routing daemon, which routes messages efficiently and reliably between networks.

TCP Transport for TIBCO Hawk

TCP Transport for TIBCO Hawk is a TCP based transport for Hawk components using the Akka clustering designs. The TCP Transport for TIBCO Hawk removes the dependency on an external server or transport (such as TIBCO Rendezvous and TIBCO Enterprise Messaging Service) as the TCP communication happens peer to peer. Hawk® can be easily deployed on the cloud when TCP Transport for TIBCO Hawk is used.

TCP Transport for TIBCO Hawk is distributed as part of the standard TIBCO OI Hawk RedTail installation. For more information on TCP Transport for TIBCO Hawk, see *TIBCO® Operational Intelligence Hawk® RedTail Documentation*.

AMI Protocol

The TIBCO Hawk Application Management Interface (AMI) is an open interface for instrumenting an application with a powerful management interface. There are two forms of API available: TIBCO Rendezvous-based protocol and TCP-based protocol. Using AMI, applications can be monitored and controlled internally with TIBCO OI Hawk RedTail in the same way as any other system component. The protocol allows for complete independence between the TIBCO Hawk agent and the managed application with a two-way dynamic discovery.

AMI features are:

- AMI API libraries, which implement the TIBCO Rendezvous-based protocol, are provided in C, C++ and Java, while TCP-based implementations are provided in Java. Direct AMI using the TIBCO Rendezvous APIs are also supported in ActiveX, and Perl.
- AMI can be easily back-fitted into existing applications, even those that do not currently utilize TIBCO Rendezvous software, and expose existing internal application methods to a Hawk agent.
- Third-party applications can be managed by writing an AMI wrapper that interfaces with an API or any other mechanism provided by the application for the purpose of managing it.

For more information, see the *TIBCO® Operational Intelligence Hawk® RedTail documentation*.

Getting Started

This section describes how to configure and begin using the Hawk® Database Adapter.

Configuring the Adapter

The following section describes the parameters you can set to configure the adapter. The adapter-specific parameters are set by modifying the default parameters in the adapter configuration file. These parameters specify configurations and license information. This configuration file `hawkdbhma.cfg`, is present in the `<HAWK_DBHMA_HOME>/config` folder. It has the following configurable parameters.

Editing the Adapter Configuration File

To follow the examples in this section, the adapter configuration file must contain the correct values for database environments. The following parameters might require modification before starting the adapter:

For TCP Configuration:

```
hawk_domain="default"
tcp_session = "localhost:2591 localhost:2571"
agent_name=""
JDBCDriver = "oracle.jdbc.driver.OracleDriver"
dbURL = "jdbc:oracle:thin:@localhost:1521:ORCL"
dbType = "ORACLE"
```

```
dbUser = "system"
dbPassword = "manager"
```

For TCP Configuration with TLS Enabled:

```
hawk_domain="default"
tcp_session = "localhost:2591 localhost:2571"
agent_name=""
tcp_key_store = "<file-name>"
tcp_trust_store = "<file-name>"
tcp_key_store_password = "<password_string>"
tcp_key_password = "<password_string>"
tcp_trust_store_password = "<password_string>"
tcp_ssl_protocol = "TLSv1.2"
tcp_enabled_algorithms = "TLS_RSA_WITH_AES_128_CBC_SHA"
JDBCDriver = "oracle.jdbc.driver.OracleDriver"
dbURL = "jdbc:oracle:thin:@localhost:1521:ORCL"
dbType = "ORACLE"
dbUser = "system"
dbPassword = "manager"
```

For Rendezvous Configuration:

```
hawk_domain="default"
agent_name=""
ami_rvd_service = "7474"
ami_rvd_network = ""
ami_rvd_daemon = "tcp:7474"
JDBCDriver = "oracle.jdbc.driver.OracleDriver"
dbURL = "jdbc:oracle:thin:@localhost:1521:ORCL"
dbType = "ORACLE"
dbUser = "system"
dbPassword = "manager"
```

Warning: Any SQL statement or operation allowed by the specified database account can be performed by the adapter. Make sure this access level is appropriate for TIBCO OI Hawk RedTail users who can access adapter methods. Special caution is required when using the `dynamicQuery` and `dynamicExecute` methods, as users can execute any SQL statement they want whenever the adapter containing these two methods is running.

TIBCO Hawk Database Adapter Configuration File Definition

Element Name	Description
`-xml_file`	Database specific AMI configuration XML file, based on `DBHma`.dtd present in `<HAWK_DBHMA_HOME>/xml` directory.
`-log_dir`	The directory of log files which are generated by the DBHma
`-log_max_size`	The maximum size of a rotating log file in KB. You may apply a suffix m or M for indicating MB values.
`-log_max_num`	The maximum number of rotating log files.
`-log_level`	The log level, for debug mode. Set it to 8 or above.
`-log_format`	The format for trace log messages. Specify “default” for default Hawk format or “ae4” for AE format. The default is “default”.
`-tcp_key_ store`	Path of the key store file
`-tcp_trust_ store`	Path of the trust store file
`-tcp_key_ store_ password`	Password for the key store file
`-tcp_key_ password`	Encrypted key password
`-tcp_trust_`	Password for the trust store file

Element Name	Description
store_password	
-tcp_ssl_protocol	Protocol for a secure connection
-tcp_enabled_algorithms	Algorithm to be used for the security protocol. You can specify multiple algorithms as a comma-separated list without any space characters.

Some sample configuration files for different databases are supplied out-of-the box with Hawk® Database Adapter installation and can be found in the <HAWK_DBHMA_HOME>/examples/<*database*> directories. For example, OracleDBHma_12c.xml is a configuration file present in the <HAWK_DBHMA_HOME>/examples/oracle directory. Using any text editor, you can specify values for attributes in the XML file. The new values are used the next time the adapter is started. This file contains sample values, as well as several default method definitions that are helpful for performing routine tasks. You can start using the adapter with the sample file, or create your own file using the sample file as a template. It is suggested that you start with the sample configuration file to understand how the adapter works. Then you can create your own adapter configuration files using this sample configuration file as a template.

The following tables describe each configuration option you can set by editing the adapter configuration file. Except where noted, attribute values can be any string value.

Note: Elements and attributes described in this section can be used to build custom adapter methods. Before building any custom methods, start with this section, then carefully read Building Custom Methods and TIBCO Hawk Database Adapter Document Type Definition (DTD).

The following table describes the top-level element, <*TIBHAWK_AMI*>, which defines TIBCO Rendezvous and database connection parameters.

TIBHAWK_AMI Element Definition

Element Name	Description	
<TIBHAWK_AMI>	This element is a container for the *<microagent>* element.	
	Attribute Name	**Description**
	`xml_file_version`	The version of this configuration file.
	`-hawk_domain`	Specifies the TIBCO OI Hawk RedTail domain to be monitored for TIBCO OI Hawk RedTail alerts by the trap publisher. It has no default value.
	`-tcp_session`	Configures the agent with a TCP session to be used to communicate with applications implementing the TIBCO Hawk Application Management Interface. The syntax for the property is: `-tcp_session` *<self_IP>:<port> <hawk_agent_IP>:<AMI_session_port>* where, • *<self_IP>:<port>* - The unique socket address of the Hawk DB Adapter for AMI communication. • *<hawk_agent_IP>:<AMI_session_port>* - The socket address of the Hawk agent for AMI communication. This socket address is same as *<self_IP>:<port>* specified for the `-ami_tcp_session` parameter in `hawkagent.cfg`.
	`agent_name`	The name of the agent. Each Microagent being monitored must have `agent_name`, by which Microagent is being monitored. This parameter is optional for TIBCO Rendezvous.

Element Name	Description	
		Note: Agent names with multiple words separated by dots are not supported.
	`ami_rvd_service`	The service parameter for the adapter TIBCO Rendezvous session. For example, 7474 listens on network port 7474. For more information on transport parameters, see *TIBCO Rendezvous Concepts.*
<TIBHAWK_AMI>	`traceLevel`	Optional. The level of detail to include in the adapter log file. This parameter is a bit mask formed by adding one or more of the following values: 0 — (default) Logs no debug information 1 — INFO Logs information messages 2 — WARNING Logs warning messages 4 — ERROR Logs error information 8 — DEBUG Logs debug information 16 — AMI Logs AMI debug information 32 — DB Logs SQL statements executed by the database A value of -1 logs all levels of information.
	`log_format`	Optional. You can select the format of log file entries. Specify either the existing Hawk format or the ActiveEnterprise format.
	`app_id`	The Application ID. Only used when the doc format is set to ActiveEnterprise. The element is used in log files.
	`product_id`	The Product ID. Only used when the doc format is set to ActiveEnterprise. The element is used in log files.

Element Name	Description	
	dbType	The type of database. Possible values are: ORACLE, SYBASE, or SQLSERVER.
	JDBCDriver	The name of the JDBC driver used to communicate with the database. The possible values are: • Oracle - oracle.jdbc.driver.OracleDriver • Sybase - com.sybase.jdbc4.jdbc.SybDriver • MySQL - com.mysql.jdbc.Driver • Microsoft SQL Server - com.microsoft.sqlserver.jdbc.SQLServerDriver • IBM DB2 - com.ibm.db2.jcc.DB2Driver For details, refer to your JDBC driver documentation. **NOTE:** When using jConnect5.2, the JDBC driver must use com.sybase.jdbc2.jdbc.SybDriver instead of com.sybase.jdbc4.jdbc.SybDriver
<TIBHAWK_AMI>	ami_rvd_network	The network parameter for the adapter TIBCO Rendezvous session. For example, "" represents a NULL value, which specifies the primary network.
	ami_rvd_daemon	The daemon parameter for the adapter TIBCO Rendezvous session. For example, tcp:7474 uses a local daemon at TCP socket number 7474.
	traceFile	Optional. The fully-qualified path and name of the adapter log file. For example, /tmp/HawkDBHma.log. If no value is specified, error information is written to stdout on UNIX. If the adapter is running on Windows as a Windows service, the Windows Event Log is used.
	traceFileMaxSize	Optional. The maximum size, in kilobytes, that an adapter log file can reach before another log file is started. The default value is 1024 Kb.

Element Name	Description	
	`traceFileMaxNumber`	Optional. The maximum number of log files to keep in the trace directory. After this number of log files is reached, the first file is overwritten. The default value is 4.
	`traceLevel`	Optional. The level of detail to include in the adapter log file. This parameter is a bit mask formed by adding one or more of the following values: 0 — (default) Logs no debug information 1 — INFO Logs information messages 2 — WARNING Logs warning messages 4 — ERROR Logs error information 8 — DEBUG Logs debug information 16 — AMI Logs AMI debug information 32 — DB Logs SQL statements executed by the database A value of -1 logs all levels of information.
<TIBHAWK_AMI>	`dbUser`	The user name for the database account. If this user cannot access tables or perform operations referenced in adapter methods, the method invocation fails. To restrict database access in the TIBCO Hawk environment, specify a restricted user name.
	`dbPassword`	The password (plain text or encrypted) for the database account.
	`dbEnvVars`	Optional. Additional environment variable settings to configure the JDBC driver. For details on environment variable support and usage, see your JDBC driver documentation.

Element Name	Description	
	`dateFormat`	The format for expressing date values, for example `MM/dd/yyyy:HH:mm:ss`. The adapter converts DATE and DATETIME type values retrieved from a database to string type. The possible values are determined by `java.text.SimpleDateFormat`. For details, see Date and Time Value Syntax.
	`dbURL`	The URL of the JDBC driver used to communicate with the database. For example,`jdbc:oracle:thin:@localhost:1521:adb8`. For details, refer to the JDBC driver documentation.
	`sqlFileDir`	The default location of SQL scripts if a full path is not specified. For example, `${hawk.hawk_root}/adapters/dbhma/oracle/sql`, where `hawk.hawk_root` is a Java property variable. For details on variable syntax and usage, see the Java documentation. The default value is the directory where the adapter is started.
	Contained Elements: *<microagent>*	

The following table describes the ***<microagent>*** element, which defines microagents for this instance of the adapter. Multiple microagents can be defined in a single configuration file. These microagents are independent of the standard adapter microagents described in Using TIBCO Hawk Database Adapter Methods.

microagent Element Definition

Element Name	Description	
<TIBHAWK_AMI>	This element is a container for the *<method>* element. It defines a custom microagent for performing database operations.	
	Attribute Name	**Description**

Element Name	Description	
	`name`	The name of this microagent. This value must be unique on the adapter machine.
	`display_name`	Optional. The microagent name as it should appear in Hawk Console. If no value is specified, the `name` value for this element is used.
	`help`	Optional. A help string that describes the microagent. If no value is specified, the `name` value for this element is used.
	`dbURL`	Optional. The URL of the JDBC driver used to communicate with the database. If specified, overrides a `dbURL` value in the *<TIBHAWK_AMI>* element.
	`dbUser`	Optional. The user name for the database account. If this user cannot access tables or perform operations referenced in adapter methods, method invocation does not succeed. To restrict database access in the TIBCO Hawk environment, specify a restricted user. If specified, overrides a `dbUser` value in the *<TIBHAWK_AMI>* element.
	`dbPassword`	Optional. Password for the database account. If specified, overrides a `dbPassword` value in the *<TIBHAWK_AMI>* element.
	`dbEnvVars`	Optional. Additional environment variable settings to configure the JDBC driver. If specified, overrides a `dbEnvVars` value in the *<TIBHAWK_AMI>* element. For details on environment variable support and usage, see the JDBC driver documentation.
<TIBHAWK_AMI>	`traceFile`	Optional. The fully-qualified path and name of a log file for this microagent. For example,

Element Name	Description	
		`/tmp/queryma.log`. If specified, it overrides the `traceFile` value in the <*TIBHAWK_AMI*> element.
	`traceFileMaxSize`	Optional. The maximum size, in Kbytes, an adapter log file can reach before another log file is started. The default value is 1024 Kb. If specified, overrides the `traceFileMaxSize` value in the <*TIBHAWK_AMI*> element.
	`traceFileMaxNumber`	Optional. The maximum number of log files to keep in the trace directory. After this number of log files is reached, the first file is overwritten. The default value is 4. If specified, overrides the `traceFileMaxSize` value in the <*TIBHAWK_AMI*> element.
	`traceLevel`	Optional. The level of detail to include in the adapter log file. A bit mask, by adding together one or more of the following values: 0—(default) Logs no debug information 1—INFO Logs information messages 2—WARNING Logs warning messages 4—ERROR Logs error information 8—DEBUG Logs debug information 16—AMI Logs AMI debug information 32—DB Logs SQL statements executed by the database A value of `-1` logs all levels of information.
	`maxThreads`	Optional. The maximum number of threads a microagent can use to perform parallel method invocations. The default value is 1. Each thread uses

Element Name	Description
	a separate database connection.
	Contained Elements: *<method>*

The following table describes the ***<method>*** element, which defines methods for a microagent. Methods are used to perform database tasks, such as executing a query or calling a stored procedure.

method Element Definition

Element Name	Description	
<method>	This element defines a method for the current microagent.	
	Attribute Name	**Description**
	`name`	The name of this method.
	`help`	Optional. A help string that describes the method. If no value is specified, the `name` value for this element is used.
	`index`	Optional. The name of an output parameter that uniquely identifies each row. Must be specified for methods that return more than one row of results. If the index uses more than one column, separate the column names with commas.
	`timeout`	Optional. The maximum number of seconds to wait for method results after invocation. The default value is 10. If results are not returned in this interval, an error is generated.
<method>	`handlerType`	The type of database operation this method performs. The valid values are:

Element Name	Description	
		Q: Use for queries and for executing stored procedures (using `exec`) that return data through SELECT statements. For stored procedures that use `call`, use the `P` option. **QD:** Reserved for the `dynamicQuery()` method. **U:** Use for all types of SQL statements except SELECT. An optional output parameter can return the number of rows updated. **UD:** Reserved for the `dynamicExecute()` method. **I:** Use for insert methods. No `SQL` attribute value is required, but the `tableName` attribute must be specified. **P:** Use for stored procedure or function calls (using `call`) that return no data, or return data through output parameters. **PR:** Use for stored procedures or function calls (using `call`) that return data through SELECT statements. **S:** Reserved for the `shutdown` method.
	SQL	The SQL statement to execute when this method is invoked. The valid values are either a correct and complete SQL statement, or `file:`<*script*> where *script* is the name of a file containing the SQL statements. If a full path to the script is not specified, and no directory is specified in the `sqlFileDir` attribute of the `TIBHAWK_AMI` element, the directory where the adapter is installed is used. The SQL statements can contain input and output parameters (except where noted). Parameters should be referenced as ${<*parameter_name*>}.

Element Name	Description	
		Required for all non-reserved `handlerType` values except `I`.
<method>	`tableName`	The database table where values should be inserted. Use only for `handlerType I`.
	`resultSetIndex`	Optional. A positive integer that identifies the result set corresponding to method output parameters. For example, if output parameters correspond to the second SELECT statement that returns data, set this value to `2`.
	`useRowNumberAsIndex`	Optional. The number of the row returned to use as the `index` value. Reserved for `handlerType` values `QD`, and `UD`.
	`useCache`	Optional. Specifies whether to store the SQL statement for this method in cache. Use only for frequently invoked methods, since an open cursor is kept for each cached statement. The valid values are `true` and `false`.
	`logReturnValue`	Optional. Specifies whether output parameter values returned by a stored procedure or function call are written to the adapter log file. Use only for `handlerType P`. The valid values are `true` and `false`.
	Contained Elements: *<inputParameter>, <outputParameter>*	

The following table describes the ***<inputParameter>*** element, which defines input parameters for the SQL statement executed by the current method. For each parameter, you can define either one or more acceptable choices in a `valueChoices` element, or one or more legal choices in a `legalValueChoices` element. If one of these elements is included in the parameter definition, users can select method argument values from a drop down list in Hawk Console. If neither is included, the corresponding method argument is represented by an editable field.

inputParameter Element Definition

Element Name	Description	
<inputParameter>	This element defines the SQL statement input parameters.	
	Attribute Name	**Description**
	`name`	The name of the parameter.
	`help`	Optional. A help string that describes the parameter. If no value is specified, the `name` value of this element is used.
	`type`	The type of data the parameter accepts. Types are defined by the `valueType` entity in the adapter DTD.
	`dateFormat`	For parameters of type `TIMESTAMP` only. If specified, applies only to the current parameter, and overrides a `dateFormat` value specified in the *<TIBHAWK_AMI>* element.
	`pattern`	Specifies a `java.text.MessageFormat` pattern for converting input text data to another form. For more information, see Java Pattern Matching Syntax.
	Contained Elements:*<valueChoices>* or *<legalValueChoices>*	

The following table describes the ***<valueChoices>*** element, which defines some acceptable values for the ***<inputParameter>*** element. These values are included in the drop-down list for method arguments in Hawk Console. Users can also type a value not in the list. This element is optional, and can only be used if ***<legalValueChoices>*** is not specified.

valueChoices Element Definition

Element Name	Description
<valueChoices>	This element defines a set of acceptable input values for input parameters. Other values can be specified.

Element Name	Description	
	Attribute Name	**Description**
	`value`	One or more acceptable input values. Separate multiple values with commas.
	Contained Elements: None	

The following table describes the ***<legalValueChoices>*** element, which defines the only possible values that can be specified in the ***<inputParameter>*** method. This element is optional, and can only be used if ***<valueChoices>*** is not specified. Use this element to limit users to specific values, which are included in the drop-down list for method arguments in TIBCO Hawk Console. You can only specify values included in the list.

legalValueChoices Element Definition

Element Name	Description	
<legalValueChoices>	This element defines a set of legal input values for input parameters. Only defined values can be specified.	
	Attribute Name	**Description**
	`value`	One or more legal input values. Separate multiple values with commas.
	Contained Elements: None	

The following table describes the ***<outputParameter>*** element, which defines output parameters for the SQL statement executed by the current method.

outputParameter Element Definition

Element Name	Description	
<outputParameter>	This element defines the SQL statement output parameters.	
	Attribute Name	**Description**
	`name`	The name of the parameter.
	`help`	Optional. A help string that describes the parameter. If no value is specified, the `name` value of this element is used.
	`type`	The type of data the parameter returns. Types are defined by the `valueType` entity in the adapter DTD.
	`dateFormat`	For parameters of type `TIMESTAMP` only. If specified, applies only to the current parameter, overriding a `dateFormat` value specified in the *<TIBHAWK_AMI>* element.
	Contained Elements: None	

Validating the XML

The adapter includes an external DTD file, `dbhma.dtd`. During installation, this file is copied to the directory where the adapter is installed. The DTD defines valid syntax for all elements and attributes in the adapter configuration file.

Since this information is stored in a separate file, you can use a validating XML parser to check the syntax after editing the file. The parser applies the syntax rules specified in the DTD to the file contents. Validating the XML is an optional step that helps prevent errors resulting from invalid syntax.

To validate XML syntax in the adapter configuration file:

1. Using any text editor, open the file.
2. Change the following line:

```
<!-- <!DOCTYPE TIBHAWK_AMI SYSTEM "<HAWK_DBHMA_
HOME>/xml/dbhma.dtd"> -->
```

to:

```
<!DOCTYPE TIBHAWK_AMI SYSTEM "<HAWK_DBHMA_HOME>/xml/dbhma.dtd">
```

by deleting the comment characters.

3. Save and close the file.
4. Pass the file name to any available validating XML parser.

After validating the configuration file, you should restore the deleted comment characters and save the file. If the line that references the adapter DTD is active, the adapter does not start.

Performing XML Consistency Checks

The adapter also includes a utility for checking the consistency of XML syntax. This utility applies adapter-specific logic to the contents of the configuration file, providing more advanced syntax validation than a generic parser.

Checking for consistency detects whether the internal syntax dependencies are met. For example, the `SQL` attribute of the `method` element is either required or optional, depending on the value of the `handlerType` attribute. Since the `SQL` attribute is not required for all methods, it is defined as optional in the adapter DTD. The consistency checker verifies that this attribute is specified for particular `handlerType` values that require it.

- To check XML consistency, add the <*install_dir*>`/bin` directory to your PATH environment variable and execute the following at a command prompt:

  On Microsoft Windows:

  ```
  tibhawkdbhma --name <servicename> -xml_file <filename> -check_only
  ```

  On UNIX:

  ```
  tibhawkdbhma -xml_file <filename> -check_only
  ```

Where, <*filename*> is the complete path to the configuration XML file and <*servicename*> is the Windows service name given to this instance of the adapter.

> **Note:** The <*servicename*> for the default service installed is <*database*>DBHma, where <database> is either oracle, sybase, or sqlserver.

When this option is used, starting the adapter checks the XML syntax only. It does not communicate with the local Hawk agent or connect to the database. Any syntax errors are displayed in the console window where the command was run. The adapter process stops automatically after checking completes.

Implementing Security

You can increase the level of security for the adapter by doing one or more of the following:

- Securing the adapter configuration file
- Restricting database access for the adapter
- Restricting database access for specific methods

Permissions on the adapter configuration file should be limited, since it contains database connection information that could be misused. When started, the adapter must have read access to the configuration file. Write access is needed only to modify adapter configuration.

To limit adapter access to the database, specify a restricted database account for the `dbUser` attribute of either the `TIBHAWK_AMI` element or the `microagent` element. The adapter uses this account to log into the database when it is started. The adapter inherits all the restrictions to database objects and operations specified for this account.

Using a similar mechanism, you can limit the adapter methods that can be successfully invoked. To restrict access at the microagent level, use the `dbUser` attribute of the `microagent` element to specify a restricted database user account. The value of this attribute overrides the `dbUser` attribute of the `TIBHAWK_AMI` element. If the specified account either lacks access to database objects used by the method, or rights to perform the operation, method invocation fails. No changes are written to the database, and no data is returned.

To build a more secure adapter environment that includes authentication, authorization ,and encryption, implement a TIBCO Hawk security policy. For more information, see *TIBCO® Operational Intelligence Hawk® RedTail Programmer's Guide*.

Sending a Dynamic SQL Statement

This section is a tutorial that shows how to dynamically execute SQL statements using the `dynamicQuery()` and `dynamicExecute()` methods. The `dynamicQuery()` method is designed to execute a query statement in the same manner as an interactive SQL tool. The `dynamicExecute()` method is for executing SQL statements that return no data. These methods are provided as convenient tools for executing SQL statements without formally defining methods in the adapter configuration file. They are designed to be invoked interactively, rather than used as the data source of a rule in a Hawk rulebase.

Complete definitions of `dynamicQuery()` and `dynamicExecute()` are provided in the sample adapter configuration file. To use one or both methods, either use the sample file or copy the definition from the sample file to your configuration file.

> **Note:** For `dynamicQuery()`, the `handlerType` value of the `method` element is `QD`. For `dynamicExecute()`, the value is `UD`. Both values are reserved, and should not be used for custom methods.

Do not modify the `dynamicQuery()` and `dynamicExecute()` definitions. To perform more complicated or customized tasks, build custom methods. For more information, see Building Custom Methods.

Invoking the dynamicQuery() Method

When you start the adapter using the sample configuration file, a microagent for the adapter is dynamically added to the local agent. The adapter microagent is named `dbhma.<database>`, and its methods are described in detail in the following sections.

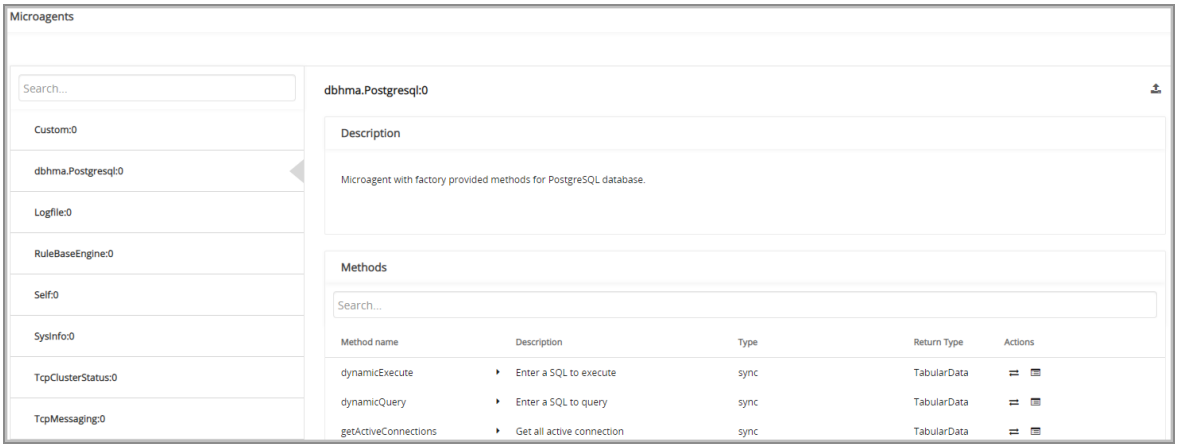
The following steps describe how to invoke the `dynamicQuery()` method of the adapter microagent. This method is a template for executing queries against a relational database. The database accessed by this method is determined by `dbUser` attribute values, specified in the adapter configuration file.

A single string argument, the SQL statement for the query, is required. Up to fifteen columns of results can be returned. For a full method description, see dbhma:dynamicQuery.

To invoke the `dynamicQuery()` method on a TIBCO Hawk agent:

1. Click an agent listed in the **Agents** portlet on the Hawk Dashboard. The **Agent** tab opens. The name of the **Agent** tab is same as the selected agent name *<agent_name>*. The following tabs are available on the top right corner of the screen:
   - Dashboard (Default)
   - Alerts
   - Rulebase
   - Schedules
   - Microagents.

   If Hawk security is implemented on your system and you do not have access to microagents on this agent, an error dialog is displayed. Select another agent, or contact your system administrator to obtain access.
2. Click the **Microagents** tab. The Microagent Info dialog is displayed. The panel on the left lists microagents you can access on the current agent.



   This dialog has two modes, **Invoke** and **Subscribe**. Invoking a method immediately returns a single set of current results. Subscribing provides updates of the current results at regular intervals. Buttons in the actions column of the dialog control these modes.
3. Click the microagent name, in this case **dbhma.Postgresql:0**, to display a list of associated methods and text descriptions in the following panels. For detailed

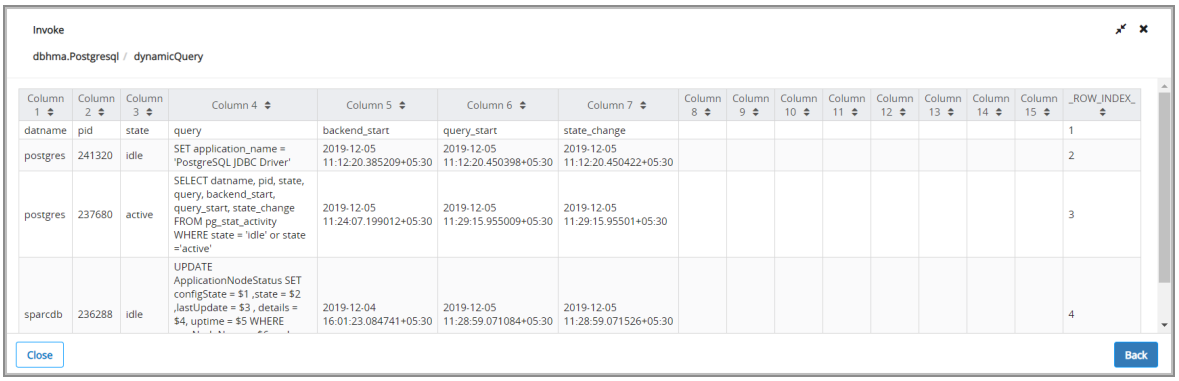
descriptions of these methods, see dbhma:dynamicQuery.

4. In the Methods panel, select **dynamicQuery()** as the method to invoke.
5. Click **Invoke** to execute the query.
6. In the Arguments panel, type the **SQL** statement for the query in the text field. For example, in PostgreSQL:

```
SELECT datname, pid, state, query, backend_start, query_
start, state_change FROM pg_stat_activity WHERE state =
'idle' or  state ='active';
```

This statement returns database name, process ID, state of the query, time when the client application connected to the database server, time at which query execution was initiated, and time when the state of the query was changed. No statement delimiter character is required. Only one SQL statement can be executed per method invocation.

The Invocation Results dialog displays method results, the database rows returned by the query:



Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11	Column 12	Column 13	Column 14	Column 15	_ROW_INDEX_
datname	pid	state	query	backend_start	query_start	state_change									1
postgres	241320	idle	SET application_name = 'PostgreSQL JDBC Driver'	2019-12-05 11:12:20.385209+05:30	2019-12-05 11:12:20.450398+05:30	2019-12-05 11:12:20.450422+05:30									2
postgres	237680	active	SELECT datname, pid, state, query, backend_start, query_start, state_change FROM pg_stat_activity WHERE state = 'idle' or state ='active'	2019-12-05 11:24:07.199012+05:30	2019-12-05 11:29:15.955009+05:30	2019-12-05 11:29:15.95501+05:30									3
sparcdb	236288	idle	UPDATE ApplicationNodeStatus SET configState = $1 ,state = $2 ,lastUpdate = $3 , details = $4, uptime = $5 WHERE	2019-12-04 16:01:23.084741+05:30	2019-12-05 11:28:59.071084+05:30	2019-12-05 11:28:59.071526+05:30									4

7. Click **Close** to close the dialog.

These steps describe how to interactively invoke the `dynamicQuery()` microagent method and view the results in TIBCO Hawk Console. You can also use the `dynamicExecute()` method to execute an SQL statement that returns no data, such as a DDL statement or stored procedure. For more information on invoking Hawk methods, see *TIBCO® Operational Intelligence Hawk® RedTail Administrator's Guide*.

Invoking the dynamicExecute Method

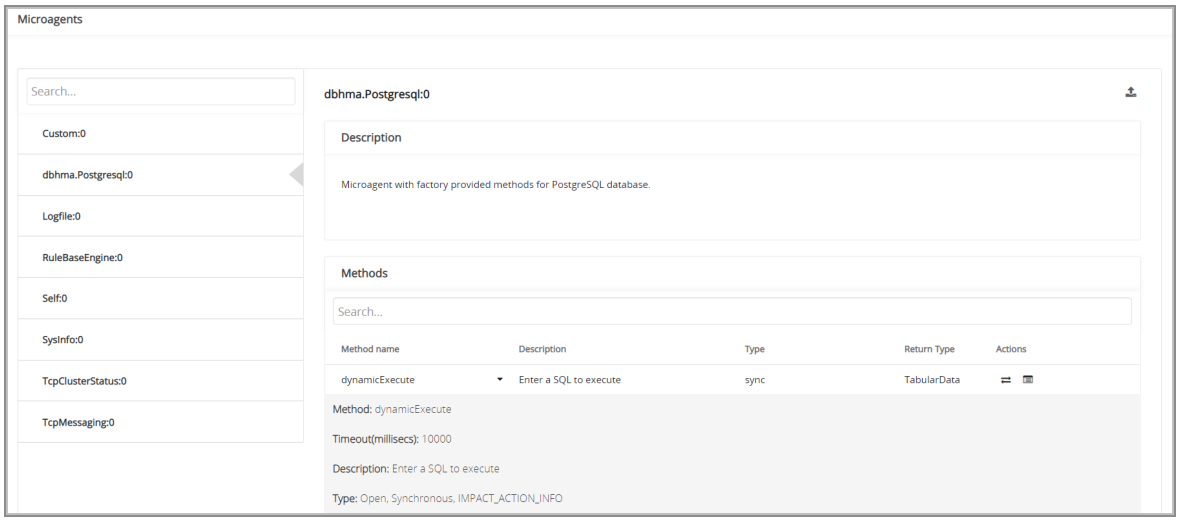
The following steps describe how to invoke the `dynamicExecute()` method of the adapter microagent. This method is a template for SQL statements that perform a task but do not return data. The database accessed by this method is determined by `dbUser` attribute values specified in the adapter configuration file.

A single string argument, the SQL statement to execute, is required. For a full method description, see dbhma:dynamicExecute.

To invoke a microagent method on a Hawk agent:

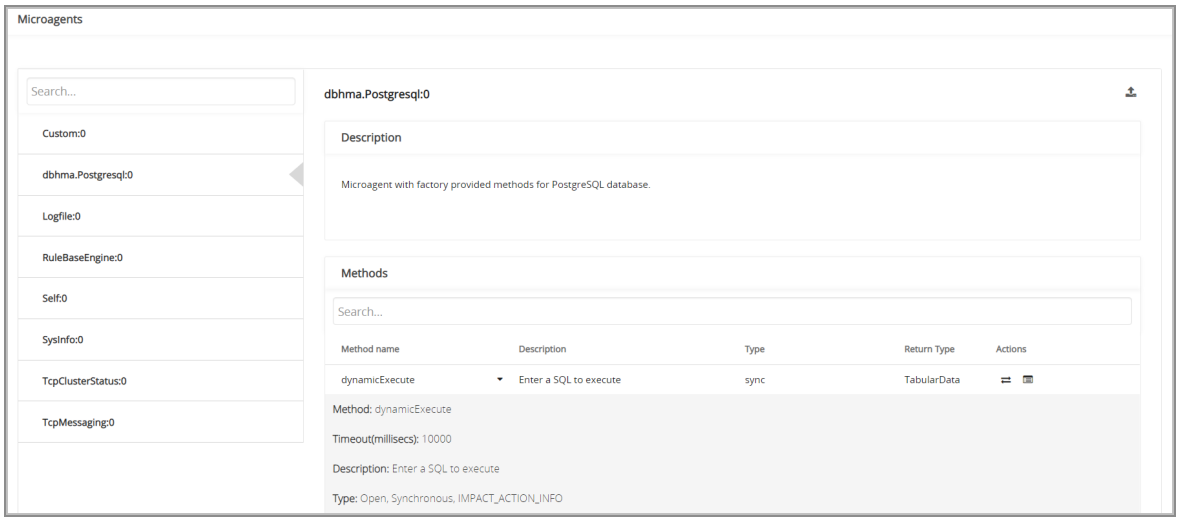
1. Click an agent listed in the **Agents** portlet on the Hawk Dashboard. The **Agent** tab opens. The name of the **Agent** tab is same as the selected agent name *<agent_name>*. The following tabs are available on the top right corner of the screen:
    - Dashboard (Default)
    - Alerts
    - Rulebase
    - Schedules
    - Microagents.

    If TIBCO Hawk security is implemented on your system and you do not have access to microagents on this agent, an error dialog is displayed. Select another agent, or contact your system administrator to obtain access.
2. Click the **Microagents** tab. The Microagent Info dialog is displayed. The panel on the left lists microagents you can access on the current agent.



This dialog has two modes, **Invoke** and **Subscribe**. Invoking a method immediately returns a single set of current results. Subscribing provides updates of current results at regular intervals. Radio buttons at the top of the dialog control these modes.

3. Click the microagent name, in this case **dbhma.Postgresql:0**, to display a list of associated methods and text descriptions in the following panels. For detailed descriptions of these methods, see dbhma.Postgresql.
4. In the Methods panel, select **dynamicExecute()** as the method to invoke.

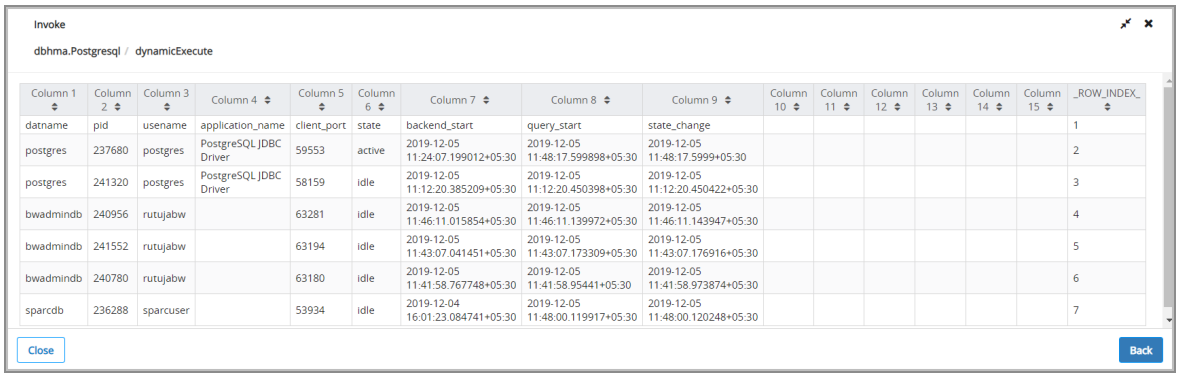


5. Click **Invoke** to execute the SQL statement.
6. In the Arguments panel, type the **SQL** statement to execute in the text field. For example, in PostgreSQL:

```
SELECT datname, pid, usename, application_name, client_port, state,
backend_start, query_start, state_change FROM pg_stat_activity
WHERE state = 'idle' or  state ='active';
```

If the database user specified in the adapter configuration file does not have sufficient privileges for this operation, a warning is displayed.

The Invocation Results dialog displays the results returned by the method.



Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11	Column 12	Column 13	Column 14	Column 15	_ROW_INDEX_
datname	pid	usename	application_name	client_port	state	backend_start	query_start	state_change							1
postgres	237680	postgres	PostgreSQL JDBC Driver	59553	active	2019-12-05 11:24:07.199012+05:30	2019-12-05 11:48:17.599898+05:30	2019-12-05 11:48:17.5999+05:30							2
postgres	241320	postgres	PostgreSQL JDBC Driver	58159	idle	2019-12-05 11:12:20.385209+05:30	2019-12-05 11:12:20.450398+05:30	2019-12-05 11:12:20.450422+05:30							3
bwadmindb	240956	rutujabw		63281	idle	2019-12-05 11:46:11.015854+05:30	2019-12-05 11:46:11.139972+05:30	2019-12-05 11:46:11.143947+05:30							4
bwadmindb	241552	rutujabw		63194	idle	2019-12-05 11:43:07.041451+05:30	2019-12-05 11:43:07.173309+05:30	2019-12-05 11:43:07.176916+05:30							5
bwadmindb	240780	rutujabw		63180	idle	2019-12-05 11:41:58.767748+05:30	2019-12-05 11:41:58.95441+05:30	2019-12-05 11:41:58.973874+05:30							6
sparcdb	236288	sparcuser		53934	idle	2019-12-04 16:01:23.084741+05:30	2019-12-05 11:48:00.119917+05:30	2019-12-05 11:48:00.120248+05:30							7

Like `dynamicExecute()`, this method returns fifteen columns. However, only one column is used. The first row always contains the column name `Update Count`. The second row contains the process ID of rows that were updated by this statement execution.

These steps describe how to interactively invoke the `dynamicExecute()` microagent method and view the results in TIBCO Hawk Console. You can also use the `dynamicQuery()` method to execute a SQL statement returns data. For more information on invoking TIBCO Hawk methods, see *TIBCO® Operational Intelligence Hawk® RedTail Administrator's Guide*.

Viewing the Adapter Log File

The adapter log file contains status information specific to the adapter. The location of this file depends on the value of the `traceFile` attribute in the adapter configuration file. By default, minimal information is written to the log file. The following lines are typical of a log file with the default setting:

```
2000/04/25 13:40:12.107  DBHma started with configuration file:
c:\tibco\hawk\adapters\db\oracle\examples\sample1.xml
```

```
2000/04/25 13:40:12.848  DBHma is ready...
```

```
2000/04/25 17:54:00.664  DBHma Shutting down...
```

You can change the level of information logged in the file using the `traceLevel` attribute for the adapter or the `traceLevel` attribute for a microagent. Any values specified at the microagent level override the adapter settings for that microagent. However, it is recommended to use a single log file per adapter.

You can also modify trace settings by invoking the standard `setTraceLevel()` method.

In addition to the debug information and SQL statements, some types of custom methods can write a return value to the adapter log file.

Using TIBCO Hawk Database Adapter Methods

The adapter is represented by microagents that communicate with Hawk agents. These microagents provide methods for monitoring and managing database servers, applications, or the microagents themselves. This section describes these methods, which you can invoke interactively or in a Hawk rulebase.

Database Independent Methods

The following database independent methods are exposed for monitoring and managing the adapter.

Method	Description
dbhma:dynamicQuery	Executes the specified SQL statement that returns data, typically a SELECT statement or syntax for calling a stored procedure that returns data.
dbhma:dynamicExecute	Executes the specified SQL statement that returns no data, typically an INSERT, UPDATE or DELETE statement or syntax for calling a stored procedure that does not return data.

Method	Description
dbhma:ping	Checks the database connection.
dbhma:shutdown	Stops the adapter gracefully.
dbhma:getTraceLevel	Gets the trace level settings.
dbhma:setTraceLevel	Sets the trace level settings.
dbhma:getTraceParameters	Gets trace parameter values.
dbhma:setTraceParameters	Sets trace parameter values.
dbhma:getMaxThreads	Gets the maximum thread values.
dbhma:setMaxThreads	Sets the maximum thread values.
dbhma:getReleaseVersion	Gets adapter version information.
dbhma:_onUnsolicitedMsg	Returns any unsolicited notifications sent to the database server.

dbhma:dynamicQuery

Executes the specified query.

Type

SYNC, IMPACT_INFO

Arguments

Name	Type	Description
SQL	String	A SQL statement that returns no data, such as an INSERT, UPDATE, or

Name	Type	Description
		DELETE statement, or syntax for calling a stored procedure that does not return data.

Returns

Name	Type	Description
Column *n*	String	One or more columns with any values returned by the query, where *n* is an integer from 1 through 15.
ROW INDEX	String	The accumulated row number for each result.

Usage Notes

Invoking this method executes the specified query against the database instance used by the adapter. This method is not designed to be used as the data source of a Hawk rule. To use a query as a data source, build a custom query method. For more information, see Building Custom Methods.

This method does not enforce read-only access to data, since executing stored procedures is supported. To restrict database access, configure the adapter to use a restricted user account. For more information, see Implementing Security.

You can also limit access to this method by implementing TIBCO Hawk security classes. For more information, see TIBCO Hawk® documentation.

To execute statements that do not return data, see dbhma:dynamicExecute.

dbhma:dynamicExecute

Executes the specified SQL statement.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

Name	Type	Description
SQL	String	A SQL statement that returns no data, such as an INSERT, UPDATE, or DELETE statement, or syntax for calling a stored procedure that does not return data.

Returns

Name	Type	Description
Column *n*	String	One or more columns with any values returned by the SQL statement, where *n* is an integer from 1 through 15.
ROW INDEX	String	The accumulated row number for each result.

Usage Notes

Invoking this method executes the specified statement against the database instance used by the adapter. This method is designed to be interactively invoked, not used as the data source of a Hawk rule. To use a statement execution as a data source, build a custom method. For more information, see Building Custom Methods.

This method provides read and write access to data, but you can restrict database access by configuring the adapter to use a restricted user account. For more information, see Implementing Security.

You can also limit access to this method by implementing TIBCO Hawk security classes. For more information, see TIBCO Hawk® documentation.

To execute a SQL statement that returns data, see dbhma:dynamicQuery.

dbhma:ping

Checks the database connection.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

Name	Type	Description
User Name	String	Database connection user name.
Password	String	Database connection password (plain text or encrypted).
Connection URL	String	Database connection URL
Driver Class	String	Database JDBC Driver. If not specified, the default JDBC driver for this adapter is used.

Returns

Name	Type	Description
Result	String	Information trace setting.
Details	String	Warning trace setting.

Usage Notes

If no input arguments are provided, the default database connection configuration parameters are used.

dbhma:shutdown

Stops the adapter.

Type

SYNC, IMPACT_ACTION

Arguments

None.

Returns

None.

Usage Notes

If the adapter process is running, invoking this method stops the adapter gracefully. All database connections are properly terminated before the adapter is stopped.

dbhma:getTraceLevel

Gets the current trace level settings.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Information	String	Information trace setting.

Name	Type	Description
Warning	String	Warning trace setting.
Error	String	Error trace setting.
Debug	String	Adapter debug trace setting. This option is for troubleshooting purposes only, and under normal circumstances should not be enabled.
AMI	String	AMI debug trace setting. This option is for troubleshooting purposes only, and under normal circumstances should not be enabled.
DB	String	Database-related debug tracing. This option is for troubleshooting purposes only, and under normal circumstances should not be enabled.

Usage Notes

This method returns the level of information written to the adapter log file. For information on trace level usage, see the TIBCO Hawk® documentation.

This standard method is not defined in the adapter configuration file.

dbhma:setTraceLevel

Sets the trace level settings.

Type

SYNC, ACTION_INFO

Arguments

Name	Type	Description
Information	String	Information trace setting.
Warning	String	Warning trace setting.
Error	String	Error trace setting.
Debug	String	Adapter debug trace setting. This option is for troubleshooting purposes only, and under normal circumstances should not be enabled.
AMI	String	AMI debug trace setting. This option is for troubleshooting purposes only, and under normal circumstances should not be enabled.
DB	String	Database-related debug tracing. This option is for troubleshooting purposes only, and under normal circumstances should not be enabled.

Usage Notes

This method determines the level of information written to the adapter log file. Specifying the string `current` preserves the current setting. For information on trace level usage, see the TIBCO Hawk documentation.

This standard method is not defined in the adapter configuration file.

dbhma:getTraceParameters

Gets trace parameter values.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Directory	String	Directory path for the trace log file. The current directory is the default value.
File Name	String	Trace file name.
Max File Size	Integer	Trace file maximum size before it is rolled over, in Kbytes.
Max Trace File	Integer	Maximum number of trace files to keep in the trace directory.

Usage Notes

This method returns log file parameter values. This standard method is not defined in the adapter configuration file.

dbhma:setTraceParameters

Sets trace parameter values.

Type

SYNC, ACTION_INFO

Arguments

Name	Type	Description
Directory	String	Directory path for the trace log file. The current directory is the default value.
File Name	String	Trace file name.
Max File Size	Integer	Trace file maximum size before it is rolled over, in Kbytes.
Max Trace File	Integer	Maximum number of trace files to keep in the trace directory.

Usage Notes

This method modifies log file parameter values. To keep the current setting for string parameters, use the string `current`. For integer parameters, use `0`.

This standard method is not defined in the adapter configuration file.

dbhma:getMaxThreads

Returns the maximum number of AMI invocation handling threads for the adapter.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Maximum Threads	Integer	The maximum number of AMI invocation handling threads.

Usage Notes

This method returns the number of method invocations that can run in parallel. If more methods are simultaneously invoked than the `Maximum Threads` value, extra methods are delayed. This value should be high enough to accommodate typical usage.

This method is available only if a `maxThreads` value is specified for a microagent in the adapter configuration file.

This standard method is not defined in the adapter configuration file.

dbhma:setMaxThreads

Sets the maximum number of AMI invocation handling threads for this session of the adapter.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

Name	Type	Description
Maximum Threads	Integer	The maximum number of AMI invocation handling threads.

Returns

None.

Usage Notes

This method sets the number of method invocations that can run in parallel. If more methods are simultaneously invoked than the `Maximum Threads` value, extra methods are delayed. This value should be high enough to accommodate typical usage.

This method is available only if a `maxThreads` value is specified for a microagent in the adapter configuration file.

dbhma:getReleaseVersion

Gets version information for the adapter.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
Name	String	The names of adapter components. Possible values are: `TIBCO Hawk AMI Java API`, `TIBCO Hawk Database Adapter`, and the fully qualified name of the adapter configuration file.
Version	String	The release version of the component, for example, `1.1`. For the adapter configuration file, this is the value of the `xml_file_version` attribute in the file.
Date	Integer	The date and time this component was last modified.
Major	Integer	The release version major number.
Minor	Integer	The release version minor number.

Usage Notes

This method retrieves the release version of the current adapter. This standard method is not defined in the adapter configuration file.

dbhma:_onUnsolicitedMsg

Returns any unsolicited notifications the adapter sends to the Hawk agent.

Type

ASYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
TYPE	String	The notification type. Must be one of the following: INFO, WARNING or ERROR.
TEXT	String	The text description of the message.
ID	Integer	A user defined number that is typically used to assign a unique ID to each message. This field makes monitoring notifications more efficient. This field must exist in the message even if its value has no significance.
INBOX	String	The inbox address of the adapter AMI interface. Used to identify the source of this notification.

Usage Notes

Invoking this method returns the last unsolicited notification message that was received (if any). Subscribing to the method receives all incoming unsolicited messages.

This standard method is not defined in the adapter configuration file.

dbhma.Oracle

The adapter `Oracle` microagent exposes the following methods for monitoring an Oracle database.

Method	Description
Oracle:getVersion	Gets versions of Oracle software components.
Oracle:getLicenseUsage	Gets license usage information.
Oracle:getInitParameters	Gets the values of `init.ora` parameters.
Oracle:getSGAInfo	Gets the size of each System Global Area (SGA) memory structure.
Oracle:getMemoryUsage	Gets the current status of each database memory buffer.
Oracle:getSpaceUsage	Gets tablespace usage information.
Oracle:getIOInfo	Gets information on datafile I/O activity.
Oracle:getRBSegActivity	Gets current activities of each rollback segment.
Oracle:getRedoLogInfo	Gets redo log file information.
Oracle:getBufHitRatio	Gets read hit ratio of database block buffers.
Oracle:getLibCacheHitRatio	Gets library cache statistics.
Oracle:getDictHitRatio	Get dictionary cache statistics.

Method	Description
Oracle:getHeavySQL	Gets SQL statements that cause heavy disk access.
Oracle:getBlockedUser	Gets names of users that are currently blocked.
Oracle:getLockingUser	Gets names of users currently blocking other users.
Oracle:getCurrUsedObj	Gets names of database objects currently being accessed.
Oracle:getOpenCursor	Gets open cursor information for database users.
Oracle:getUserSession	Gets detailed information on all current user sessions.
Oracle:getSessionCount	Gets the number of current database sessions.
Oracle:getSessionIO	Gets input and output activity information for all sessions.
Oracle:getSessionStat	Gets detailed statistics for each session.
Oracle:getSessionSQL	Gets currently executing SQL statement for all sessions.

Oracle:getVersion

Gets versions of Oracle software components.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Banner	String	Version information for installed Oracle product components.

Usage Notes

Invoking this method returns the banner information that displays when you start up an Oracle database instance. The banner lists each Oracle software component and the version that is installed.

Oracle:getLicenseUsage

Gets license usage information for the database.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
SESSIONS_MAX	Integer	The maximum number of concurrent user sessions allowed for the instance. This is the value of the `license_max_sessions` parameter in init.ora, and is controlled by license. A value of 0 means that the parameter `license_max_sessions` is not set and no limit exists.
SESSIONS_	Integer	The warning limit for concurrent user sessions for the instance.

Name	Type	Description
WARNING		This is the value of the `license_sessions_warning` parameter in init.ora, and is controlled by license. A value of 0 means that the `license_sessions_warning` is not set and no warning message is displayed.
SESSIONS_ CURRENT	Integer	The current number of concurrent user sessions.
SESSIONS_ HIGHWATER	Integer	The highest number of concurrent user sessions since the instance started.
USERS_MAX	Integer	The maximum number of named users allowed for the database. This is the value of the `license_max_users` parameter in `init.ora`, and is controlled by license. A value of 0 means that the `init.ora` parameter is not set and no limit exists.

Usage Notes

Invoking this method returns information about the current number of sessions compared to the maximum allowed sessions. This is useful for determining whether more or fewer database licenses are needed. By using this method in a Hawk rulebase, you can regularly evaluate session usage and the proper licensing level.

Oracle:getInitParameters

Gets the values of `init.ora` parameters.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Name	String	The name of the `init.ora` parameter. Table is indexed on the Name.
Value	String	The current parameter value.

Usage Notes

Invoking this method returns parameter names and values stored in the `init.ora` file. It is useful for verifying configuration settings for database sessions. Invoke this method only when needed.

Oracle:getSGAInfo

Gets the size of each System Global Area (SGA) memory structure.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Name	String	The name of the SGA memory structure: `Fixed Size`, `Variable Size`, `Database Buffers` or `Redo Buffers`. Table is indexed on the Name.
Value	Integer	The memory size, in bytes.

Usage Notes

Invoking this method displays summarized information for the SGA memory structures of your system. `Database Buffers` is the value of the init.ora parameter `db_block_buffers` multiplied by the `db_block_size` value, and `Redo Buffers` is the `log_buffer` value.

Oracle:getMemoryUsage

Gets memory usage information for the database.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Pool	String	The pool the memory structure belongs to. Table is indexed on the Pool and Name.
Name	String	The SGA component name. Table is indexed on the Name.
Bytes	Integer	The memory size, in bytes.

Usage Notes

Invoking this method returns current usage information for memory structures.

Oracle:getSpaceUsage

Gets tablespace usage information for the database.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
File ID	Integer	The identifier of the datafile for the tablespace. Table is indexed on the File ID.
Tablespace	String	The name of the tablespace.
Total	Long	The total number of bytes allocated for each tablespace.
Used	Long	The number of used bytes for each tablespace.
Free	Long	The number of free bytes for each tablespace.
% Used	Integer	The percentage of bytes allocated that are used for each tablespace. This value is `Used/Total`.

Usage Notes

Invoking this method returns disk usage statistics for all tablespaces. This is useful for determining whether the size of a tablespace is approaching its allocated disk space. By using this method in a Hawk rulebase, you can regularly evaluate tablespace usage for the database.

Oracle:getIOInfo

Gets information on physical disk read and write operations.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Name	String	The name of the datafile. Table is indexed on the Name.
Status	String	Type of file (system or user) and its status. Possible values are OFFLINE, SYSOFF, ONLINE, SYSTEM and RECOVER.
Bytes	Integer	The size of this datafile, in bytes.
phyrds	Integer	The total number of physical read operations performed on this datafile.
phyblkrd	Integer	The total number of physical blocks read from this datafile.
phywrts	Integer	The total number of physical write operations performed on this datafile.
phyblkwrt	Integer	The total number of physical blocks written to this datafile.

Usage Notes

This method returns information on disk input and output operations for all datafiles in the database. It is useful for comparing estimates used to initially configure data and index structures with actual usage activity. By comparing the distribution of `phyrds` and `phywrts` values, you can evaluate the configuration of your system.

By using this method in a Hawk rulebase, you can regularly evaluate your physical database configuration. You should also invoke this method after loading large amounts of data and after adding tables or indexes.

Oracle:getRBSegActivity

Gets activity information for each rollback segment.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Segment Name	String	The name of the rollback segment. Table is indexed on the Segment Name.
xacts	Integer	The number of active transactions for the rollback segment.
SID	Integer	The session identifier.
Serial #	Integer	The session serial number.
Username	String	The name of the user accessing the rollback segment.
SQL Text	String	The text of the SQL statement accessing the rollback segment.

Usage Notes

Invoking this method returns SQL statements currently accessing each rollback segment. If system performance is slower than normal, you can use this method to view DML operations taking place. This method is also useful for identifying runaway processes or statements that require termination. Using the `SID` and `Serial #` values, you can terminate any database operation with an `ALTER SYSTEM` command.

Oracle:getRedoLogInfo

Gets the characteristics of each redo log.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Member	String	The redo log member name. This is the full path to the redo log file. *Table is indexed on the Member.
group #	Integer	The redo log group identifier number.
bytes	Integer	The size of the redo log, in bytes.
archived	String	The redo log archive status: TRUE or FALSE.
status	String	The current status of this log member: INVALID, STALE, DELETED or NULL.
Member	String	The redo log member name. This is the full path to the redo log file. *Table is indexed on the Member.

Usage Notes

Use this method to retrieve the characteristics of redo log files. Invoke this method only when needed.

Oracle:getBufHitRatio

Gets the hit ratio of database block buffers.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Hit Ratio	Integer	The number of disk read compared to memory read operations.

Usage Notes

This hit ratio identifies the differences between the number of reads from disk versus memory (block buffers). The buffer hit ratio should be greater than 85 percent for a batch reporting system and 95 percent for an interactive system. If memory exists to increase the database block buffers without affecting system performance, adding more memory to this area increases the hit ratio.

By using this method in a Hawk rulebase, you can regularly evaluate whether the size of database block buffers is sufficient.

Oracle:getLibCacheHitRatio

Gets the hit ratios for database library cache.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
sum(pins)	Integer	The total number of executions against the database library cache.
sum(pinhits)	Integer	The number of database library executions that were fulfilled from cache.
Execution Hit Ratio	Integer	The number of executions fulfilled from cache, `sum(pinhits)`, compared to the total number of executions, `sum(pins)`. Values close to `100` indicate that most database library executions were fulfilled from cache.
sum (reloads)	Integer	The number of times library objects are re initialized and reloaded with data due to age or invalidation.
Reload Hit Ratio	Integer	`sum(reloads) x 100/(sum(pinhits) + sum(reloads))`
sum(pins)	Integer	The total number of executions against the database library cache.

Usage Notes

This method returns an execution (pin) hit ratio and a reload hit ratio. The recommended hit ratio for both is greater than 99 percent. If either of the hit ratio falls below this percentage, the shared pool could be increased to improve overall performance.

By using this method in a Hawk rulebase, you can regularly evaluate whether the size of the shared pool is sufficient.

Oracle:getDictHitRatio

Gets the hit ratios for database dictionary cache.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
sum(gets)	Integer	The total number of requests for information on data objects that were fulfilled using cached information.
sum (getmisses)	Integer	The total number of data requests resulting in cache misses.
Hit Ratio	Integer	The `sum(gets)` value compared to the total number of requests (`(sum(gets) + sum(getmisses)`).

Usage Notes

This method measures the effectiveness of database dictionary cache, a shared pool component. The recommended hit ratio is greater than 90 percent. If the hit ratio falls below this percentage, increasing the shared pool can improve overall performance.

By using this method in a Hawk rulebase, you can regularly evaluate the amount of resources allocated for the shared pool.

Oracle:getHeavySQL

Gets SQL statements causing more than the specified threshold number of disk read operations.

Type

SYNC, IMPACT_INFO

Arguments

Name	Type	Description
disk_reads_threshold	Integer	The threshold value to use for evaluating disk read operations. SQL statements with a `disk_reads` value higher than this value are returned by the method.

Returns

Name	Type	Description
Username	String	The user account used to execute the SQL statement. Table is indexed on the Username.
disk_reads	Integer	The number of disk read operations caused by this SQL statement.
execution	Integer	The total number of times this SQL statement has been executed.
reads_exec Ratio	Integer	The `disk_reads` value compared with `execution` value.
SQL Text	String	The text of the SQL statement.
ROW INDEX	String	The accumulated row number for each result.

Usage Notes

This method returns heavy disk access SQL statements that should be reviewed and optimized to improve overall performance. Typically, statements with a `reads_exec ratio` value greater than 30 should be optimized to use an index.

By using this method in a Hawk rulebase, you can test regularly for statements that are not properly optimized.

Oracle:getBlockedUser

Gets all users and SQL statements that are currently blocked by other users.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Username	String	The name of the user that is currently blocked.
Serial #	Integer	The session serial number. Table is indexed on this.
ID1	Integer	The Oracle locking resource identifier #1
SQL Text	String	The text of the blocked SQL statement.

Usage Notes

Invoking this method returns information about all user operations that are currently blocked because they require resources locked by another user. This is useful for determining the source of a lockout. If an Oracle operation is blocked by another user, invoke `getLockingUser` to identify the blocking user and SQL statement.

Currently, the timeout value for this method is set to 30 seconds. You may adjust this value by changing the `timeout` attribute in the configuration file.

Oracle:getLockingUser

Gets all users executing SQL statements that are currently locking out other users.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Username	String	The name of the locking user.
SID	Integer	The database session identifier. Table is indexed on this.
Serial #	Integer	The session serial number.
ID1	Integer	The locked resource identifier #1.
SQL Text	String	The text of the locking SQL statement.

Usage Notes

Invoking this method returns information about all user operations that are currently locking resources required by other users. This is useful for deciding how a lockout should be resolved. You can identify users and statements that are blocked by invoking `getBlockedUser`. Using the `SID` and `Serial #` values, you can terminate the database session that is locking resources.

Oracle:getCurrUsedObj

Gets a list of database objects that are currently being accessed.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Username	String	The name of the user currently accessing the object.
SID	Integer	The database session identifier. Table is indexed on the SID, Object, and Object Type.
Owner	String	The name of the object owner.
Object	String	The name of the object being accessed. Table is indexed on the SID, Object, and Object Type.
Object Type	String	The type of object being accessed. Table is indexed on the SID, Object, and Object Type.

Usage Notes

Invoking this method returns a list of all database objects that are currently being accessed. The list includes tables, synonyms, views, stored source code, and other types of objects. If an application needs to be modified, you can use this method to determine if related objects are currently in use.

By using this method in a TIBCO Hawk rulebase, you can test for object usage and generate an alert message that notifies users of any changes.

Oracle:getOpenCursor

Gets information about all cursors currently opened by users.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Username	String	The name of the user with the open cursor. Table indexed on Username and SQL Text.
SQL Text	String	The text of the SQL statement using the open cursor. Table indexed on Username and SQL Text.
Count	Long	Instance Count

Usage Notes

This method reports on cursors currently open and the SQL statements used to open cursors on your system. It allows you to determine if cursors are being properly closed. If a particular user has a large number of open cursors, an application or application module might be failing to close cursors when an operation completes.

By using this method in a Hawk rulebase, you can test regularly for applications that are not properly coded.

Oracle:getUserSession

Gets information about current user sessions.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
SID	Integer	The database session identifier. Table is indexed on the SID.
Username	String	The name of the user.
Command	Integer	The code number of the command currently being executed. A value of `0` means this session is a system session. For descriptions of other values, refer to your Oracle documentation.
Status	String	The status of the session. Possible values are ACTIVE, INACTIVE, or KILLED.
Process ID	Integer	The operating system process identifier (PID).
Program	String	The operating system program name.

Usage Notes

This method provides basic information about current database sessions. Invoke this method only when needed.

Oracle:getSessionCount

Gets the number of sessions for each user.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Username	String	The name of the user. Table is indexed on the Username.
Count	Integer	The number of active database sessions for the user.

Usage Notes

Invoke this method to determine the number of active database sessions for each user. If a particular user has a large number of sessions executing, system performance could be affected.

By using this method in a Hawk rulebase, you can regularly check the number of sessions per user.

Oracle:getSessionIO

Gets disk input and output information for each user session.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
SID	Integer	The database session identifier. Table is indexed on the SID.

Name	Type	Description
Username	String	The name of the user executing the session.
block_gets	Integer	The number of blocks retrieved during the session.
consistent_gets	Integer	The number of consistent get operations performed during the session.
physical_reads	Integer	The number of physical read operations performed during the session.
block_changes	Integer	The number of block change operations performed during the session.
consistent_ changes	Integer	The number of consistent change operations performed during the session.

Usage

This method returns information on the physical disk and memory hits for each user session. Invoke this method to identify sessions that perform a large number of physical disk or memory reads. This is useful for determining the system resource requirements of new application modules or other significant upgrades. After a session is identified, invoking the `getSessionSQL` method returns the SQL statement being executed by each session. By using this method in a Hawk rulebase, you can test regularly to determine whether the performance of some applications should be improved.

Oracle:getSessionStat

Gets session statistics for current user sessions.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
SID	Integer	The database session identifier. Table is indexed on the SID and Statistic Name.
Username	String	The name of the user. A value of NULL represents an Oracle background process.
Statistic Name	String	The name of the session statistic. Table is indexed on the SID and Statistic Name.
Value	Integer	The statistic value.
consistent_ changes	Integer	The number of consistent change operations performed during the session.

Usage Notes

Invoking this method returns detailed statistics for all current user sessions. This is useful for identifying resource issues for new application modules or other significant upgrades.

Oracle:getSessionSQL

Gets the currently executing SQL statement for all sessions.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
SID	Integer	The database session identifier. Table is indexed on the SID and SQL Text.
Username	String	The name of the user.
SQL Text	String	The text of each SQL statement currently executed during the session. If the text exceeds the length of a single record, multiple records are used. Table is indexed on the SID and SQL Text.

Usage Notes

This method returns all SQL statements each session is executing at the time the method is invoked. It is useful for obtaining a quick snapshot of database activity.

dbhma.SQLServer

The adapter `SQLServer` microagent exposes the following methods for monitoring a Microsoft SQL Server database.

Method	Description
SQLServer:getServerInfo	Get the current Server configuration.
SQLServer:getConfigInfo	Gets the current configuration parameters.
SQLServer:getDBInfo	Get information about all databases
SQLServer:getDBSpaceUsage	Gets database space usage information
SQLServer:getDefaultDBSpaceUsage	Gets default database space usage information
SQLServer:getDBLogInfo	Provide statistics about the use of transaction log

Method	Description
	space in all databases.
SQLServer:getTableSpaceUsage	Gets database table space used
SQLServer:getTableStatistics	Displays the current distribution statistics for the specified target on the specified table.
SQLServer:getServerCpuUsage	Gets database server CPU usage information
SQLServer:getServerTrafficInfo	Get database server traffic information
SQLServer:getServerIOInfo	Get database server I/O information
SQLServer:getLockInfo	Get user lock information
SQLServer:getUserProcesses	Get user process information

SQLServer:getServerInfo

Gets the server information.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Attribute Id	Integer	Attribute Id.
Attribute Name	String	Attribute Name. Table is indexed on the Attribute Name.
Attribute Value	String	Attribute Value

Usage Notes

This method returns a list of attribute names and matching values for the SQL Server settings.

SQLServer:getConfigInfo

Gets the current server configuration information.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
Name	String	Configuration parameter name. Table is indexed on the Name.
Minimum	Integer	Minimum value

Name	Type	Description
Maximum	Integer	Maximum value
Config Value	Integer	The value to which the configuration parameter has been set with sp_configure.
Run Value	Integer	The value that the server is currently using.

Usage Notes

This method returns the global configuration settings for the current server. Refer to the system stored procedure `sp_configure` for details.

SQLServer:getDBInfo

Gets information about all databases.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
Name	String	The database name.
DB size	String	The database size.
Owner	String	The database owner.

Name	Type	Description
DB ID	Integer	The database ID. Table is indexed on the DB ID.
Created	String	The date the database was created.
Status	String	The current database status.
Compatibility Level	Integer	The compatibility level.

Usage Notes

This method returns information about all databases on the server. The `DB ID` parameter returned by this method is used in a number of SQLServer stored procedures as an argument.

Refer to the system stored procedure `sp_helpdb` for details.

SQLServer:getDBSpaceUsage

Gets the database space usage information.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

Name	Type	Description
Database Name	String	Database name. Cannot be blank.

Returns

Name	Type	Description
Name	String	The logical file name for this fragments
File ID	Integer	The numeric file ID for this fragment.
File Name	String	The physical filename for this fragment.
File Group	String	The file group this logical file belongs to.
Size	Long	The file size in KB on this fragment.
Max Size	String	The maximum size of this fragment.
Growth	String	The growth increment of this file in percentage.
Usage	String	The usage of this file.
_ROW_INDEX	Long	Accumulated row number.

Usage Notes

This method reports information about the specified database. The Group indicates the group to which the file belongs. The grouping is for allocation and administration purposes only. The Max Size indicates the amount of space added to the file each time new space is needed. The Growth increment indicates the amount of space added to the file each time new space is needed.

Refer to the system stored procedure sp_helpdb <*databasename*> for details.

SQLServer:getDefaultDBSpaceUsage

Gets default database space usage information.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
Reserved	Integer	The reserved space size in KBytes.
Data	Integer	The space used for data in KBytes.
Index_size	Integer	The space used for indexes in KBytes.
Unused	Integer	The unused space in KBytes.

Usage Notes

This method returns the disk space reserved and used for data and indexes by the entire database. This method is particularly useful because it also reports on the unused space.

Refer to the system stored procedure `sp_spaceused` for details.

SQLServer:getDBLogInfo

Provides statistics about the use of transaction-log space in all databases.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Database Name	String	Name of the database for the log statistics displayed. Table is indexed on the database name.
Log Size	Double	The actual amount of space available for the log. This amount is smaller than the amount originally allocated for log space because Microsoft SQL Server reserves a small amount of disk space for internal header information.
Log Space Used (%)	Double	Percentage of the log file currently occupied with transaction log information
Status	Integer	The status of the log file (always contains a 0)

Usage Notes

This method returns the disk space reserved and used for data and indexes by the entire database.

By using this method in a Hawk rulebase you can periodically monitor the computer running the database Server to confirm there is enough space remaining for transactional logs.

Refer to the `DBCC SQLPERF (logspace)` statement for details.

SQLServer:getTableSpaceUsage

Gets information about the database table space used.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

Name	Type	Description
Table Name	Integer	Table name. Cannot be blank.

Returns

Name	Type	Description
Table name	String	Name of the table.
Total Rows	Integer	The total number of rows in the table.
Reserved	Integer	The reserved space size in KBytes.
Data Size	Integer	The space used for data in KBytes.
Index Size	Integer	The space used for indexes in KBytes.
Unused	Integer	The unused space in KBytes.

Usage Notes

This method reports the amount of space allocated (reserved) for the specified table, the amount used for data, the amount used for index(es), and the unused space reserved by database objects.

By using this method in a Hawk rulebase you can periodically monitor the table to confirm that space usage is within normal ranges.

Refer to the system stored procedure `sp_spaceused` *<tablename>* for details.

SQLServer:getTableStatistics

Gets information about the database table space used.

Type

SYNC, IMPACT_INFO

Arguments

Name	Type	Description
Table	String	Table name. Cannot be blank.
Target	String	The name of the object (index name or collection) for which to display statistics information. Target names must conform to the rules for identifiers. If the target is both an index name and a statistics collection name, both index and column statistics are returned. If no index or statistics collection is found with the specified name, an error is returned. Cannot be blank.

Returns

Name	Type	Description
Updated	String	Date and time the last statistics were updated.
Rows	Integer	The number of rows in the table.
Rows Sampled	Integer	The number of rows sampled for statistics information.
Steps	Integer	The number of distribution steps.
Density	Double	The selectivity of the index.
Average Key Length	Double	The average length of the row.

Usage Notes

The results returned indicate the selectivity of an index (the lower the density returned, the higher the selectivity) and provide the basis for determining whether or not an index is

useful to the optimizer. The results returned are based on distribution steps of the index. By using this method in a Hawk rulebase you can periodically monitor the table to confirm that space usage is appropriate.
Refer to the `SHOW_STATISTICS` DBCC statement for details.

SQLServer:getServerCpuUsage

Gets information about the database server CPU usage.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
CPU Busy	Integer	The % of time the CPU was doing server work since the last time this method was run.
IO Busy	Integer	The % of time the CPU was doing input/output work since the last time this method was run.
Idle	Integer	The % of time the CPU was idle since the last time this method was run.

Usage Notes

SQL Server keeps track, through a series of functions, of how much work it has done. The percentage in this output is based on the time that CPU was allocated to Server, it is not a percentage of the total sample interval. Invoking this method, returns the CPU usage statistics computed by these functions and shows how much they have changed since the last time the method was invoked.

Refer to the system stored procedure `sp_monitor` for details.

SQLServer:getServerTrafficInfo

Gets information about packet transfer.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
Packet Received	Integer	The number of input packets received by the database server.
Packet Sent	Integer	The number of output packets written by the database server.
Packet Errors	Integer	The number of errors encountered by the database while reading and writing packets.

Usage Notes

SQL Server keeps track, through a series of functions, of the number of packets received, sent and those in error. Invoking this method, returns the packet statistics computed by these functions and shows how much they have changed since the last time the method was invoked.

Refer to the system stored procedure `sp_monitor` for details.

SQLServer:getServerIOInfo

Gets information about the database server I/O.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
Total Read	Integer	The number of reads by the database server since the last time this method was run.
Total Write	Integer	The number of writes by the database server since the last time this method was run.
Total_errors	Integer	The number of errors encountered by the database server reading and writing since the last time this method was run.
Connections	Integer	The number of login attempts to the database server since the last time this method was run.

Usage Notes

SQL Server keeps track, through a series of functions, of the number of physical disk and memory hits performed by the SQLServer. Invoking this method, read and write activity computed by these functions and show how much they have changed since the last time the method was invoked.

By using this method in a Hawk rulebase, you can test regularly to determine whether the performance of some applications should be improved.

Refer to the system stored procedure `sp_monitor` for details.

SQLServer:getLockInfo

Gets information about locks.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
(Table is indexed on the spid, dbid, Objid, and IndId.)		
spid	Integer	The user process ID in the database server.
dbid	Integer	The database ID in the database server
ObjId	Integer	The object ID in the database server.
IndId	Integer	The index ID in the database server.
Type	String	The type of the resource currently locked
Resource	String	The locked resource
Mode	String	The type of lock being applied to the resource.
Status	String	The lock status

Usage Notes

This method returns snapshot information about locks. This information returned by this method is helpful in investigating contention issues caused by concurrent users and applications using a database.

Refer to the system stored procedure `sp_lock`for details.

SQLServer:getUserProcesses

Gets user process information.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
ecid	Integer	The execution context ID in the database server. Table is indexed on the spid.
Status	String	The current status of the user process.
Login Name	String	The login name of the user process.
Host	String	The host name of the computer where the user process is run.
blk	String	The process ID of the blocking process which was blocking this process.
Database Name	String	The database name.

Usage Notes

This method reports on the current users processes running on the database server. The ecid column contains the execution context identifier. The `blk` column contains the process

IDs of the blocking process, if there is one. A blocking process (which may be infected or have an exclusive lock) is one that is holding resources that another process needs. This method is useful for obtaining a quick snapshot of the database activity. This method can be used to determine if a process is being blocked, and by whom. This method is also used to see all the active and waiting processes.

Refer to the system stored procedure `sp_who` for details.

dbhma.Sybase

The adapter `Sybase` microagent exposes the following methods for monitoring a Sybase SQL Server database.

Method	Description
Sybase:getServerInfo	Get Server Information
Sybase:getConfigInfo	Gets the current server configuration information.
Sybase:getDBInfo	Gets information about all databases.
Sybase:getDBSpaceUsage	Gets the database space usage information.
Sybase:getDefaultDBSpaceUsage	Gets default database usage information.
Sybase:getDefaultDBLogUsage	Gets default database transactional log row count.
Sybase:getTableSpaceUsage	Gets information about the database table space used.
Sybase:getServerCpuUsage	Gets information about the database server CPU usage.
Sybase:getServerTrafficInfo	Gets database server traffic information.
Sybase:getServerIOInfo	Gets information about the database server I/O.
Sybase:getLockInfo	Gets information about user locks.
Sybase:getUserProcesses	Gets user statistics on system usage.

Method	Description
Sybase:getUserStats	Gets user process information.

Sybase:getServerInfo

Gets the Server information.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Name	String	Name. Table is indexed on the Attribute Name.
Comment	String	Attribute Value
Config Value	String	Configuration value

Usage Notes

This method returns the current server settings.

Sybase:getConfigInfo

Gets the current Server configuration information.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
Name	String	Configuration parameter name. Table is indexed on the Name.
Default	Integer	Default.
Memory Use	Integer	Memory use.
Config Value	Integer	The value to which the configuration parameter has been set with sp_configure.
Run Value	Integer	The value that the server is currently using.
Unit	String	Unit.
Type	String	Type.

Usage Notes

This method returns all configuration parameters by group, their current values, their default values, the value (if applicable) to which they have most recently been set, and the amount of memory used by this setting.

Refer to the system stored procedure `sp_configure` for details.

Sybase:getDBInfo

Gets information about all databases.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
Name	String	The database name.
DB size	String	The database size in MB.
Owner	String	The database owner.
DB ID	Integer	The database ID. Table is indexed on the DB ID.
Created	String	The date the database was created.
Status	String	The current database status.

Usage Notes

Invoking this method returns a complete list of the database options. The `DB ID` parameter returned by this method is used in a number of system procedures as an argument.

Refer to the system stored procedure `sp_helpdb` for details.

Sybase:getDBSpaceUsage

Gets database space usage information.

Type

SYNC, IMPACT_ACTION_INFO

Arguments.

Name	Type	Description
Database Name	String	Database name. Cannot be blank.

Returns

Name	Type	Description
Device Fragments	String	Device Fragments.
Size	Integer	The size in MB of this fragment.
Usage	String	The usage type.
Created	String	Date the database was created.
Free KBytes	Integer	The current free space in KBytes.
_ROW_INDEX*	Long	Accumulated row index. Table is indexed on the _ROW_ INDEX.

Usage Notes

This method reads stored values to provide a quick report on the space used by a database. This method is particularly useful because it also reports on the unused space.

Refer to the system stored procedure `sp_helpdb` *<databasename>* for details.

Sybase:getDefaultDBSpaceUsage

Gets default database space usage information.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
Reserved	Integer	The reserved space size in KBytes.
Data	Integer	The space used for data in KBytes.
Index_size	Integer	The space used for indexes in KBytes.
Unused	Integer	The unused space in KBytes.

Usage Notes

This method reads stored values to provide a quick report on the default space used by a database. This method is particularly useful because it also reports on the unused space.

Refer to the system stored procedure `sp_spaceused` for details.

Sybase:getDefaultDBLogUsage

Gets default database transaction log row count.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
Total Rows	Integer	The total number of rows.

Usage Notes

This method returns the number of rows used by the transactional logs.

Sybase:getTableSpaceUsage

Gets information about the database table space used.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

Name	Type	Description
Table Name	Integer	Table Name. Cannot be blank.

Returns

Name	Type	Description
Table name	String	Name of the table.
Total Rows	Integer	The total number of rows in the table.
Reserved	Integer	The reserved space size in KBytes.
Data Size	Integer	The space used for data in KBytes.
Index Size	Integer	The space used for indexes in KBytes.
Unused	Integer	The unused space in KBytes.

Usage Notes

This method reads stored values to provide a quick report on the space used by an object. This method is particularly useful because it also reports on the unused space.

By using this method in a Hawk rulebase you can periodically monitor the table to confirm that space usage is within normal ranges.

Refer to the system stored procedure `sp_spaceused` *<tablename>* for details.

Sybase:getServerCpuUsage

Gets information about the database server CPU usage.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
CPU Busy	Integer	The % of time the CPU was doing server work since the last time this method was run.
IO Busy	Integer	The % of time the CPU was doing input/output work since the last time this method was run.
Idle	Integer	The % of time the CPU was idle since the last time this method was run.

Usage Notes

This method returns information about how busy the CPU was during the sample period. The percentage in this output is based on the time that CPU was allocated to Server, it is not a percentage of the total sample interval. Invoking this method, returns the CPU usage statistics computed by these functions and shows how much they have changed since the last time the method was invoked.

Refer to the system stored procedure `sp_monitor` for details.

Sybase:getServerTrafficInfo

Gets database server traffic information.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
Packet Received	Integer	The number of input packets received by the database server.
Packet Sent	Integer	The number of output packets written by the database server.
Packet Errors	Integer	The number of errors encountered by the database while reading and writing packets.

Usage Notes

This server keeps track, through a series of functions, of the number of packets received, sent and those in error. Invoking this method, returns the packet statistics computed by these functions and shows how much they have changed since the last time the method was invoked.

Refer to the system stored procedure `sp_monitor` for details.

Sybase:getServerIOInfo

Gets information about the database server input and output operations.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
Total Read	Integer	The number of reads by the database server since the last time this method was run.
Total Write	Integer	The number of writes by the database server since the last time this method was run.
Total_errors	Integer	The number of errors encountered by the database server reading and writing since the last time this method was run.
Connections	Integer	The number of login attempts to the database server since the last time this method was run.

Usage Notes

The server keeps track, through a series of functions, of the number of physical disk and memory hits performed by the server. Invoking this method, read and write activity computed by these functions and show how much they have changed since the last time the method was invoked. This method is useful for determining the server system resource requirements.

By using this method in a Hawk rulebase, you can test regularly to determine whether the performance of some applications should be improved.

Refer to the system stored procedure `sp_monitor` for details.

Sybase:getLockInfo

Gets information about user locks.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
(Table is indexed on the spid, table_id, and page.)		
fid	Integer	The file ID.
spid	Integer	The user process ID in the database server.
loid	Integer	The lock owner ID.
locktype	String	The lock type.
table_id	Integer	The table ID in the database server.
page	Integer	The page ID in the database server.
row	Integer	The row number.
dbname	String	The database name
class	String	The lock class
context	String	The lock context.

Usage Notes

This method reports the locks currently being held on the server. Invoking this method allows one to see the current state of the locks on the databases. The `locktype` column indicates not only whether the lock is a shared lock ("Sh" prefix), an exclusive lock ("Ex" prefix), or an "update" lock, but also whether it is held on a table ("table" or "intent") or on a "page."

Refer to the system stored procedure `sp_lock` for details.

Sybase:getUserProcesses

Gets user process information.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
fid	Integer	The file ID.
spid	Integer	The user process ID in the database server. Table is indexed on the spid.
Status	String	The current status of the user process.
Login Name	String	The login name of the user process.
Original Name	String	The original name of the user process.
Host	String	The host name of the computer where the user process is run.
blk	String	(blk_spid) The process ID of the blocking process which was blocking this process.
Database Name	String	The database name.
Command	String	The current command being executed.
blk_xloid	Integer	The lock owner ID of the blocking transaction.

Usage Notes

This method reports on the current user's processes running on the database server. The `spid` column contains the process identification numbers. The `blk` column contains the process IDs of the blocking process, if there is one. A blocking process (which may be infected or have an exclusive lock) is one that is holding resources that another process needs. This method is useful for obtaining a quick snapshot of the database activity. This method can be used to determine if a process is being blocked, and by whom. This method is also used to see all the active and waiting processes or to see all the valid `spids`.

Refer to the system stored procedure `sp_who` for details.

Sybase:getUserStats

Gets user statistics on system usage.

Type

SYNC, IMPACT_ACTION_INFO

Arguments

None.

Returns

Name	Type	Description
Name	String	The user process ID in the database server
Since	String	The time the user logged on to the database.
CPU	Integer	CPU used.
Percent CPU	Double	Percent CPU used
I/O	Integer	I/O used.

Name	Type	Description
Percent I/O	Double	The percent I/O used.

Usage Notes

This method returns basic information about the current users and processes. This is useful for identifying resource issues for new applications modules or other significant upgrades.

Refer to the system stored procedure `sp_reportstats` for details.

dbhma.DB2

The adapter `DB2` microagent exposes the following methods for monitoring an DB2 database.

Method	Description
DB2:getVersionInfo	Retrieve information about the system.
DB2:getBufferPoolHitRatio	Retrieve bufferpool hit ratio information.
DB2:getBufferPoolRead	Retrieve bufferpool read performance information.
DB2:getBufferPoolWrite	Retrieve bufferpool write performance information.
DB2:getConnectedApplications	Retrieve connected database application information.
DB2:getDBInstanceInfo	Retrieve information about the current instance.

Note: The following methods can be used IF-AND-ONLY-IF you have access and privileges to execute SYSPROC database objects.

DB2:getDictionaryInfo	View a report of the dictionary information of tables in the specified schema.

Method	Description
DB2:getIndexInfo	Get Index Information.
DB2:getLockInfo	List all locks in the currently connected database.
DB2:getMemoryUsage	Retrieve memory usage statistics from all partitions in megabyte (MB) values
DB2:getMemoryPoolInfo	Retrieve metrics from the memory pools contained within a memory set
DB2:getSpaceUsage	Get tablespace usage information.
DB2:getTableMetrics	Get license usage information.
DB2:getTableSpaceMetrics	List the activity on all tables accessed since the database was activated, aggregated across all database members, ordered by the highest number of reads.

DB2:getVersionInfo

Retrieve information about the system.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
OS Name	String	Name of the operating system.
OS Version	String	Version number of the operating system.
OS Release	String	Release number of the operating system.
Host Name	String	Host Name
Total CPU	Integer	Total number of physical CPUs on the system.
Configured CPU	Integer	Number of configured physical CPUs on the system.
Total Memory	Integer	Total amount of memory on the system (in megabytes).

Usage Notes

Invoking this method returns system information about each DB2 component and the version installed.

DB2:getBufferPoolHitRatio

Retrieve bufferpool hit ratio information.

Type

SYNC, IMPACT__INFO

Arguments

None.

Returns

Name	Type	Description
Database Name	String	Database Name.
Buffer Pool Name	String	Buffer Pool Name.
Total Hit Ratio	Double	Total hit ratio (index, XDA and data reads).
Data Hit Ratio	Double	Data hit ratio.
Index Hit Ratio	Double	Index hit ratio.
XDA Hit Ratio	Double	Auxiliary storage objects hit ratio.
DB Partition Number	Integer	The database partition from which the data was retrieved for this row.

Usage Notes

Invoking this method you can evaluate the bufferpool hit ratio in terms of Data, Index and XDA.

DB2:getBufferPoolRead

Retrieves bufferpool read performance information.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Buffer Pool Name	String	Buffer Pool Name.
Total Reads	String	Total physical reads.
Avg Read Time (MS)	String	Average read time in milliseconds.
DB Partition Number	String	The database partition from which the data was retrieved for this row.

Usage Notes

Invoking this method summarizes the number of reads performed on the bufferpool.

DB2:getBufferPoolWrite

Retrieves bufferpool write performance information.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Buffer Pool Name	String	Buffer Pool Name.

Name	Type	Description
Total Writes	String	Total writes.
Avg Write Time (MS)	String	Average write time in milliseconds.
DB Partition Number	String	The database partition from which the data was retrieved for this row.

Usage Notes

Invoking this method displays information about the number of writes performed on the bufferpool.

DB2:getConnectedApplications

Retrieves connected database application information.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Database Name	String	Database Name.
Application Name	String	Application Name.

Name	Type	Description
Agent ID	Integer	Application handle (agent ID).
Authorization ID	String	Authorization ID.
Application ID	String	Application ID.
Application Status	String	Application Status is one of: BACKUP, COMMIT_ACT, COMP, CONNECTED, CONNECTPEND, CREATE_DB, DECOUPLED, DISCONNECTPEND, INTR, IOERROR_WAIT, LOAD, LOCKWAIT, QUIESCE_TABLESPACE, RECOMP, REMOTE_RQST, RESTART, RESTORE, ROLLBACK_ACT, ROLLBACK_TO_SAVEPOINT, TEND, THABRT, THCOMT, TPREP, UNLOAD, UOWEXEC, UOWWAIT, WAITFOR_REMOTE.
DB Partition Number	Integer	The database partition from which the data was retrieved for this row.

Usage Notes

Invoking this method displays summarized information about application databases that are currently connected.

DB2:getDBInstanceInfo

Retrieves information about the current instance.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Instance Name	String	Name of the current instance.
Partitionable?	Integer	Whether the current instance is a partitionable database server instance or not.
Partitions	Integer	Number of database partitions.
Bit Size	Integer	Bit size of the current instance (32 or 64).
Release Number	String	Internal release number.
Service Level	String	Service Level.
Build Level	String	Build Level.
PTF ID	String	Program temporary fix (PTF) identifier.
Fix Pack Number	Integer	Fix Pack number.

Usage Notes

This method reports about the currently executing instance.

DB2:getDictionaryInfo

View a report of the dictionary information of tables in the specified schema.

Note: This method can be used IF-AND-ONLY-IF you have access and privileges to execute SYSPROC database objects.

Type

SYNC, IMPACT_INFO

Arguments

Name	Type	Description
Schema Name	String	Schema Name

Returns

Name	Type	Description
Table Name	String	Table Name
Schema Name	String	Schema Name
Table Name	String	Table Name
Object Type	String	The type of the object. Output can be one of the following values: 'XML', 'DATA'
Partition	Big Integer	Database partition number.
Compressed?	String	The state of the COMPRESS attribute on the table which can be one of the following values: 'Y' = Row compression is set to yes, 'N' = Row compression is set to no
Dictionary Size	Big Integer	Total physical size of the dictionary in bytes.
Sampled Row Count	Integer	Number of records that contributed to building the dictionary.
Percent Saving	Integer	Percentage of pages saved from compression.

Name	Type	Description
in Pages		
Percent Saving in Bytes	Integer	Percentage of bytes saved from compression.
Avg Compressed Record	Integer	Average compressed record length of the records contributing to building the dictionary.

Usage Notes

Invoking this method displays summarized information of the dictionary of tables in the database used.

DB2:getIndexInfo

Get Index Information.

Note: This method can be used IF-AND-ONLY-IF you have access and privileges to execute SYSPROC database objects.

Type

SYNC, IMPACT_INFO

Arguments

Name	Type	Description
Schema Name	String	Schema Name

Returns

Name	Type	Description
Table Schema	String	Table Schema
Table Name	String	Table Name
Table Name	String	Table Name
Index Schema	String	Index Schema
Index Name	String	Index Name
Partition	Big Integer	Database partition number.
Index Id	Big Integer	Index identifier.
Data Partition	Big Integer	Data partition identifier.
Compressed?	String	Physical index format. 'Y' = Index is in compressed format, 'N' = Index is in uncompressed format
Partitioned?	String	Identifies the partitioning characteristic of the index. 'N' = Nonpartitioned index, 'P' = Partitioned index, Blank = Index is not on a partitioned table.
Logical Size	Big Integer	Logical size of the index object.
Physical Size	Big Integer	Physical size of the index object.
Rebuild Required?	String	Rebuild status for the index. 'Y' = if the index defined on the table or data partition requires a rebuild, 'N' = otherwise.

Name	Type	Description
Large RIDS present?	String	Indicates whether or not the index is using large row IDs (RIDs) (4 byte page number, 2 byte slot number). 'Y' = indicates that the index is using large RIDs, 'N' = indicates that the index is not using large RIDs, 'P' = (pending) indicates that the table that the index is defined on supports large RIDs (that is, the table is in a large table space), but the index for the table or data partition has not been reorganized or rebuilt yet.

Usage Notes

Invoking this method returns index information about the table schema.

DB2:getLockInfo

List all locks in the currently connected database.

Note: This method can be used IF-AND-ONLY-IF you have access and privileges to execute SYSPROC database objects.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Lock Name	String	Lock Name

Name	Type	Description
Member Id	Integer	Database member from which the data was retrieved for this row.
Lock Mode	String	Lock Mode is one of: G ==> Represents the mode that the application is currently holding the lock in, W or C ==>> Represents the mode that the application is currently waiting to acquire the lock in.
Lock status	String	Lock status.
Application handle	Big Integer	Application handle is one of: G ==> Represents the application that is currently holding the lock, W or C ==> Represents the application that is currently waiting to acquire the lock.
Table Space ID	Big Integer	Table Space ID.

Usage Notes

Invoking this method returns information about all locks that are currently active on the connected database.

DB2:getMemoryUsage

Retrieves memory usage statistics from all partitions in megabyte (MB) values.

Note: This method can be used IF-AND-ONLY-IF you have access and privileges to execute SYSPROC database objects.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Partition		The database partition number from which memory usage statistics is retrieved.
Max Memory (MB)	Big Integer	The maximum amount of instance memory (in bytes) allowed to be consumed in the database partition if an instance memory limit is enforced.
Current Memory (MB)	Big Integer	The amount of instance memory (in bytes) currently consumed in the database partition.
Peak Memory (MB)	Big Integer	The peak or high watermark consumption of instance memory (in bytes) in the database partition.

Usage Notes

Invoking this method returns statistics for the amount of memory used from all partitions.

DB2:getMemoryPoolInfo

Retrieves metrics from the memory pools contained within a memory set.

Note: This method can be used IF-AND-ONLY-IF you have access and privileges to execute SYSPROC database objects.

Type

SYNC, IMPACT_INFO

Arguments

Name	Type	Description
Database Name	String	Database Name.

Returns

Name	Type	Description
Host Name	String	Host Name
Database Name	String	Database Name.
Set Type	String	Set Type is one of: DBMS ==> Database Manager Memory Set, FMP ==> Fenced Mode Process Memory Set, PRIVATE ==> Private Memory Set, DATABASE ==> Database Memory Set, APPLICATION ==> Application Memory Set, FCM ==> Fast Communication Manager Memory Set, NULL ==> All sets at instance and database level.
Pool Type	String	Memory pool type.
Memory Pool Used (Bytes)	Big Integer	Amount of memory pool in use.
High Water Mark (Bytes)	Big Integer	Memory pool high water mark.

Usage Notes

Invoking this method returns information about the memory pool in a memory set.

DB2:getSpaceUsage

Get tablespace usage information.

Note: This method can be used IF-AND-ONLY-IF you have access and privileges to execute SYSPROC database objects.

Type

SYNC, IMPACT_INFO

Arguments

Name	Type	Description
Schema Name	String	Schema Name.
Table Name	String	Table Name

Returns

Name	Type	Description
Schema Name	String	Name of the operating system.
Table Name	String	Version number of the operating system.
Table Type	String	Release number of the operating system.
Partition	String	Host Name
Availability	String	Total number of physical CPUs on the system.
Occupied	String	Number of configured physical CPUs on the system.

Name	Type	Description
Size		
Index Type	Big Integer	Total amount of memory on the system (in megabytes).
Load Status	Integer	Current status of a load operation against the table. The status can be one of the following values: IN_PROGRESS, NULL, PENDING
Dictionary Size	String	Size of the table dictionary, in bytes, used for row compression if a row compression dictionary exists for the table. If a historical dictionary exists, this value is the sum of the current and historical dictionary sizes.

Usage Notes

Invoking this method displays summarized information about tablespace usage with respect to size and other characteristics.

DB2:getTableMetrics

Get license usage information.

Note: This method can be used IF-AND-ONLY-IF you have access and privileges to execute SYSPROC database objects.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Table Schema	String	Schema name.
Table Name	String	Table Name
Table Type	String	Table Type. One of: USER_TABLE, TEMP_TABLE, CATALOG_TABLE
Total Table Scans	Big Integer	Total Table Scans.
Total Rows Read	Big Integer	Total Rows Read.
Total Rows Inserted	Big Integer	Total Rows Inserted.
Total Rows Updated	Big Integer	Total Rows Updated.
Total Rows Deleted	Big Integer	Total Rows Deleted.

Usage Notes

Invoke this method to identify information about various table metrics.

DB2:getTableSpaceMetrics

List the activity on all tables accessed since the database was activated, aggregated across all database members, ordered by highest number of reads.

Note: This method can be used IF-AND-ONLY-IF you have access and privileges to execute SYSPROC database objects.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Table Space Name	Integer	Table space Name.
Table Space Id	Integer	Table space Identifier.
Table Space Type	String	Table space Type.
Page Size	String	Table space page size.
Extent Size	Big Integer	Table space extent size.
Prefetch Size	Big Integer	Table space prefetch size.
Rebalancer Mode	String	Rebalancer mode and is one of: NO_REBAL, FWD_REBAL, REV_REBAL, FWD_REBAL_OF_2PASS, REV_REBAL_OF_2PASS.
Auto Storage?	Big Integer	Table space enabled for automatic storage or not.
Auto Resize?	Big Integer	Table space automatic resizing enabled or not.
Direct Reads	Big Integer	Direct reads from database.

Name	Type	Description
Direct Writes	Big Integer	Direct writes to database.
BufferPool Physical Reads	Big Integer	Buffer pool data physical reads.
BufferPool XDA Physical Reads	Big Integer	Buffer pool XDA data physical reads.
BufferPool Index Physical Reads	Big Integer	Buffer pool index physical reads.
BufferPool Writes	Big Integer	Buffer pool data writes.
BufferPool XDA Physical Writes	Big Integer	Buffer pool XDA data writes.
BufferPool Index Physical Writes	Big Integer	Buffer pool index writes.
Used Pages	Big Integer	Used pages in table space.
Free Pages	Big Integer	Free pages in table space.
Usable Pages	Big Integer	Usable pages in table space.
Total Pages	Big Integer	Total pages in table space.

Usage Notes

Invoking this method, returns results computed on the basis of all activities performed since the database was first activated.

dbhma.Mysql

The adapter `MySQL` microagent exposes the following methods for monitoring an MySQL database.

Method	Description
Mysql:getDatabaseList	Get all database information.
Mysql:getEngineList	Get all engine information.
Mysql:getOpenTableList	Get all open table information.
Mysql:getProcessList	Get all processes information.
Mysql:getServerStatus	Provide information about server operation.
Mysql:getTableList	Get all table information.

Mysql:getDatabaseList

Get all database information.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Database Name	String	The user process ID in database server

Usage Notes

Invoking this method returns the complete information of the database used.

Mysql:getEngineList

Get all engine information.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Engine	String	The engine type in the database server.
Support	String	Is the engine supported? YES ==> The engine is supported and active, DEFAULT ==> Same as YES plus DEFAULT engine, NO ==> The engine is NOT supported, DISABLED ==> The engine is supported but disabled.
Comments	String	Additional Information
Transaction Support	String	Does this engine support transactions?
XA Support	String	Does this engine support multi-phase transactions?
Save Point Support	String	The current command being executed

Usage Notes

Invoking this method returns a complete list of the engine parameters.

Mysql:getOpenTableList

Get all open table information.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Database	String	The database containing the table.
Table	String	The table name
In Use	Integer	The number of table locks or lock requests there are for the table
Locked	Integer	Is this table locked?

Usage Notes

Invoking this method returns a complete list of the options of open tables.

Mysql:getProcessList

Get all process information.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Process Id	String	The user process ID in database server
User	String	The initiator of this process
Host	String	The host name of the machine where the user process is running
DB Name	String	The database name associated with this user process
Command	String	The current command being executed
Execution Time (ms)	Integer	The execution time (ms) of this user process
State	String	The current state of this process
Information	String	Additional Information

Usage Notes

This method reports on the current user's processes running on the database server.

Mysql:getServerStatus

Provide information about server operation.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Variable Name	String	The name of the server variable.
Value	String	The value of server variable

Usage Notes

This method returns information about the operations of the server that is currently running.

Mysql:getTableList

Get all table information.

Type

SYNC, IMPACT_INFO

Arguments

None.

Returns

Name	Type	Description
Table Name	String	The user process ID in the database server.
Type	String	The object type ==> TABLE v/s VIEW.

Usage Notes

Invoking this method displays summarized information of a database table used.

dbhma.Postgresql

The dbhma.`Postgresql` microagent of the adapter exposes the following methods for monitoring a PostgreSQL database.

Method	Description
Postgresql:getDatabaseList	Get a list of database names and database size.
Postgresql:getActivityStatus	Get database activity information.
Postgresql:getUserTableList	Get a list of open tables.
Postgresql:getQueryStatus	Get information about queries running on databases.
Postgresql:getTime	Get information about database start time.
Postgresql:getIndexScan	Get the number of index scans performed on a table.
Postgresql:getSequentialScan	Get the number of sequential scans performed on a table.

Method	Description
Postgresql:getRowsFetchedAndReturned	Get the number of rows fetched versus the number of rows returned by queries to the database.
Postgresql:getTempBytes	Get amount of data written temporarily to disk to execute queries
Postgresql:getRowsStatsPerDB	Get the number of rows inserted, updated, deleted by queries (per database).
Postgresql:getRowsStatsPerTable	Get the number of rows inserted, updated, deleted by queries (per table).
Postgresql:getHeapOnlyTuple	Get Heap-only tuple (HOT) updates.
Postgresql:getTransactionsCount	Get information about the total number of transactions executed (commits + rollbacks).
Postgresql:getDeadlocks	Get the total number of deadlocks.
Postgresql:getDeadRows	Get the number of dead rows.
Postgresql:getCheckpointMetrics	Get information about checkpoint metrics.
Postgresql:getActiveConnections	Get information about active connections.
Postgresql:getDatabaseUsageStats	Get disk and index usage for the database.
Postgresql:getTableUsageStats	Get disk and index usage for a table.

Postgresql:getDatabaseList

Invoking this method displays summarized information about the database currently in use.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
database	VARCHAR	Name of the database
size	VARCHAR	Size of the database

Postgresql:getActivityStatus

Invoking this method displays summarized information about the database activities.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
datname	VARCHAR	Database name specified on the database server

Name	Type	Description
pid	INTEGER	Process ID of the process running in the database server
usename	VARCHAR	User name in the database
Application Name	VARCHAR	Application name connected to the database
Client Port	INTEGER	TCP port number that the client is using for communicating with the database server
state	VARCHAR	Current state of the database
backend_ start	VARCHAR	Time when the client application connected to the database server
query_start	VARCHAR	Time when the query in execution was started or time when the last executed query was started if there is no query being executed
state_change	VARCHAR	Time when the state was last changed

Postgresql:getUserTableList

Invoking this method displays summarized information about currently opened tables of a database.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
schemaname	VARCHAR	Schema name of the database
relname	VARCHAR	Name of the table

Postgresql:getQueryStatus

Invoking this method displays summarized information about the queries running on the database.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
datname	VARCHAR	Name of the database
pid	INTEGER	Process ID of the process running a query on the database server
state	VARCHAR	State of a query that is running on the database
query	VARCHAR	The query running on the database

Name	Type	Description
backend_ start	VARCHAR	Time when the client application connected to the database server
query_start	VARCHAR	Time when the query in execution was started or time when the last executed query was started if there is no query being executed
state_ change	VARCHAR	Time when the state was last changed

Postgresql:getTime

Invoking this method displays summarized information about the database start time.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
datname	VARCHAR	Name of the database
backend_start	VARCHAR	Time when the database was started

Postgresql:getIndexScan

Invoking this method displays summarized information about the index scans performed on a table.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
table_name	VARCHAR	Name of the table
index_scan	INTEGER	Number of index scans performed on a table

Postgresql:getSequentialScan

Invoking this method displays summarized information about the sequential scans performed on a table.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
table_name	VARCHAR	Name of the table
seq_scan	INTEGER	Number of sequential scans performed on a table

Postgresql:getRowsFetchedAndReturned

Invoking this method displays summarized information about the number of rows fetched compared to the number of rows received by queries performed on the database.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
datname	VARCHAR	Name of the database
tup_fetched	INTEGER	Rows fetched by queries on the database
tup_returned	INTEGER	Rows returned by queries on the database

Postgresql:getTempBytes

Invoking this method displays summarized information about the amount of temporary data used by the database to execute queries.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
datname	VARCHAR	Name of the database
temp_bytes	INTEGER	Amount of data written temporarily on a disk to execute queries
stats_reset	VARCHAR	Time at which the statistics were last reset

Postgresql:getRowsStatsPerDB

Invoking this method displays summarized information about the operations performed on the rows of a database.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
datname	VARCHAR	Name of the database
tup_inserted	INTEGER	Number of rows inserted by queries
tup_updated	INTEGER	Number of rows updated by queries
tup_deleted	INTEGER	Number of rows deleted by queries

Postgresql:getRowsStatsPerTable

Invoking this method displays summarized information about the operations performed on the rows of a table in a database.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
relname	VARCHAR	Name of the table
n_tup_ins	INTEGER	Number of rows inserted by queries (per table)
n_tup_upd	INTEGER	Number of rows updated by queries (per table)
n_tup_del	INTEGER	Number of rows deleted by queries (per table)

Postgresql:getHeapOnlyTuple

Invoking this method displays summarized information about the Heap updates performed on a table.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
relname	VARCHAR	Name of the table
n_tup_hot_upd	INTEGER	Heap Only tuple updates

Postgresql:getTransactionsCount

Invoking this method displays summarized information about the transaction queries performed on a database.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
datname	VARCHAR	Name of the database
total	INTEGER	Total number of transactions executed (commits + rollbacks)
stats_reset	VARCHAR	Time at which the statistics were last reset

Postgresql:getDeadlocks

Invoking this method displays summarized information about the deadlocks existing in the database.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
datname	VARCHAR	Name of the database
deadlocks	INTEGER	Total number deadlocks

Postgresql:getDeadRows

Invoking this method displays summarized information about the dead rows existing in the table of a database.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
relname	VARCHAR	Name of the table
n_dead_tup	INTEGER	Number of dead rows

Postgresql:getCheckpointMetrics

Invoking this method displays summarized information about the checkpoint metrics in a database.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
checkpoints_req	INTEGER	Number of checkpoints requested
checkpoints_timed	INTEGER	Number of checkpoints scheduled
buffers_checkpoint	INTEGER	Number of buffers written during checkpoints
buffers_clean	INTEGER	Number of buffers written by the background writer
buffers_backend	INTEGER	Number of buffers written by backends

Postgresql:getActiveConnections

Invoking this method displays summarized information about the active connections in a database.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
datname	VARCHAR	Name of the database
numbackends	INTEGER	Number of active connections

Postgresql:getDatabaseUsageStats

Invoking this method displays summarized information about the disk and index usage of a database.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
datname	VARCHAR	Name of the database
blks_hit	INTEGER	Blocks in the database that were shared buffer hits from disk
blks_read	INTEGER	Blocks in the database that were shared buffer read from disk

Postgresql:getTableUsageStats

Invoking this method displays summarized information about the table usage.

Type

SYNC, IMPACT_INFO

Arguments

None

Returns

Name	Type	Description
relname	VARCHAR	Name of the table
heap_blks_hit	INTEGER	Blocks in the table that were shared buffer hits from disk
heap_blks_read	INTEGER	Blocks in the table that were shared buffer read from disk
idx_blks_hit	INTEGER	Blocks from indexes in the table that were shared buffer hits from disk
idx_blks_read	INTEGER	Blocks from indexes in the table that were shared buffer read from disk

Building Custom Methods

This section explains how to build custom Hawk® Database Adapter methods for executing SQL statements. It describes how to create custom methods for various types of operations, including executing queries, inserting data, and executing stored procedures or functions.

Overview

Since no two database environments are alike. Therefore, the adapter has a flexible design that is easily customized. This section presents strategies for building custom microagent methods, including examples and considerations for the various method types, such as SELECT and INSERT. It explains how to design adapter methods that meet your monitoring, access, and integration requirements.

Define method characteristics by specifying values for XML elements and attributes in the adapter configuration file. A custom method can execute nearly any type of SQL statement, including functions and stored procedures. The SQL to execute can be referenced explicitly in the file, or implicitly in an existing script. For details on the configuration file structure and syntax elements, see Getting Started.

A configuration file must define at least one microagent and method, but a single file can contain as many microagent and method definitions as necessary. All methods can belong to a single microagent, or divided into groups among multiple microagents. The default microagents and methods described in Using TIBCO Hawk Database Adapter Methods have been predefined in the sample configuration file.

After defining methods in the file and starting the adapter, you can invoke methods interactively or use them as either a data source or an action in a rulebase. Any rulebase errors are written to the adapter log file, so you should test custom methods by invoking them interactively before using them in a rulebase. For more information on invoking methods and building a rulebase, see *TIBCO® Operational Intelligence Hawk® RedTail Administrator's Guide*.

Normally, a method is one of three types: IMPACT_ACTION, IMPACT_INFO, or IMPACT_ACTION_INFO. In custom adapter methods, these standard types are not referenced directly. Instead, the method type is embedded in a set of `handlerType` values designed for database operations. The `handlerType` attribute for methods is described in Getting Started.

Building Query Methods

A query method selects rows from one or more database tables. Query methods must have a value of `Q` for the `handlerType` attribute of the `method` element. The query syntax is specified in the `SQL` attribute for the method. This attribute, which is required for query methods, can also reference a script containing query syntax. Input parameters specified in the configuration file become method arguments in the Hawk Console. By building a generic query method and specifying different input parameters for each invocation, a single method can be used for multiple queries against the same table.

To qualify as a query method, the SQL statement must either be a SELECT statement, or a stored procedure executed using the `exec` keyword that returns data through a SELECT statement. The `tableName` attribute is required for query methods. If the table specified by this attribute does not exist, a database error is returned.

The SQL statement can reference input or output parameters. For example, a method that gets information about database tables having extents higher than the specified threshold has the following input and output parameters defined:

```
<inputParameter
  name = "Extent Threshold"
  type = "INTEGER"
>
</inputParameter>
<outputParameter
  name = "Segment"
  type = "VARCHAR"
```

```
    >
    </outputParameter>
    <outputParameter
      name = "Tablespace"
      type = "VARCHAR"
    >
    </outputParameter>
    <outputParameter
      name = "Count"
      type = "INTEGER"
    >
    </outputParameter>
    <outputParameter
      name = "Sysdate"
      type = "TIMESTAMP"
      dateFormat = "MM-dd-yyyy:hh:mm:ss"
    >
    </outputParameter>
```

In TIBCO Hawk Console, Extent Threshold is the name of the method input argument. When the method is invoked, four columns of data are returned: `Segment`, with the name of the segment, Tablespace with the name of the table, `Count` with the number of extents each table segment uses, `Sysdate` with a current timestamp value.

Methods that return more than one row must specify an index column or columns. The `index` attribute of the method specifies a column value that uniquely identifies each row returned. If no single column can act as a unique identifier, two or more columns can act as a composite index. For example, the method in the previous example includes the following attribute:

```
index = "Segment, Tablespace"
```

A custom query method can be invoked interactively or can be the data source of a rule in a rulebase. You can also execute a query using the `dynamicQuery()` method. For more information, see dbhma:dynamicQuery.

Building Insert Methods

An insert method adds a row to a database table. Insert methods are useful for periodically adding new records to gather historical information. For example, a rule in a rulebase could receive method results from a data source, apply a test to check for null values or

other criteria, then pass the results as input parameters to an action. The action might be an insert method invocation that stores the results in a database table.

Each column in the table that receives data requires a corresponding method input parameter defined in the adapter configuration file. The adapter uses the table name and input parameters to build the SQL INSERT statement, so the `SQL` attribute is not required for insert methods. If a row already exists and the table has a unique index, an error is returned. Otherwise a duplicate row is inserted. Insert methods are defined by a value of `I` for the `handlerType` attribute of the `method` element.

Note: You can also add rows to a table by building a custom method with `handlerType=U`. In this case, the `SQL` attribute is required. For more information, see Building Other Execution Methods.

The `tableName` attribute is required for insert methods. If the table does not exist, the adapter creates it before inserting the first row. For input parameters of type CHAR or VARCHAR, VARCHAR(80) columns are created. If this size is too large or too small for the type of value being stored, create the table manually before invoking the method. When storing large amounts of data, create the table manually with exact column sizes and storage parameters before inserting any data.

For this type of method, names of input parameters must match column names in the destination table. For example, a method that inserts a row into the `spaceUsage` table has `tableName` = `spaceUsage` and the following input parameters:

```
<inputParameter
   name="timestamp"
   type="TIMESTAMP">
</inputParameter>

<inputParameter
   name="percent_used"
   type="INTEGER">
</inputParameter>
```

The table in this example has two columns, `timestamp` and `percent_used`. In TIBCO Hawk Console, these input parameters are names of method arguments. When the method is invoked, a row containing the values specified for each argument is inserted.

Note: The adapter does not validate the values specified for method arguments. Method arguments, or input parameters, are passed to the database regardless of content. If an invalid value is specified for the column type, the database returns an error message to the local Hawk agent.

When invoking a method with TIMESTAMP input parameters, you can either specify a value in the correct format or no value. When no value is specified, the adapter inserts the current date and time in the correct format into the timestamp field for that record. However, if the timestamp value to insert was generated by an application, it might be useful to preserve the original value. In this case, you could use the output parameter of the data source as the input parameter for the timestamp column.

Building Other Execution Methods

For database operations other than queries, simple insert operations, or stored procedure executions, specify a value of `U` for the `handlerType` attribute of the `method` element. Use methods with `handlerType` value of `U` for statements that return no data, such as CREATE and UPDATE.

This type of method can include a single output parameter. If defined in the configuration file, this parameter returns a single value for the number of database rows updated, and must be of INTEGER type. For example:

```
<outputParameter
   name="update_count"
   type="INTEGER">
</outputParameter>
```

The update count parameter is optional, and no other output parameters can be defined.

Building Methods with Stored Procedures or Function Calls

A method can execute a function call or stored procedure that has already been defined in the database. Stored procedure and function call methods must have a value of `P` or `PR` specified for the `handlerType` attribute of the `method` element.

- The value P is for procedures or functions that return data through output parameters.
- The value PR is for procedures that return data through SELECT statements.

For these two handlerType values, the SQL attribute must include the call keyword in the syntax for executing the function or procedure. Either method type can include input parameters. Input parameters defined in the adapter configuration file can be used to pass parameters to the stored procedure or function.

Note: It is also possible to execute a stored procedure or function with a query (with handlerType =Q) or an update method (with handlerType =U). In these cases, the keyword exec must be used in the SQL statement and no output parameters are allowed.

For PR methods, the SQL statement must be in the following form:

```
call <procedure_name> (${<input_param_1>}, ...,${<input_param_n>})
```

where input_param_1 to input_param_n are names of input parameters defined in the adapter configuration file. Parameter names in the configuration file do not need to match database function or procedure parameter names. However, for handlerType=PR, the order input parameters defined in the file must match the order they are referenced in the SQL statement. For example, a method for calling the procedure Proc1 has handlerType = PR and the following input parameters:

```
<inputParameter
   name="P1"
   type="VARCHAR">
</inputParameter>

<inputParameter
   name="P2"
   type="INTEGER">
</inputParameter>
```

The following syntax specified in the SQL attribute for this method executes the stored procedure:

```
call Proc1 (${P1}, ${P2})
```

For P methods, the SQL statement must be in one of the following forms:

```
call <procedure_name> (${<input/output_param_1>}, ...,${<input/output_param_n>})
```

or

```
${<output_param_x>}=call <procedure_name> (${<input/output_param_1>}, ...,${<input/output_param_n>})
```

where <*input/output_param*_1> and <*input/output_param_n*> are the names of input or output parameters defined in the adapter configuration file. For `handlerType=P` methods, parameter names in the configuration file do not need to match names defined in function or stored procedure syntax. For example, the procedure `PINOUT` with the following input and output parameters defined:

```
CREATE PROCEDURE PINOUT (x1 in VARCHAR, x2 in out SMALLINT, x3 out INT)
```

has the following input and output parameters defined in the adapter configuration file:

```
<inputParameter
   name="P1"
   type="VARCHAR">
</inputParameter>

<inputParameter
   name="P2"
   type="INTEGER">
</inputParameter>

<outputParameter
   name="P2"
   type="SMALLINT">
</outputParameter>

<outputParameter
   name="P3"
   type="INTEGER">
</outputParameter>

<outputParameter
   name="Return Value"
   type="INTEGER">
</outputParameter>
```

For this method, the syntax specified for the `SQL` attribute might be as follows:

```
${Return Value} = call PINOUT (${P1}, ${P2}, ${P3})
```

In this example, the order that parameters are defined in the adapter configuration file is not required to match the order of parameters in the `SQL` attribute.

For input parameters, the corresponding stored procedure or function parameter must be type `in` or type `in out`. For output parameters, the corresponding parameters must be either type `out` or `in out`. To use `in out` parameters, define an input parameter and an output parameter with the same name in the configuration file.

Note: Some databases do not allow a parameter to be used for both input and output in stored procedures or functions.

Using Input and Output Parameters

A custom method can have input and output parameters defined in the adapter configuration file. Input parameters are the equivalent of TIBCO OI Hawk RedTail arguments, and output parameters are the equivalent of method return fields. You can use parameters to pass values to an SQL statement, stored procedure, or function, and to receive return values for certain method types.

Valid data types for input and output parameters are defined in the `valueType` entity in the adapter DTD. If no type is specified, the VARCHAR data type is used. For more information, see TIBCO Hawk Database Adapter Document Type Definition (DTD).

When you reference parameters in the `SQL` attribute for a method, use the following format:

```
${parameter_name}
```

where `parameter_name` is the name of an input or output parameter defined in the adapter configuration file. This name can be different than the corresponding parameter name defined in the stored procedure or function.

Parameter values with a `type` value of `TIMESTAMP` can be converted to `DATE` or `DATETIME` type using the `dateFormat` attribute. The adapter converts date values retrieved from a database to a character string, using the specified `dateFormat` value. For input parameters, string values that represent dates are converted to the `DATE` or `DATETIME` type before they

are sent to the database. Pattern logic can be used to convert values. For more information on date formats and conversion, see Date and Time Value Syntax.

Using Input Parameters

Custom methods can reference input parameters in the SQL statement. An input parameter is a placeholder for one or more values passed to the SQL statement, procedure or function call. Input parameters must be defined in the adapter configuration file.

For example, the following SQL statement specified for the `SQL` attribute:

```
select first_name from employee where last_name=${Last Name}
```

requires an input parameter with `name="Last Name"` defined in the configuration file. This method is designed so that users can specify a last name as an argument in TIBCO Hawk Console or as a variable in a rulebase. When the method is invoked, the supplied name is used.

Note: Except for methods with a `handlerType` value of `P`, input parameters must be referenced in the SQL statement in the same order they are defined in the adapter configuration file.

You can use the `pattern` attribute to convert input text data into a different format. The syntax for conversion is defined by the `java.text.MessageFormat` class. For example, the pattern `{0,number} KB` converts the string value `123 KB` to the number `123`. The pattern `{0} KB` converts the same value to a string value `123`. For more information, see Java Pattern Matching Syntax.

The definition of an input parameter can include an optional element for defining values. The `valueChoices` element defines a partial list of possible values. When `valueChoices` is defined in the adapter configuration file, the method argument drop-down list in TIBCO Hawk Console contains some valid options. It also includes an editable field for a user to enter other values.

The `legalValueChoices` element defines a complete list of legal values. When you use `legalValueChoices`, the method argument drop-down list in TIBCO Hawk Console contains the only valid options. No other values can be entered. These two elements are mutually exclusive, and if neither is used the method argument field in TIBCO Hawk Console is a blank editable field.

Using Output Parameters

Custom methods can reference output parameters in the SQL statement. An output parameter is a placeholder for one or more values returned by the SQL statement, procedure or function call. Output parameters must be defined in the adapter configuration file, and in most cases, referenced in the SQL statement in the same order they are defined in the file.

Note: The exception to this rule is a `handlerType= P` method. For this type of method, the order in which the output parameters are defined in the file is not significant.

A result set is a group of records returned by a single SELECT statement in the stored procedure. By default, for methods with `handlerType = PR`, output parameters are mapped to the first result set returned. If multiple sets of results are returned, use the `resultSetIndex` attribute of the `method` element to map the desired result set to output parameters. For example, if output parameters for the procedure correspond to rows returned by the second SELECT statement, the value of the `resultSetIndex` attribute for the stored procedure method should be `2`.

Note: Date values retrieved from a database are converted to character strings by the local agent. Once converted, numeric test operators cannot be applied to date values in a rulebase. If you need to perform date comparisons on output parameters, first use a database function to represent the date or date and time as a numeric value.

Logging Return Values

Methods with a value of `P` for the `handlerType` attribute can write the return value of the procedure or function to the adapter log file. The return value is useful for indicating execution success or failure. Log file entries for return values have the following form:

```
<method_name> returns <return_value>
```

To log a return value, the function or procedure must include an output parameter that corresponds to the return value. The value of the `logReturnValue` attribute of the `method` element must also be set to `true`.

A separate entry is created each time the method is invoked. After the entry is created, you can use the `logFile()` microagent to monitor the adapter log file and determine method success or failure.

Note: If the function or procedure returns an integer value, the value must be greater than 0, otherwise no entry is created in the log file.

Using Methods in a Rulebase

This section describes several sample implementations of default and custom adapter methods. An Adapter method can be used as a data source or an action in a rulebase, a collection of user-defined monitoring policies.

Overview

In a TIBCO OI Hawk RedTail environment, rulebases contain the logic used to automate monitoring and management tasks. A rulebase contains one or more rules, user-defined policies that collect information and perform tasks. A rule consists of a data source that is a microagent method, one or more tests that check for specific conditions, and one or more actions to take if a test evaluates to `true`. Rulebases can be distributed to multiple agents on the network, and rulebase activities can be controlled by user-defined schedules.

Adapter methods can be used either as the data source or action of the rule. A method that returns data through a SELECT statement or output parameters can act as the data source. When the rule is active, the local agent invokes the method and passes data returned to the test. The action is the consequence of a rule, and in a database environment; it might be another adapter method executing a DDL statement such as UPDATE or ALTER TABLE, or a SQL script that performs some management task.

The following sections present several practical examples of how to use adapter methods in a rulebase. These examples should be used as templates to help you customize the adapter to your environment. They represent a small fraction of useful database tasks that can be performed using the adapter. Custom adapter methods combined with existing rulebase features can provide nearly unlimited flexibility in rulebase design.

Capturing Historical Data

You can use custom adapter methods to periodically create snapshots of information and store them in a database. The source of information can be any managed object represented by a microagent. Values returned by the microagent method can be inserted into a data store using a custom insert method. When a capture operation is performed regularly by a rulebase, a historical record is created.

This section demonstrates how to build a rule for capturing historical data. The data gathered in this example is tablespace usage information for an Oracle database using the method `Oracle:getSpaceUsage`. For a method description, see Oracle:getSpaceUsage.

The method that records the usage data is a sample insert method, `Record tablespace usage()`, that is defined in the sample configuration file, `sample1.xml`. To use this method, this example starts a new instance of the adapter that uses the sample configuration file. You can also copy the definition from the sample file to your adapter configuration file, then restart the adapter.

Building a Rule for Data Capture

These steps demonstrate how to create rulebase for capturing historical data. The rulebase contains a simple rule that uses the `Oracle:getSpaceUsage()` method as a data source. In this example, a test restricts the scope of data returned by the data source to a single tablespace. At the data collection interval, the agent where the adapter is running retrieves usage data, filters it, and uses a sample insert method to store it in a database.

For instructions on starting TIBCO Hawk Console, see *TIBCO® Operational Intelligence Hawk® RedTail Administrator's Guide*.

To create a new rulebase in Hawk Console:

1. Click an agent listed in the **Agents** portlet on the Hawk Dashboard. The **Agent** tab opens. The name of the **Agent** tab is the same as selected *<agent_name>*. The following tabs are available on the top right corner of the screen:
    - Dashboard (Default)
    - Alerts
    - Rulebase

— Schedules

— Microagents

2. Click the **Rulebase** tab. The Rulebase screen lists the rulebases loaded by the selected agent and provides a toolbar for performing rulebase operations.

   Click the **Create new Rulebase** icon from the toolbar to create a new rulebase. A new rulebase is created with the default name "New-Rulebase". Type **DB_HistData** in the **RuleBase Name** field.

3. Enter the other fields, as required.

4. Click **Create and Add Rule** to save the changes and proceed to the rule building process.

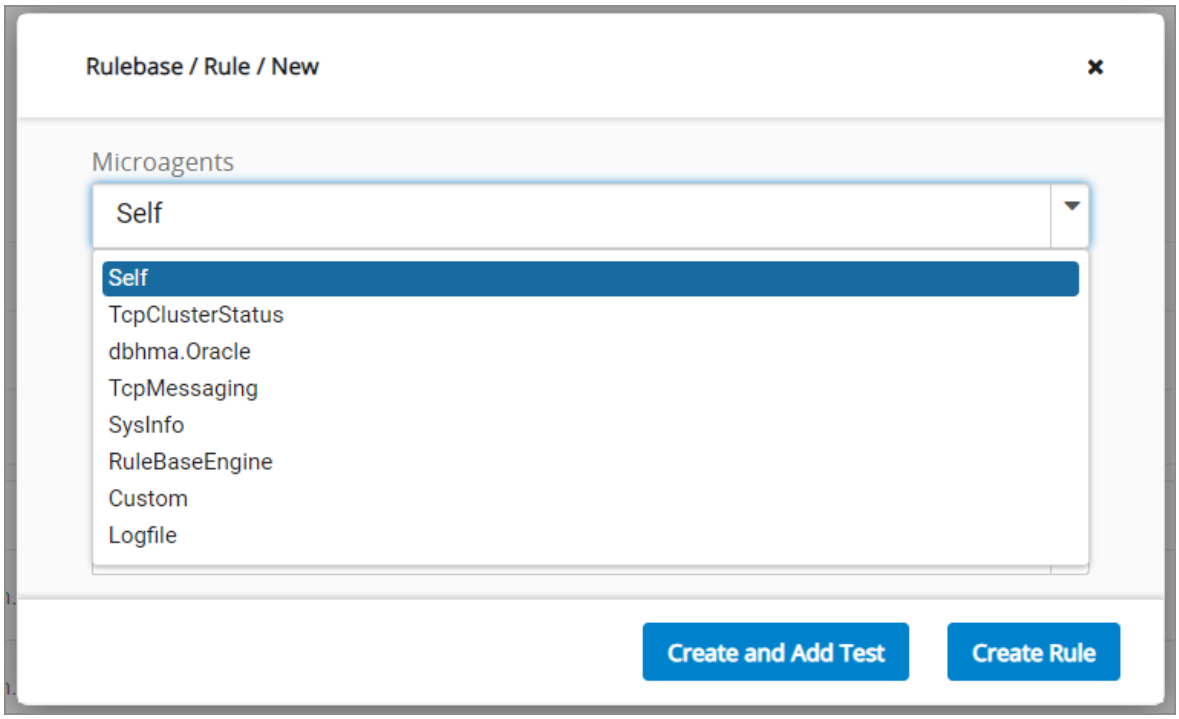
Create a new rule

On the Rule Editing screen, choose a microagent and select the corresponding method.

5. Click the **dbhma.**<*database*> microagent.

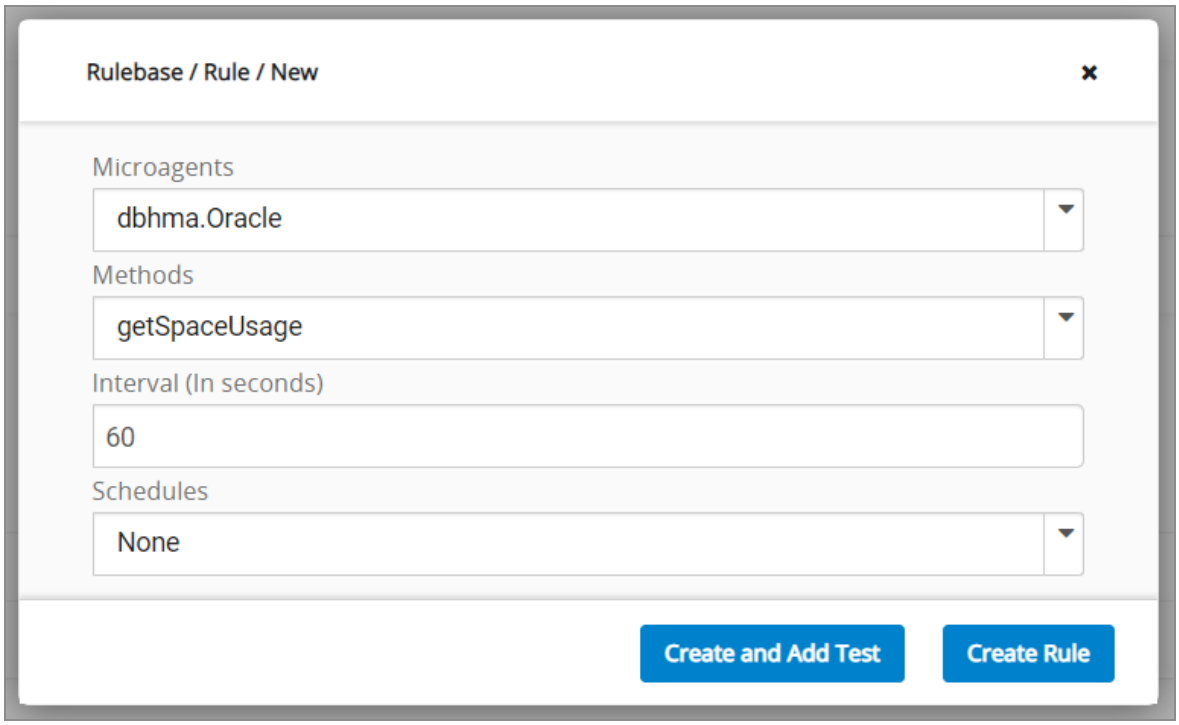
   Methods for the microagent are displayed in the panel on the right. For a detailed description of this microagent, see Using TIBCO Hawk Database Adapter Methods.

   The microagents tab should look like the following:



6. Select the **getSpaceUsage** method.

   This method returns several fields of tablespace measurement data for the database accessed by the adapter. With no argument specified, data for all tablespaces is returned.This is a synchronous method. When the rule is active, the agent subscribes to this method and receives data every 60 seconds, by default. As the rule gathers historical data, a longer data collection interval is acceptable.



7. Click **Create and Add Test** to save the changes and proceed to defining the test process.

The rule is now configured to use the `getSpaceUsage()` method of the `dbhma.`*<database>* microagent.

Applying Test Logic

The following test restricts the tablespace usage information to a single tablespace. Only data for this tablespace is received by the action.

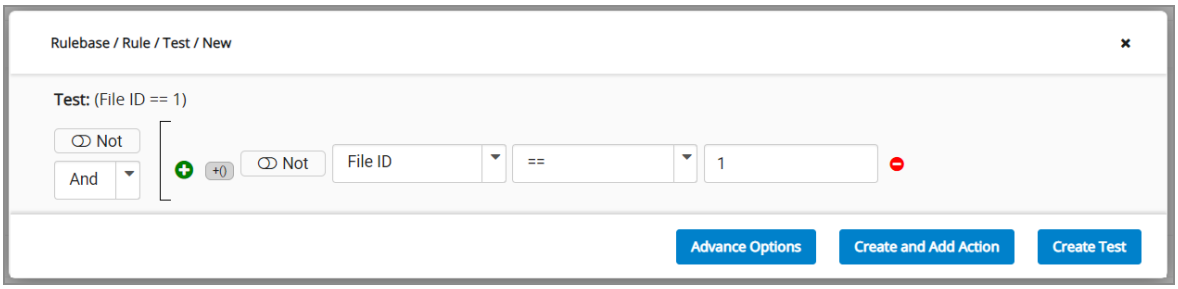
To create the test:

1. Go to the **Test** screen.
2. You can specify if you want to match all, match any, or match none.
3. Specify the test expression as `File ID` equal to (==) 1.

   `File ID` is one of six result fields returned by the `getSpaceUsage` method, the data source for this rule. `Tablespace` is a text string parameter, and all other parameters in the list are of the integer type.
4. Click **Create and Add Action** to save the changes and proceed to defining the action

process.

The dialog should look like the following:



Storing Captured Data

The following action executes a sample method for inserting data into a database table. The database, method, and table are defined in the adapter configuration file.

To create the action:

1. Go to the Action screen and select **Method**.
2. Select the **dbhma.Oracle_Sample1** microagent.

   This microagent represents the adapter instance that uses the configuration file, where the sample insert method is defined. If you added the method definition to the existing adapter configuration file, use **dbhma.Oracle** instead.
3. Select the **Record tablespace usage** method.

   This method has arguments (input parameters) defined that match return fields for the `getSpaceUsage()` method. When the action is performed, the method inserts usage information into a table named `spaceusage`.
4. Specify the arguments:

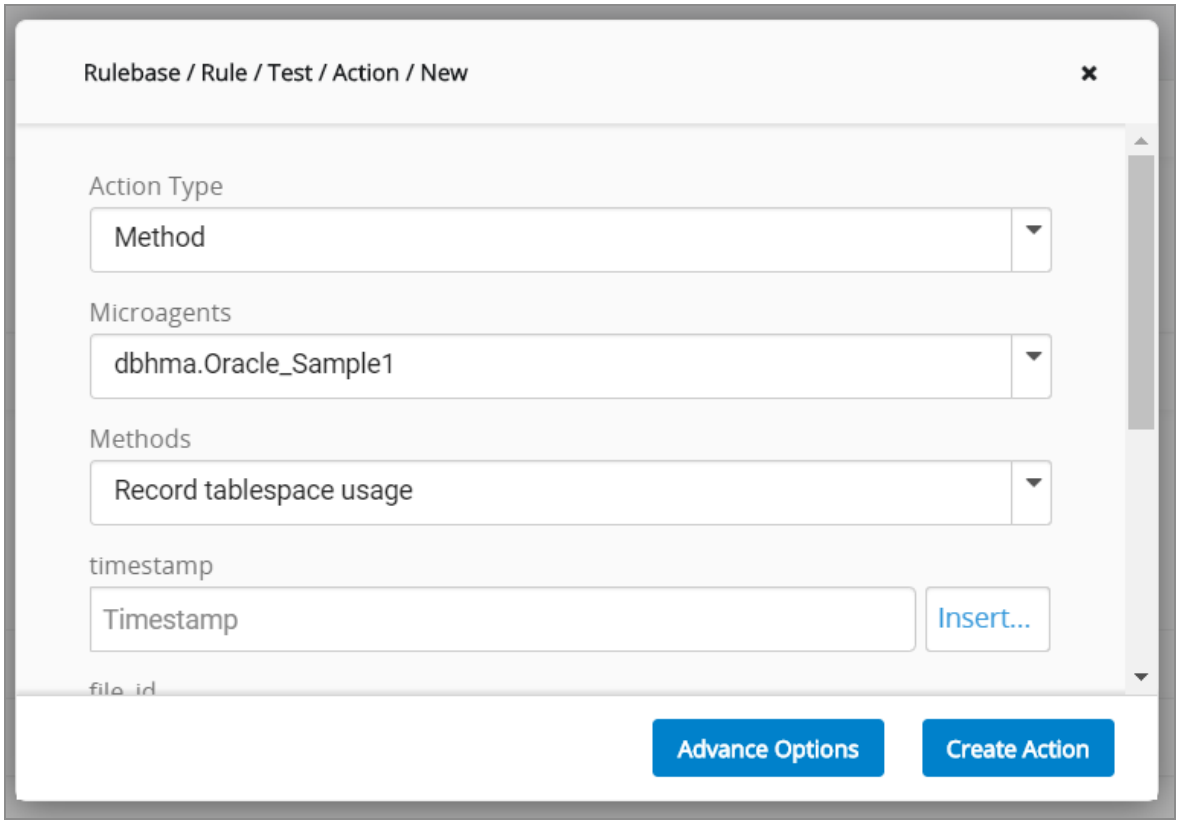
   — `timestamp:` You do not specify a data source variable for the timestamp argument, because the data source does not return any corresponding. In the method definition, this input parameter has a TIMESTAMP type. When the method is invoked, the adapter inserts a timestamp value with the current date and time.

   — `file_id`: When the method is invoked, the value of the `File ID` field returned by the data source is used as input for this method argument. For more information on data source variables, see *TIBCO® Operational*

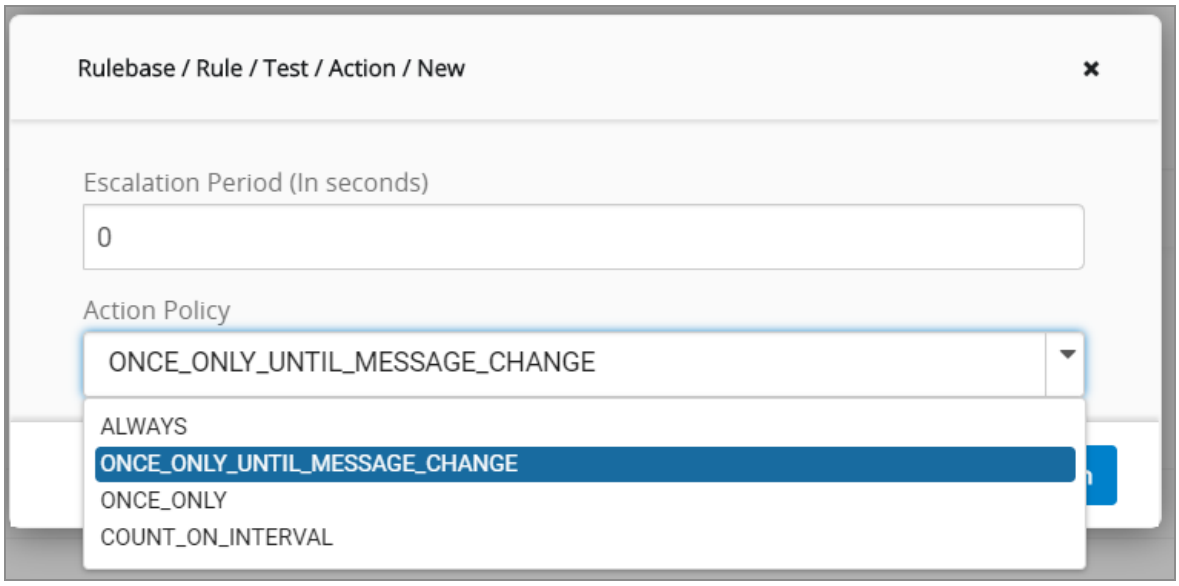
Intelligence Hawk® RedTail Administrator's Guide.

- `used`
- `percent_used`

The dialog should look like the following:



5. Click **Advance Options** in the Action screen. The Advanced Action options are displayed:



6. To specify how actions are performed, click the **ALWAYS**, **ONCE_ONLY_UNTIL_MESSAGE_CHANGE**, **ONCE_ONLY or COUNT_ON_INTERVAL**.

   By default, actions are performed only once. For this rule to retrieve data periodically, modify the default action settings. For more information on how actions are triggered and performed, see *TIBCO® Operational Intelligence Hawk® RedTail Administrator's Guide*.

7. Select **ALWAYS**.

   This setting triggers the action every time the associated test is performed. In this case, at every data collection interval.

8. Click one the following buttons on the Action Editing screen:

   — **Create Action**: To save in the browser until the user logs out.

   — **Deploy**: To save it and deploy it to the agent or repository.

Viewing the Results

You might want to view records captured and inserted by this rulebase to verify that the data capture rule is working as designed. As data is stored in a database table, you can use the `dynamicExecute()` adapter method for access.

Note: These instructions begin in the Microagents, Methods and Arguments dialog for the local agent on the adapter machine. For instructions on accessing this dialog, see Invoking the dynamicExecute Method.

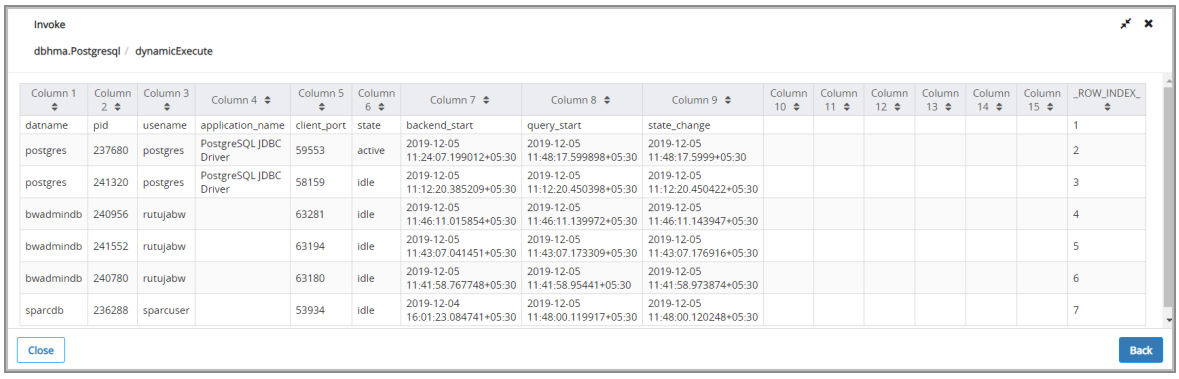
To invoke the `dynamicExecute()`method:

1. Select the microagent name, in this case **dbhma.Postgresql**, to display a list of associated methods and text descriptions. For detailed descriptions of these methods, see Using TIBCO Hawk Database Adapter Methods.
2. Select the name of the method to invoke, that is, **dynamicExecute()**.
3. Type the SQL statement for the query in the **Arguments** field. For example, on PostgreSQL:

   ```
   SELECT datname, pid, usename, application_name, client_port, state,
   backend_start, query_start, state_change FROM pg_stat_activity WHERE
   state = 'idle' or state ='active';
   ```

   This statement returns database name, process ID, state of the query, time when the client application connected to the database server, time at which query execution was initiated, and time when the state of the query was changed. No statement delimiter character is required. Only one SQL statement can be executed per method invocation.
4. Click **Invoke** to execute the query.

   The Invocation Results dialog displays method results, the database rows returned by the query:



Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11	Column 12	Column 13	Column 14	Column 15	_ROW_INDEX_
datname	pid	usename	application_name	client_port	state	backend_start	query_start	state_change							1
postgres	237680	postgres	PostgreSQL JDBC Driver	59553	active	2019-12-05 11:24:07.199012+05:30	2019-12-05 11:48:17.599898+05:30	2019-12-05 11:48:17.5999+05:30							2
postgres	241320	postgres	PostgreSQL JDBC Driver	58159	idle	2019-12-05 11:12:20.385209+05:30	2019-12-05 11:12:20.450398+05:30	2019-12-05 11:12:20.450422+05:30							3
bwadmindb	240956	rutujabw		63281	idle	2019-12-05 11:46:11.015854+05:30	2019-12-05 11:46:11.139972+05:30	2019-12-05 11:46:11.143947+05:30							4
bwadmindb	241552	rutujabw		63194	idle	2019-12-05 11:43:07.041451+05:30	2019-12-05 11:43:07.173309+05:30	2019-12-05 11:43:07.176916+05:30							5
bwadmindb	240780	rutujabw		63180	idle	2019-12-05 11:41:58.767748+05:30	2019-12-05 11:41:58.95441+05:30	2019-12-05 11:41:58.973874+05:30							6
sparcdb	236288	sparcuser		53934	idle	2019-12-04 16:01:23.084741+05:30	2019-12-05 11:48:00.119917+05:30	2019-12-05 11:48:00.120248+05:30							7

For more information on invoking TIBCO Hawk methods, see *TIBCO® Operational Intelligence Hawk® RedTail Administrator's Guide*.

Monitoring Database Activity

This section demonstrates how to monitor a database server using default methods provided by the adapter. The method in this example monitors database block buffer size, but any of the adapter methods that monitor database parameters could be used. Default adapter methods are described in Using TIBCO Hawk Database Adapter Methods.

The steps in this section assume that a custom method for executing a script is defined in the adapter configuration file. The XML code for the method in this example is:

```
<method
    name = "analyzeBuffer"
    help = "Analyzes the database buffer"
    SQL = "file:c:\dba\scripts\maintenance\analyzebuffer.sql"
    handlerType = "U"
  >
</method>
```

For descriptions of the configuration file syntax, see Getting Started.

Building a Rule for Database Monitoring

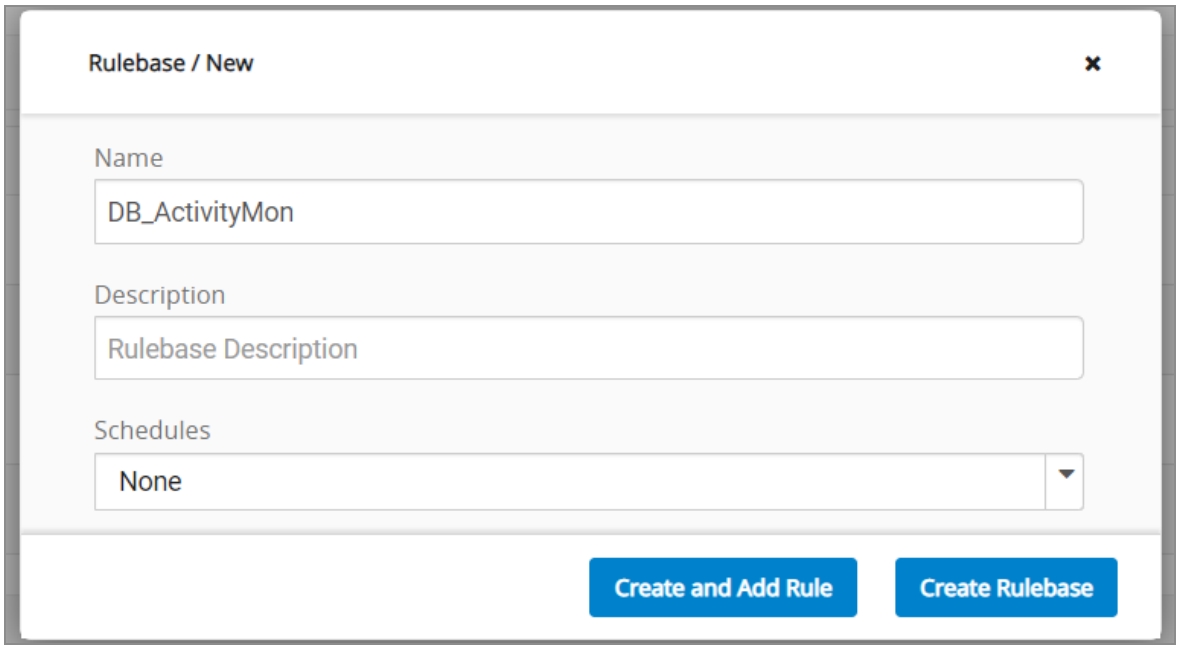
These steps demonstrate how to create a rulebase for monitoring a database server. The rulebase contains a simple rule that uses the `Oracle.getBufHitRatio()` method as a data source. This method compares the number of reads from disk versus memory (block buffers) for the entire database. A test checks for an acceptable hit ratio value, which should be greater than 95 percent for an interactive system. At the data collection interval, the agent where the adapter is running retrieves usage data, tests it, and executes a script to analyze the buffer size for later tuning.

For instructions on starting TIBCO Hawk Console, see *TIBCO® Operational Intelligence Hawk® RedTail User Guide*.

To create a new rulebase in TIBCO Hawk Console:

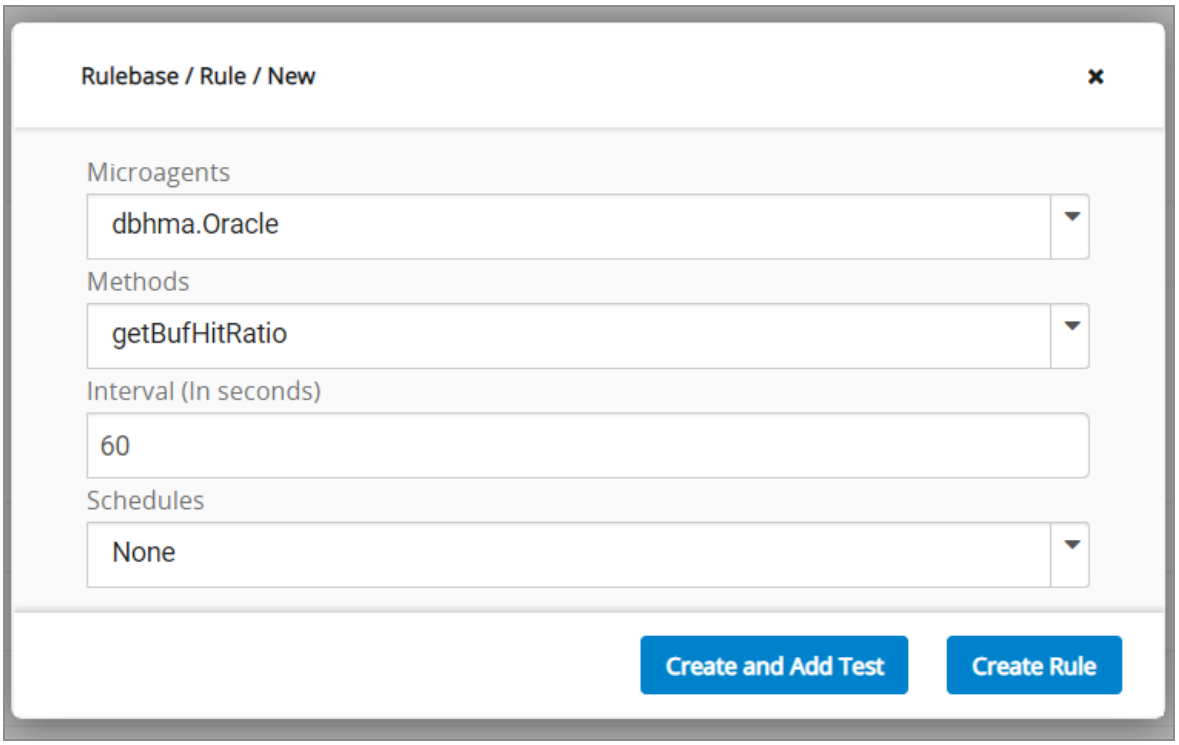
1. Go to the **Rulebase** tab. The Rulebase screen lists the rulebases loaded by the selected agent and provides a toolbar for performing rulebase operations.
2. Click the **Create new Rulebase** icon from the toolbar to create a new rulebase. A new rulebase is created with the default name "New-Rulebase". Type **DB_ ActivityMon** in the **Rulebase Name** field.

3. Enter the other fields, as required.
4. Click **Apply Changes** to save the current changes.
5. Click **Create & Add Rule** to save the changes and proceed to the rule building process.



Create a new rule

6. On the Rule Editing screen, choose a microagent and select the corresponding method.



7. Select the **dbhma.**<*database*> microagent. The Methods panel displays the methods for the microagent. For a detailed description of this microagent, see Using TIBCO Hawk Database Adapter Methods.
8. Select the **getBufHitRatio** method.

   This method takes no arguments and returns a single value, the database block buffer hit ratio. This is a synchronous method. When the rule is active, the agent invokes this method and receives data every 60 seconds, by default. On a non-critical system, you might want to perform this check less often, so a longer data collection interval is acceptable.
9. Select **3600** as the **Data Delivery**.

   The dialog should look like the following:
10. Click **Create and Add Test** to save the changes and proceed to defining the test process.

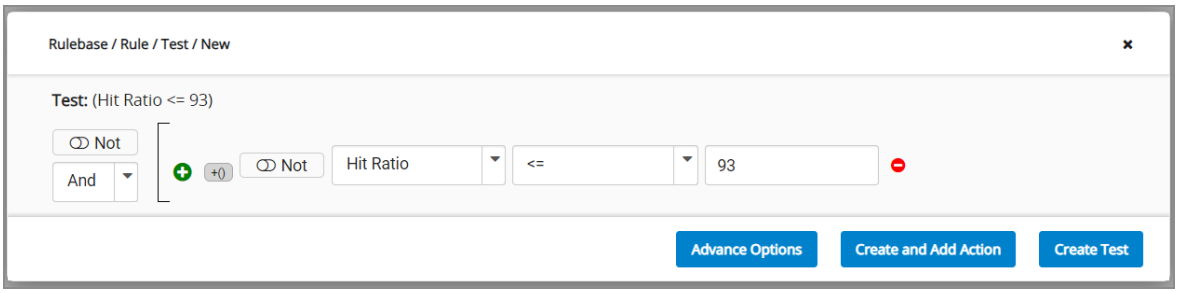
The rule is now configured to use the `getBufHitRatio()` method of the **dbhma.**<*database*> microagent.

Applying Test Logic

The following test checks the hit ratio for values less than or equal to 93 percent. If the hit ratio is within this range, the action is triggered.

To create the test:

1. Go to the **Test** screen.



2. You can specify if you want to match all, match any, or match none.
3. Specify the test expression as `Hit Ratio <= 93`.

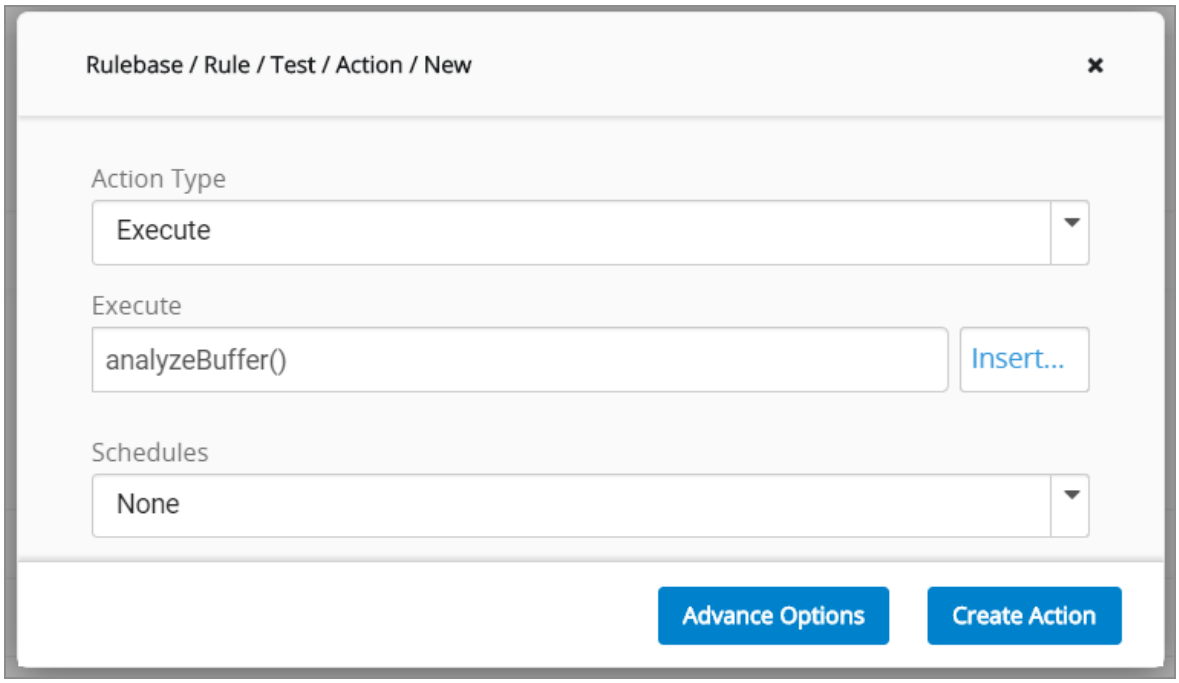
   `Hit Ratio` is the only result field returned by the `getBufHitRatio` method, the data source for this rule. `Hit Ratio` is an integer field, so only numeric operators are listed. For descriptions of test operators, see *TIBCO® Operational Intelligence Hawk® RedTail Administrator's Guide*.
4. Click **Create and Add Action** to save the changes and proceed to defining the action process.

Analyzing the Buffer Usage

The following action executes a user-defined SQL script for analyzing the database block buffer usage. The path to the script is specified in the `SQL` attribute of this method in the adapter configuration file.

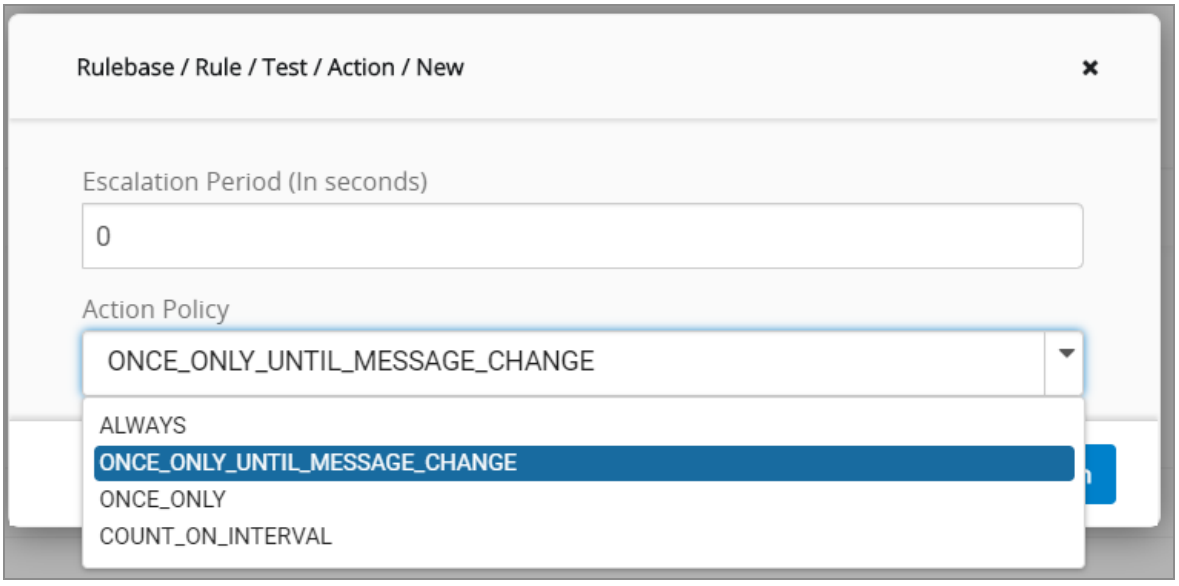
To create the action:

1. Go to the Action screen.



2. Select **Execute**.
3. Specify **analyzeBuffer()** in **Execute**.

   This method is a custom adapter method that executes a script. The script analyzes the database buffer usage. It takes no arguments. When the action is performed, the method executes a script named `analyzebuffer.sql`.
4. Click **Advance Options** in the Action screen. The **Advance Options** are displayed:



To specify how actions are performed, select one from the **ALWAYS**, **ONCE_ONLY_UNTIL_MESSAGE_CHANGE**, **ONCE_ONLY or COUNT_ON_INTERVAL** drop down menu.

By default, actions are performed only once. For this rule to execute the script each time hit ratio is tested, modify the default action settings. For more information on how actions are triggered and performed, see *TIBCO® Operational Intelligence Hawk® RedTail Administrator's Guide*.

5. Select **ALWAYS**.

   This setting triggers the action every time the associated test evaluates to `true`. In this case, the action is performed at data collection intervals when the hit ratio is 93 percent or lower.

6. Click one the following buttons on the Action Editing screen:

   — **Create Action**: To save in the browser until the user logs out.

   — **Deploy**: To save it and deploy it to the agent or repository.

Receiving Notification

You might want to receive notification that the `analyzebuffer.sql` script was executed. Many consecutive script executions could indicate a serious problem with the database. A second action can create an alert message after the script is executed.

To receive the notification:

1. Go to the Action screen.
2. Select **Alert**.
3. Select the **Alert Level** as **Medium**.
4. In the **Alert Message** field, type:

```
analyzebuffer.sql was executed
```

5. Click **Advanced Options** in the Action screen. The Advanced Action options are displayed.

   To specify how actions are performed, select one from the **ALWAYS**, **ONCE_ONLY_UNTIL_MESSAGE_CHANGE**, **ONCE_ONLY or COUNT_ON_INTERVAL** drop down menu.

By default, actions are performed only once. For this rule to send an alert message each time the script is run, modify the default action settings. For more information on how actions are triggered and performed, see *TIBCO® Operational Intelligence Hawk® RedTail Administrator's Guide*.

6. Select **ALWAYS**.

   This setting triggers the action every time the associated test evaluates to `true`. In this case, an alert is created at data collection intervals when the hit ratio is 93 percent or lower.

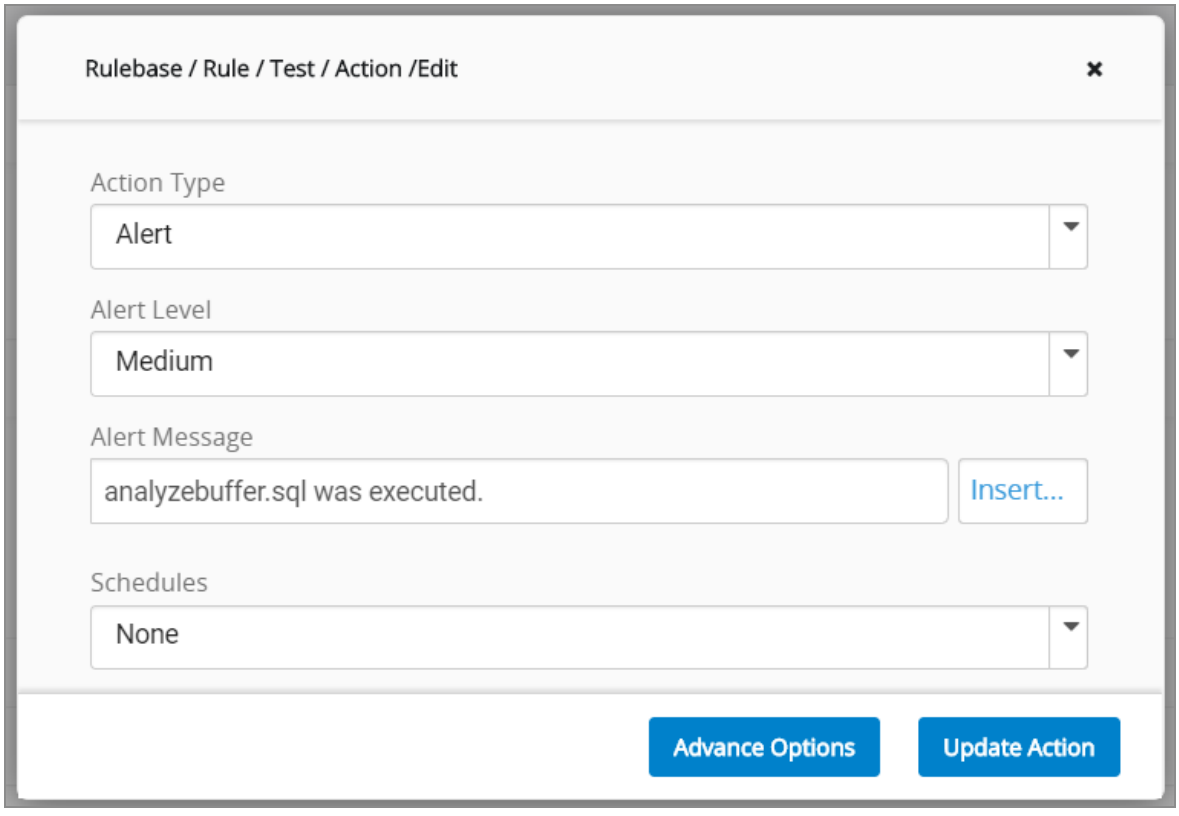
7. Click one the following buttons on the Action Editing screen:

   — **Create Action**: To save in the browser until the user logs out.

   — **Deploy**: To save it and deploy it to the agent or repository.

   This action sends an alert message when the action that executes `analyzebuffer.sql` is performed. The alert message can be viewed in TIBCO Hawk Console.

   The dialog should look like the following:

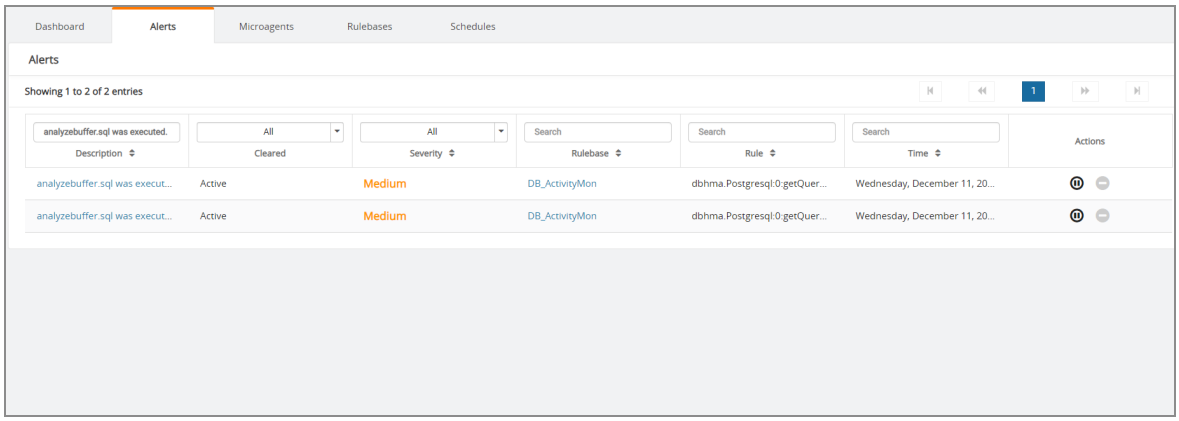


Viewing the Results

When the database block buffer size gets too low, the agent where the adapter is running executes a script to analyze the buffer. The agent also generates a medium-level alert, with the alert text '`analyzebuffer.sql was executed`'.

To view alerts:

1. Click on the **Alerts** tab. The Alert window displays messages for the selected agent.



2. Expand the alert to display detailed properties.

For more information on invoking TIBCO Hawk methods, see *TIBCO® Operational Intelligence Hawk® RedTail Microagent Reference Guide*.

Integrating Data from Multiple Sources

Heterogeneous database environments supporting different ERP applications can be found in the same organization. Along with advantages, it also presents challenges for integrating data from multiple sources. Multiple database environments can be integrated by having multiple instances of the adapter, each communicating with a different database. These adapters would run on the same machine which also runs the agent.

The following example illustrates how data from multiple sources can be integrated.

Problem: An organization had its manufacturing data stored in an Oracle database and its inventory management data in a Sybase database. When the manufacturing process reaches the final stage, the inventory for that product (identified by its unique manufacturing code) needs to be conveyed to the inventory management system.

Solution: Each of these databases are monitored by their own instance of the adapter. These adapters would run on the same machine which also runs the agent. When the manufacturing process reaches the final stage, a record is inserted in a table which holds the quantities and the manufacturing codes for the inventory that has completed processing. A rulebase could be set up to use a method from the adapter monitoring this Oracle database to query the data from this table, and then use another adapter monitoring the Sybase inventory database to update the inventory for the particular product code. This could be done by either invoking a stored procedure in the Sybase inventory management environment or by executing some SQL statements.

TIBCO Hawk Database Adapter Document Type Definition (DTD)

This appendix describes the Document Type Definition (DTD) that defines syntax rules for the adapter configuration file.

Viewing the dbhma.dtd File

The `dbhma.dtd` file contains syntax rules for the adapter configuration file. During installation, this file is copied to your `HAWK_DBHMA_HOME/examples/<database>` directory. When you use a validating parser to validate the XML in an adapter configuration file, instructions specified in the DTD are used to check file entries for syntax errors. For instructions on validating configuration file syntax, see Validating the XML.

This appendix presents the contents of the `dbhma.dtd` file for reference while using this manual. For detailed descriptions of all elements and attributes defined in the DTD, see Getting Started.

```
<!--

    Copyright(c) 1999-20197 TIBCO Software Inc. All rights reserved.

    **** The DBHma Document Type Definition ****


    This file defines the grammar for all the constructs used

    in the DBHma XML configuration file. With this file included,
```

```
    the XML configuration file can be syntax checked using any
    "validating" XML parser.
-->

<!ENTITY % string "CDATA">

<!--
     The "valueType" defines the data type supported.
     The data types are following the JDBC convention. The
     JDBC API document can be referenced for details.

     The types CHAR and VARCHAR are basically the same. They can be
     used for database types char(n), or varchar(n).

     The TIMESTAMP type should be used for all database "date" or
     "datetime" types.

     The BIGINT type is for integer greater than 2 ** 31 -1 or
```

```
      smaller than - 2 ** 31.

      The DOUBLE is for 64 bit floating point numbers while the FLOAT
      and REAL are for 32 bit floating point numbers.

      The rest of types are used for the corresponding database
      types if it's applicable.
-->

<!ENTITY % valueType     "
(CHAR|VARCHAR|TIMESTAMP|TINYINT|SMALLINT|INTEGER|BIGINT|REAL|FLOAT|DOUBLE)">

<!--
     All the "help" attributes below are each paired with a "name"
     attribute. The "help" attributes are optional.
      If a "help" attribute were not defined, the "name"
     (attribute) would be used as the "help text".
-->
```

```
<!ELEMENT TIBHAWK_AMI (microagent+)>

<!--
     The 'xml_file_version' is optional and it can be used to
     identify the version of this xml file. It has to be in the
     form of <major>.<minor>.<update>, for example, "1.1.0". The
     xml file name and the version number can be queried by the
     built-in method "getReleaseVersion".

     The 'ami_rvd_service', 'ami_rvd_network', and
     'ami_rvd_daemon' together configures the RV parameters for
     creating an RV session for AMI RVD connection.

     The 'as_listen_url', 'as_discover_url',
     'as_transport_timeout', 'as_receive_buffer_size',
     'as_virtual_node_count' and 'as_worker_thread_count' together
     configures the AS parameters for creating an AS session for
```

```
      AMI connection.

      The 'traceFile', 'traceFileMaxSize', 'traceFileMaxNumber',
      and 'traceLevel' specify the AMI trace file name (a full name
      which includes the directory path), the maximum size the trace
      file is allowed to grow, the maximum number of "roll over"
      trace files to be maintained, and the starting trace level.
      (Refer the DBHma User's Guide for details.) These
      attributes are optional, if not defined, "stderr" would be
      used as default. Either the defined trace file or the default
      trace file would be used for all the microagents defined in
      this XML configuration file unless different values of these
      attributes are specified along with a "microagent" tag. If so,
      the attributes specified there will be used for the tracing of
      that microagent only.

      If the attribute 'log_format' is specified to "ae4", the trace
      log entries will be in TIBCO AE format. The default is in Hawk
```

```
      format. The attributes 'app_id' and 'product_id' are used only

      if the log_format is set to "ae4". They are used to specify

      the application ID and product ID in log file entries.



      The 'JDBCDriver' is required to specify the database JDBC

      driver class.

      For example,

      Database     Version    StandardDriverLibrary         JDBC
Driver                     AccessURL

      ORACLE     9i         ojdbc14.jar             oracle.jdbc.driver.OracleDriver
jdbc:oracle:thin:@<HOST>:<PORT>:<SID>

      ORACLE     10g-
Rx         ojdbc5.jar/ojdbc6.jar         oracle.jdbc.driver.OracleDriver         jdbc:oracl
e:thin:@<HOST>:<PORT>:<SID>

      ORACLE     11g-
Rx         ojdbc5.jar/ojdbc6.jar         oracle.jdbc.driver.OracleDriver         jdbc:oracl
e:thin:@<HOST>:<PORT>:<SID>

      ORACLE     12g-
Rx         ojdbc6.jar             oracle.jdbc.driver.OracleDriver         jdbc:oracle:thi
n:@<HOST>:<PORT>:<SID>



      SQLSERVER     2K5         sqljdbc.jar (JRE
5.x)         com.microsoft.sqlserver.jdbc.SQLServerDriver     jdbc:sqlserver://
<HOST>:<PORT>;databaseName=<DBNAME>

      SQLSERVER     2K5         sqljdbc4.jar (JRE
6/7)        com.microsoft.sqlserver.jdbc.SQLServerDriver     jdbc:sqlserver://
```

```
<HOST>:<PORT>;databaseName=<DBNAME>

     SQLSERVER     2K8-Rx         sqljdbc.jar (JRE
5.x)        com.microsoft.sqlserver.jdbc.SQLServerDriver     jdbc:sqlserver://
<HOST>:<PORT>;databaseName=<DBNAME>

     SQLSERVER     2K8-Rx         sqljdbc4.jar (JRE
6/7)        com.microsoft.sqlserver.jdbc.SQLServerDriver     jdbc:sqlserver://
<HOST>:<PORT>;databaseName=<DBNAME>

     SQLSERVER     2K10-Rx     sqljdbc.jar (JRE
5.x)        com.microsoft.sqlserver.jdbc.SQLServerDriver     jdbc:sqlserver://
<HOST>:<PORT>;databaseName=<DBNAME>

     SQLSERVER     2K10-Rx     sqljdbc4.jar (JRE
6/7)        com.microsoft.sqlserver.jdbc.SQLServerDriver     jdbc:sqlserver://
<HOST>:<PORT>;databaseName=<DBNAME>

     SQLSERVER     2K12-Rx     sqljdbc.jar (JRE
5.x)        com.microsoft.sqlserver.jdbc.SQLServerDriver     jdbc:sqlserver://
<HOST>:<PORT>;databaseName=<DBNAME>

     SQLSERVER     2K12-Rx     sqljdbc4.jar (JRE
6/7)        com.microsoft.sqlserver.jdbc.SQLServerDriver     jdbc:sqlserver://
<HOST>:<PORT>;databaseName=<DBNAME>

     SQLSERVER     2K14-Rx     sqljdbc.jar (JRE
5.x)        com.microsoft.sqlserver.jdbc.SQLServerDriver     jdbc:sqlserver://
<HOST>:<PORT>;databaseName=<DBNAME>

     SQLSERVER     2K14-Rx     sqljdbc4.jar (JRE
6/7)        com.microsoft.sqlserver.jdbc.SQLServerDriver     jdbc:sqlserver://
<HOST>:<PORT>;databaseName=<DBNAME>



     SYBASE     ASE-
11.x         jconn3.jar         com.sybase.jdbc4.jdbc.SybDriver           jdbc:sybase:Tds:
<HOST>:<PORT>?CHARSET=iso_1

     SYBASE      ASE-
12.x         jconn3.jar         com.sybase.jdbc4.jdbc.SybDriver           jdbc:sybase:Tds:
```

```
<HOST>:<PORT>?CHARSET=iso_1

     SYBASE      ASE-
15.x         jconn4.jar         com.sybase.jdbc4.jdbc.SybDriver            jdbc:sybase:Tds:
<HOST>:<PORT>?CHARSET=iso_1

     MySQL     5.x         mysql-connector-java-5.1.7-
bin.jar    com.mysql.jdbc.Driver             jdbc:mysql://<HOST>:<PORT>/<DBNAME>

     DB2     v9.7+         db2jcc.jar+db2jcc_license_
cu.jar    com.ibm.db2.jcc.DB2Driver         jdbc:db2://<HOST>:<PORT>/<DBNAME>

     Refer to the Hawk Database Adapter documentation and JDBC

     driver documentation for detailed explanation on their usage.

     The 'dbUser' and 'dbPassword' determines which database user

      is to login to the database. The database user would determine

     what kind of information can be accessed and what kind of

     operation can be performed. The 'dbURL', 'dbUser' and

     'dbPassword' would determine a "database connection". The

     "database connection" specified here would be used as default
```

```
      for all the microagent methods in this configuration file
      unless different values of these attributes are specified
      along with a "microagent" tag. If so, the attributes (dbURL,
      dbUser and dbPassword) specified there will be used for
      creating a separate "database connection" and is used for all
      the methods in that microagent only.

      The 'dbEnvVars' is optional. It may contain additional env.
      variable settings to be used by the JDBC driver, for example,
      "PROTOCOLTRACE=2;PROTOCOLTRACEFILE=/tmp/trace.out". Please
      refer the JDBC driver documentation on whether the usage of
      environment variables is supported.

      The 'dateFormat' attribute specifies the default "date" or
      "datetime" conversion format for the database type "date" or
      "datetime". The database "date" or "datetime" are handled as
      character string by TIBCO Hawk. Hence, it has to be converted
      to character string when it comes out of database. For input
```

```
      parameters, character strings are converted to database
      "date" or "datetime" before being sent to the database. Since
      it's a string, it can not be used with "greater than" or
      "smaller than" type of comparison is TIBCO Hawk rules. If
      those type of comparisons are needed, the user has to use
      database functions to get the "date" or "datetime" value in
      numbers (for example, how many seconds since Jan. 1, 1970,
      ...).

      The "date format" pattern is based on the JAVA
      java.text.SimpleDateFormat specification. For example,
      "MM/dd/yyyy:HH:mm:ss" would produce a datetime string like
      "07/12/1999:23:15:00". For detailed information, refer to the
      JAVA API documentation.

      If a particular parameter is in a different "dateFormat" from
      the default format (typically, data came from some external
      source), the users may specify the "dateFormat" at the
```

```
      parameter level (use the "dateFormat" attribute associated
      with the "inputParameter" or "outputParameter" tag). This way,
      it would affect that particular parameter only.

      The 'sqlFileDir' is specifies the default directory to search
      SQL files if a file specified is not as a form of full file
      path. If this attribute is not specified, the current database
      when DBHma is run would be used as the default directory.
-->

<!ATTLIST TIBHAWK_AMI
    xml_file_version   %string; #IMPLIED
    hawk_domain        %string; #IMPLIED
    agent_name         %string; #IMPLIED
    ami_rvd_service    %string; #IMPLIED
    ami_rvd_network    %string; #IMPLIED
    ami_rvd_daemon     %string; #IMPLIED
    as_listen_url      %string; #IMPLIED
```

```
    as_discover_url    %string; #IMPLIED

    as_transport_timeout      %string; #IMPLIED

    as_receive_buffer_size    %string; #IMPLIED

    as_virtual_node_count     %string; #IMPLIED

    as_worker_thread_count    %string; #IMPLIED

    traceFile          %string; #IMPLIED

    traceFileMaxSize   %string; #IMPLIED

    traceFileMaxNumber %string; #IMPLIED

    traceLevel         %string; #IMPLIED

    log_format         (ae4)    #IMPLIED

    app_id             %string; #IMPLIED

    product_id         %string; #IMPLIED

    dbType    (ORACLE|SYBASE|SQLSERVER|MYSQL|DB2) #REQUIRED

    JDBCDriver  %string;        #REQUIRED

    dbURL       %string;        #REQUIRED

    dbUser      %string;        #REQUIRED

    dbPassword  %string;        #REQUIRED

    dbEnvVars   %string;        #IMPLIED
```

```
    dateFormat  %string;        #IMPLIED
    sqlFileDir  %string;        #IMPLIED>

<!-- At least one microagent has to be defined. -->

<!ELEMENT microagent (method+)>

<!--
     The attributes 'dbURL', 'dbUser', 'dbPassword', and
     'dbEnvVars' can be specified to override the corresponding
     attributes associated with the 'TIBHAWK_AMI' tag. If so, only
     the methods belong to this microagent would be affected.

     The attributes 'traceFile', 'traceFileMaxSize',
     'traceFileMaxNumber', and 'traceLevel' can be specified to
     override the corresponding attributes associated with the
     'TIBHAWK_AMI' tag. If so, only the methods belong to this
     microagent would be affected.
```

```
      The 'maxThreads' defines the maximum number of threads a
      "microagent" can have to perform method invocations in
      parallel. The default value is 1. That is for each microagent
      only one thread is available - all the methods invocation
      of that microagent would be serialized (one method has to wait
      until previous method execution completed before it can be
      executed). Since each thread would consume one database
      connection, this attribute should be set ONLY when parallel
      method invocation is really necessary.
-->

<!ATTLIST microagent
    name               %string; #REQUIRED
    displayName        %string; #IMPLIED
    dbURL              %string; #IMPLIED
    dbUser             %string; #IMPLIED
    dbPassword         %string; #IMPLIED
```

```
    dbEnvVars          %string; #IMPLIED

    traceFile          %string; #IMPLIED

    traceFileMaxSize   %string; #IMPLIED

    traceFileMaxNumber %string; #IMPLIED

    traceLevel         %string; #IMPLIED

    maxThreads         %string; #IMPLIED

    help               %string; #IMPLIED>


<!-- At least one method has to be defined for each microagent. -->


<!ELEMENT method ((inputParameter+)?,(outputParameter+)?)>


<!--

     If a method has "arguments", the "arguments" must have at

     least one "input parameter". If a method has "returns", the

     "returns" must have at least one "output parameter".

-->
```

```
<!--
      If a method returns more than one rows of information, the
      'index' attribute must be specified with the name of the
      output parameter which can uniquely identify each row. If
      more than one output parameters are needed to uniquely
      identify a row (i.e., composite index), the "index" attribute
      must be specified with those parameter names separated with
      commas.

       The 'timeout' attribute specifies the longest time to wait
      for the return data when this methods is invoked. The default
      timeout value is 10 seconds. If the data are not returned
      before the timeout is reached, the HAWK agent would receive a
      'timeout' error even if the data are eventually received.

      The 'handlerType' specifies how a method should be handled.
      The supported handler types are "Q, QD, U, UD, I, P, PR, and
      S. The "Q" and "QD" are for queries, and "U" and "UD" are for
```

```
      all other type of SQL's. "I" is dedicated for "insert". "P"
      and "PR" are for store procedures or functions. "S" is for
      system (DBHma) provided methods. Detailed description for each
     handlerType is explained in later sections.

      The 'SQL' attribute specifies the SQL statement to execute.
      This is a required attribute for all methods except those
      methods with 'handlerType' attribute "I" or "S". If the value
      of the "SQL" attribute starts with a string "file:", then the
      string after "file:" is assumed to be a file name contains the
      actual SQL statements. If the file name is not a full file
      path name, it is assumed to be in the current directory where
      the DBHma is running unless the "sqlFileDir" attribute is
      specified along with the tag "TIBHAWK_AMI". In that case, the
      file is search through the directory specified in
      "sqlFileDir".

      The SQL statements may contain input or output parameters.
```

```
      Only for the methods with handler type "P", output parameters
      are allowed to be in SQL statement. They have to be entered in
      the form of "${<parameter name>}". For example, a SQL
      statement like "select first_name from employee where
      last_name = ${Last Name}"
      is assuming there is an input parameter of this method with
      parameter name "Last Name".

      The handler type "Q" is used for "queries. All the queries
      with "select"
      SQL statements or Stored procedure executions when they are
      done through SQL statements like "exec <store procedure name>
      (...)" that returns output through "select" statements, should
      use this handler type.

      The handler type "U" is used for all other type of SQL's such
      as "update", "delete", "create", ... etc. With this handler
      type, it allows only one output parameter (it's optional) with
```

```
    INTEGER data type to return the "update count" - Hence, only
    one row will be returned.

     The handler type "QD" can only be used by the method
     "dynamicQuery". The handler type "UD" can only be used by the
     method "dynamicExecute". The configuration for these two
     methods should not be changed from the example provided. These
     two methods let users enter SQL statements on the fly. The
     "dynamicQuery" is used for all SQL's with "select" or "exec"
     (Execution of stored procedures returning output via "select"
     statements). The "dynamicExecute" is used for all other kind
     of SQL's.

     The handler type "P" and "PR" are used for store procedure or
     function calls. In these cases, input parameters, if any, are
     used to pass parameters to store procedures or functions. The
     SQL attribute string should contain a "call" key word. This is
     different from the previous "exec" cases where parameters are
```

```
      passed as part of a SQL character string.

      The handler type "PR" should be used for store procedures
      returning data through "select" statements. The SQL statement
      has to in the form of

      call <proc name> (${<input parameter 1>}, ..., ${<input
      parameter n>})

      where the <input parameter n> corresponds to an input
      parameter name.

      By default the output parameters are mapped to the first
      "ResultSet" - a "ResultSet" means a set of returned data from
      one "select" statement - in the stored procedure. If there are
      more than one "result sets" to be returned from the store
      procedure and the user may use the 'resultSetIndex' attribute
      (index start from "1") to indicate which "ResultSet" should be
```

```
      used to map to the method output parameters.
      For example, if the output parameters are mapped to the second
      "select" statement returning data, the "resultSetIndex" should
      be set to "2".

      The handler type "P" should be used for functions or stored
      procedures      returning data through store procedure "out"
      or "in out" parameters.
      The SQL statement has to in the form of

      call <proc/func name> (${<input/output parameter m>}, ...,
      ${<input/output parameter n>})
      or

      ${output parameter x} = call <pro/func name>
      (${<input/output parameter m>), ..., ${<input/output
      parameter n>})
```

```
      where the <input/output parameter ?> corresponds to an input
      or an output parameter name of this method. If it's an input
      parameter, the corresponding store procedure or function
      parameter has to be an "in" or "in out" parameter. If it's an
      output parameter, the corresponding store procedure or
      function parameter has to be an "out" or "in out" parameter.
      For example, for a store procedure

      procedure procinout (x1 in varchar, x2 in out smallint, x3 out
     int)

      the method SQL attribute may look like

      ${Return Value} = call procinout (${P1}, ${P2}, ${P3})

      The parameters may be entered as the following

      <inputParameter
```

```
      name = "P1"
      type = "VARCHAR">
      </inputParameter>
      <inputParameter
      name = "P2"
      type = "SMALLINT">
      </inputParameter>
      <outputParameter
      name = "P2"
      type = "SMALLINT">
      </outputParameter>
      <outputParameter
      name = "P3"
      type = "INTEGER">
      </outputParameter>
      <outputParameter
      name = "Return Value"
      type = "INTEGER">
```

```
    </outputParameter>

    Note that the method parameters do not have to follow the
    order they appear in the SQL string.

    The handler type "I" is used for "insert" methods. An "insert"
    method is a method for inserting data into database only. The
    "tableName" attribute must be specified which identifies which
    database table to insert to. Also, the input parameter names
    have to match the table column names. For example, a method
    with tableName "spaceUsage" with input parameters like the
    following:

    <inputParameter
    name = "timestamp"
    type = "TIMESTAMP">
    </inputParameter>
    <inputParameter
```

```
      name = "percent_used"
      type = "INTEGER">
      </inputParameter>

      would insert a row to the table "spaceUsage" which has two
      columns "timestamp" and "percent_used".

      If the table doesn't exist in the database, DBHma would create
      the table before the first "insert". For the columns with type
      "CHAR" or "VARCHAR", a column with database type "varchar(80)"
      would be created. If the data for that column is more than 80
      characters or much less, the user should create the table with
      appropriate columns first before any "insert" is executed.

      The handlerType "S" is used for "system methods". Currently,
      there are only two "system methods", i.e., the "shutdown" and
      "reload" methods. No other methods should use this ("S")
      handler type and the configuration for these two methods
```

```
      should not be changed from the example provided.

      The "shutdown" method is to provide a "clean" way to terminate
      the DBHma which would close all the database connections
      before shutting down.

      The "reload" method is to reload the configuration file
      without shutting down the DBHma.

      If the 'useCache' attribute is set to "true", the SQL for this
      method would be "prepared" only when the first time this
      method is invoked, and the "prepared" statement would be used
      for all subsequent invocations.
      However, because each "prepared" statement would keep an
      "open" cursor (which would cost some database system
      resource), this 'useCache' attribute should be set to "true"
      only if a method is to be executed in a very high frequency.
```

```
      The "logReturnValue" attribute can only be used for the
      methods with handlerType "P" and one of the output parameter
      is the stored procedure or function return value. If the
      "logReturnValue" is set to true and the stored procedure or
      function returns an integer greater than 0, the return value
      would be logged to the trace file in a format of

      ... <method name > returns <returned value>

      If the "logReturnValue" is set to true and the stored
      procedure or function returns an non-integer value, the return
      value would always be logged with the format as above.

      A typical usage of this is that when a stored procedure or
      function is used as an "ACTION" in TIBCO Hawk rule, the stored
      procedure or function return value can be monitored with TIBCO
      Hawk logfile microagent methods.
-->
```

```
<!ATTLIST method
    name        %string;     #REQUIRED
    handlerType (Q|QD|U|UD|I|P|PR|S) #REQUIRED
    index       %string;     #IMPLIED
    timeout     %string;     #IMPLIED
    tableName   %string;     #IMPLIED
    resultSetIndex %string;  #IMPLIED
    useRowNumberAsIndex %string;        #IMPLIED
    useCache       (true|false)         #IMPLIED
    logReturnValue (true|false)         #IMPLIED
    SQL         %string;     #IMPLIED
    help        %string;     #IMPLIED>

<!--
     An argument parameter may have an optional
     either "valueChoices" or "legalValueChoices" but not both.
```

```
      "valueChoices" defines (but not limited to )a set of
      "acceptable" input values.

      "legalValueChoices" defines the only set of "legal"
       values.
-->

<!ELEMENT inputParameter ((valueChoices|legalValueChoices)?)>

<!--
      Except the methods with handlerType "P", the input parameters
      have to be entered in the same order as those appear inthe SQL
      statement.

      The "dateFormat" attribute only applies to parameters with
      the "type" attribute "TIMESTAMP". If the "dateFormat"
      attribute is specified, it would be applied to only this
      parameter which overrides the default "dateFormat".
```

```
      The "pattern" attribute specifies a pattern to convert the
      input text data to a desired form. The pattern syntax is
      based on the JAVA java.text.MessageFormat class specification.
      For example, a pattern "{0, number} KB" could be used to
      convert a string "123 KB" to a number (not a string) 123.
      If the pattern were "{0} KB" in the example above, a string
      of "123" would be extracted.
      Normally only the first argument placeholder (i.e., {0..})
      would be used to extract the value for the parameter, the
      others are ignored. However, if the subsequent parameters have
      the "pattern" attribute equal to "" (empty string), then the
      {1}, {2}, ... {n}
      of the previous parameter with non-empty "pattern" attribute
      will be applied to the subsequent parameters in order.
-->

<!ATTLIST inputParameter
```

```
    name        %string;     #REQUIRED
    type        %valueType;  #IMPLIED
    dateFormat  %string;     #IMPLIED
    pattern     %string;     #IMPLIED
    help        %string;     #IMPLIED>

<!--
     For "valueChoices", and "legalValueChoices",
     the parameter values can be defined using
     the "value" attribute associated with the tag.
     The values are separated by comma characters, for example,

     value = "0, 30, 45, 60, 90"
-->

<!ELEMENT valueChoices (#PCDATA)>

<!ATTLIST valueChoices
```

```
    value %string; #IMPLIED>

<!ELEMENT legalValueChoices (#PCDATA)>

<!ATTLIST legalValueChoices
    value %string; #IMPLIED>

<!--
     Except the methods with handlerType "P", the output parameters
     have to be entered in the same order as the returned columns
     from the SQL statement.

     The "dateFormat" attribute only applies to parameters with
     the "type" attribute "TIMESTAMP".
     If the "dateFormat" attribute is specified, it would be
     applied to only this parameter which overrides the default
     "dateFormat".
```

```
     The usage of "pattern" attribute is the same as those
     described in "inputParameter" above.
-->

<!ELEMENT outputParameter (#PCDATA)>

<!ATTLIST outputParameter
name        %string;     #REQUIRED
type        %valueType;  #IMPLIED
dateFormat  %string;     #IMPLIED
pattern     %string;     #IMPLIED
help        %string;     #IMPLIED>
```

Java Pattern Matching Syntax

This appendix describes the pattern matching syntax used for parsing and converting method input and output parameters. When a pattern is specified in the adapter configuration file, the adapter converts input text data to the specified format.

Pattern Matching Syntax

In parameter definitions, the adapter supports using the pattern matching syntax defined by the `java.text.MessageFormat` class specification. By referencing a pattern in an input or output parameter definition, you can convert the input data to various types and formats.

For more information on `java.text.MessageFormat`, see:

```
http://java.sun.com/products/jdk/1.1/docs/api/java.text.MessageFormat.html
```

Patterns are applied using argument placeholders, which have the following form:

```
{ number, [type], [format] }
```

The curly brackets and *number* argument are required. The *number* argument can be from 0 to 9.

The *type* argument is optional, and indicates the argument data type. It can be one of `time`, `date`, `number`, or `choice`.

The *format* argument is also optional, and is a string that describes the argument format. Possible format values depend on the specified type. For more information on possible values for format, see the Java API documentation for the `ChoiceFormat`, `DateFormat` and `NumberFormat` classes.

You specify the argument placeholder in the `pattern` attribute of the `inputParameter or outputParameter` element. Normally, the first argument placeholder in the pattern is used

and the others are ignored. However, if the subsequent input or output parameters have the pattern attribute equal to `" "` (blank) then the `{1}, {2}, ....{n}` of the previous one parameter with non-blank pattern attribute will be applied to the subsequent parameters in order.

Examples of Using Argument{0} in Patterns

Text	Pattern	Result (Argument {0})
`256 KB`	{0,number}KB	256
`xyz (5%)`	{1}({0,number}%)	5
`Host:smiles-no response`	{1}:{0}-{2}	smiles
Todays's date is 02-02-1996	Todays's date is {0,date}	A `date` object of 2/2/1996

Date and Time Value Syntax

This appendix describes the syntax for formatting date and time values processed by the adapter.

Formatting Date and Time Values

This section describes pattern matching syntax defined by the `java.text.SimpleDateFormat` class specification. Use this syntax to convert date and time values to text, or to parse text and convert to a date format.

For more information on `java.text.SimpleDateFormat`, see:

```
http://www.oracle.com/technetwork/java/index.html#_top_
```

You can specify the pattern syntax for the `dateFormat` attribute of an input or output parameter. A `dateFormat` value specified in the `TIBHAWK_AMI` element is applied to all date and time values processed by the adapter. Values specified in a parameter element override those specified for the adapter. For more information on element and attribute syntax, see Configuring the Adapter.

The following symbols make up the date and time pattern syntax. To specify the time format use a time pattern string. The count of letters determines the format:

- **Text formats**: use four or more letters to present the full form, for example, November; three or less for the abbreviated form, for example, Nov.
- **Number formats**: Is a count of letters determines the minimum length. Leading zeroes are used to pad shorter numbers. The exception is the year format (`y`). If represented as `yy`, the year is truncated to two digits.
- **Text and number formats**: Is a count of three or over, use text or, use number.

In the pattern syntax, all ASCII letters are reserved as pattern letters. Non-alphabetical characters (that is, not a - z or A - Z) included in the pattern are treated as quoted text, that is, they appear literally as you enter them without using single quotes. For example, to

represent a date and time such as `Fri, Nov 19, 1999 10:45` you would enter: `"EEE, MMM dd, yyyy hh:mm"`

Symbol	Description	Presentation
`G`	Era designator GG: AD	Text
`y`	Year `yy`: `99` (last two digits) `yyyy`: `2000`(four digit year)	Number
`M`	Month `MM`: `09` (number) `MMM`: `Nov` (abbreviated form) `MMMM` (or over): `November` (full form)	Text and Number
`d`	Day in month	Number
`h`	Hour of day (1-12)	Number
`H`	Hour of day (1-24)	Number
`m`	Minute in hour	Number
`s`	Second in minute	Number
`E`	Day in week	Text
`D`	Day in year	Number
`F`	Day of week in month. For example, 2nd Wednesday in July.	Number
`w`	Week in year	Number
`W`	Week in month	Number
`a`	am/pm (meridian) marker	Text

Symbol	Description	Presentation
k	Hour in day (1-24)	Number
K	Hour in am/pm (0-11)	Number
z	Time zone	Text
’	(Single quote) Escapes text between display	Delimiter
’’		Literal

Date and Time Format Examples

Format Pattern	Result
`"yyyy.MM.dd G 'at' hh:mm:ss"`	1996.07.03 AD at 15:08:56
`"EEE, MMM d, ''yy"`	Wed, July 4, '96
`"h:mm A"`	5:08 PM
`"hh 'o''clock' A, zzzz"`	12 o'clock PM, Pacific Daylight Time
`"K:mm A"`	0:00 PM
`"yyyyy.MMMMM.dd GGG hh:mm A"`	1996.July.10 AD 12:08 PM

Hawk Service Wrapper for Microsoft Windows

This appendix describes Hawk Service Wrapper for the Microsoft Windows environment.

Microsoft Windows Wrapper

The adapter for the Microsoft Windows platform is packaged as a Windows service. The `tibhawkdbhma.exe` program found in the bin directory of the TIBCO® Operational Intelligence Hawk® RedTail installation implements this Windows service. The `tibhawkdbhma.exe` program is a wrapper program that wraps the adapter Java application so that it can run as a Windows service. The `tibhawkdbhma.exe` program can be used to perform the following functions:

- Install the adapter as a Microsoft Windows service
- Uninstall the adapter as a Microsoft Windows service
- Execute the adapter as a Microsoft Windows console application
- Execute the adapter as a Microsoft Windows service

All the available options can be viewed by executing "tibhawkdbhma --help" command.

Installing the Adapter as a Microsoft Windows Service

The `tibhawkdbhma.exe` program can be installed as an instance of the adapter Microsoft Windows service. As part of the install command processing, the `tibhawkdbhma.exe` program registers itself as the Microsoft Windows service executable. For this reason, the install command should be executed from the directory where the executable is present.

The syntax of the `tibhawkdbhma.exe` program install command is:

```
tibhawkdbhma.exe --install <options>
```

The following table describes the command lines options used:

Option	Description
`-auto`	The service is configured for an auto start. **Note:** Either `manual` or `auto` should be specified. If none is specified, the default is `manual`.
`-account` *accountname*	The user account which runs the service. The default is the System account.
`-password` *password*	The account password.
`-params` *(p_1 p_2 ...p_N)*	The startup parameters for the service. The parentheses are required to distinguish startup parameters from other install command-line options and they must be delimited by blanks.
`-jvm11opts` *(o_1 o_2 ...o_N)*	The Java 1.1 Virtual Machine options. The parentheses are required to distinguish JVM parameters from other install command-line options and they must be delimited by blanks.
`-jvmopts` *(o_1 o_2 ...o_N)*	The Java 2 Virtual Machine options. The parentheses are required to distinguish JVM parameters from other install command-line options and they must be delimited by blanks.

With the `-name` and `-displayname`, multiple instances of the adapter Microsoft Windows service can be installed. Each instance must have a unique -name and should have a unique `-displayname`.

With the `-params()` option, the specification of startup parameters for Hawk® Database Adapter can be done. These startup parameters are registered with the installation and become the default startup parameters for this instance of the adapter Microsoft Windows service. If startup parameters are specified when this service is actually started they completely override these default values.

With the `-jvm11opts()` option, the specification of Java 1.1 Virtual Machine (JVM) options to be passed to the JVM at execution time can be done. This option supports all the Java 1.1 runtime options. These JVM options are platform specific. See the Java runtime executable (`jre.exe`) help for details on the options available on your platform.

With the `-jvmopts()` option, the specification of Java 2 Virtual Machine (JVM) options to be passed to the JVM at execution time can be done. This option supports all the Java 2 runtime options. These JVM options are platform specific. See the Java runtime executable (`java.exe`) help for details on the options available on your platform. This option is only required if you want to specify Java 2 specific options.

The `tibhawkdbhma.exe` program automatically detects the version of the JVM at execution time and uses the appropriate JVM options according to the following table:

JVM Options Specified	Running Under JVM 1.1	Running Under JVM 1.2 or 1.3
None	JVM defaults used	JVM defaults used
`-jvm11opts` only	`-jvm11opts` used	`-jvm11opts` used
`-jvmopts` only	JVM defaults used	`-jvmopts` used
`-jvm11opts` and `-jvmopts`	`-jvm11opts` used	`-jvmopts` used

Java Classpath

The `tibhawkdbhma.exe` program automatically constructs the Java classpath for the adapter Microsoft Windows service. Assuming that the `tibhawkdbhma.exe` program is located in directory <HAWK_HOME> directory, which is typically `c:\tibco\hawk\bin`, the constructed classpath consists of the following in the specified order:

- <TIBCO_HOME>`\bin\..\java`
- All `.jar` files located in the `c:\tibco\hawk\bin\..\java` directory.
- The Java Runtime default classpath.
- The CLASSPATH environment variable.
- <TIBCO_HOME>\adapters\dhma\security\tibcrypt.jar

This classpath can be altered by specifying the `-cp` and `-classpath` options in the `-jvm11opts` and `-jvmopts`. The classpath specified in the `-cp` option is used as a prefix to the classpath described above. The classpath specified in the `-classpath` option replaces (overrides) the classpath described previously.

Additional classpath related properties can be specified for the Java 2 platform. See the Java documentation for details. The `tibhawkdbhma.exe` program passes the classpath described above, to the Java 2 virtual machine using the `–Djava.class.path` property. The user should not specify this property directly but use the `-cp` and `-classpath` options to alter the classpath.

Note: The detailed command for installing the adapter as a Microsoft Windows service is complicated. Therefore, you can use the sample batch file, such as `<database>_sample1.bat` provided in the `examples` directory as a template to write your own to start your custom adapters.

Uninstalling the Adapter as a Microsoft Windows Service

An instance of the adapter Microsoft Windows service can be removed using the remove command-line option. The syntax of the command is:

`tibhawkdbhma.exe --remove –name <servicename>`

Where the <*servicename*> is the internal service name used when the service was installed.

The `--remove` command automatically stops the Microsoft Windows service if it is running prior to uninstalling it.

Executing the Adapter as a Microsoft Windows Console Application

The adapter can be executed as a Microsoft Windows console application from the command line using the `tibhawkdbhma.exe` program. Running the adapter as a console application is useful when diagnosing problems or when using it with a debugger. The command for executing the adapter is:

`tibhawkdbhma.exe --name <servicename> -xml_file <filename> [-check_only]`

Where *<servicename>* is the internal service name used when the service is installed, and *<filename>* is the complete pathname of the XML configuration file.

The command for specifying the **`-help`** option to display the command line syntax for the adapter is:

```
tibhawkdbhma.exe -help/-h/-? <options>
```

Note: The Microsoft Windows service for the adapter with the specified *<servicename>* must be installed before an adapter can be executed as a Windows console application.

Executing the Adapter as a Microsoft Windows Service

To start the service:

1. In Microsoft Windows Control Panel, open **Services**.
2. Click *<displayname>* **,** then click **Start.**

   The *<displayname>* is the external name given to this instance of the adapter when the service was installed.

The adapter service starts.

Sample Programs

This appendix describes the samples provided in the samples directory for each of the databases supported. These samples can be installed and registered as Windows Services or background tasks on non-Windows platforms. Scripts (bat/sh) are provided for each of these samples. Refer to *TIBCO Hawk® Database Adapter Installation Guide* on how to start the adapter before you start these samples as services or background jobs.

sample1.xml

The `Record Hawk Event` method contained in `sample1.xml` parses the TIBCO Hawk 'Event log' file entries to extract several key fields and then stores them in separate columns in the database. This method is invoked as an 'action' of a TIBCO Hawk rule. The, `logFile` microagent `onNewLine` method, which monitors the TIBCO Hawk 'Event log' file, is used as the 'data source'. The data returned from the `onNewLine` method, that is, every new line added to the ’Event log’ file is passed as the first input parameter in this method. The 'pattern' associated with the input parameter is used to extract the desired elements out from the log entry and convert them to the desired datatype.

After the data is converted, using the specified pattern, and stored in separate columns in the database, it can be manipulated easily using relational database queries. For example, the data can be sorted by time, or ordered by a number type column. TIBCO Hawk Database Adapter 'query' methods can also be created to monitor these tables. This approach can be used to monitor and parse custom application log file entries as well.

Typically, in TIBCO Hawk rules, 'tests' could be used to screen the log entries (based on the text pattern), and then set up some corresponding Hawk® Database Adapter 'insert' methods, like the one in this example, can be used to store data in specific tables. For example, if an application log file contains entries like

```
"Warning 987: Low buffer space - buffer type = socketRead, remaining
available space = 268 KB.".
```

In a rule, a test can be used to select (screen) the log entries starting with '`Warning 987:`'. An adapter 'insert' method can be used to extract the string "`socketRead`" and the number `268` and to store them in separate columns in a specific table. There could be another rule to monitor those columns and produce an alert when that number drops below some threshold, say, `100`. The data stored in those columns could also be used for some trend analyses, say, to find the pick hour during a day.

The *<database>*`_sample1.hrb` contains sample rulebases associated with the methods defined in `sample1.xml`

For more information on using patterns to convert input text data into another form, see TIBCO Hawk Database Adapter Document Type Definition (DTD).

sample2.xml

This example shows how to configure methods using simple query, simple update and calling functions or stored procedures in many different ways.

The `interop_sample2.hrb` contains sample rules associated with the methods defined in the `sample2.xml`. In these rules, the 'data source' methods and the 'action' methods are from two different adapters connected to two different databases. This example demonstrates the interoperability between the Oracle and Sybase databases using adapters.

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

Documentation for TIBCO Hawk® Database Adapter is available on the TIBCO Hawk® Database Adapter Product Documentation page.

To directly access documentation for this product, double-click the following file:

TIBCO_HOME`/release_notes/TIB_hawkdbhma_6.2.1_docinfo.html` where *TIBCO_HOME* is the top-level directory in which TIBCO products are installed. On Windows, the default *TIBCO_HOME* is `C:\tibco`. On UNIX systems, the default *TIBCO_HOME* is `/opt/tibco`.

The following documents for this product can be found in the TIBCO Documentation site:

- *TIBCO Hawk® Database Adapter Release Notes*
- *TIBCO Hawk® Database Adapter Installation*
- *TIBCO Hawk® Database Adapter User's Guide*
- *TIBCO Hawk® Database Adapter Security Guide*

How to Contact TIBCO Support

Get an overview of TIBCO Support. You can contact TIBCO Support in the following ways:

- For accessing the Support Knowledge Base and getting personalized content about

products you are interested in, visit the TIBCO Support website.

- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to TIBCO Support website. If you do not have a user name, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to TIBCO Community.

Legal and Third-Party Notices