ACI API

Software Version: 11.0

Programming Guide

Document Release Date: February 2018

Software Release Date: February 2018

Legal notices

Warranty

The only warranties for Seattle SpinCo, Inc. and its subsidiaries ("Seattle") products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Seattle shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice.

Restricted rights legend

Confidential computer software. Except as specifically indicated, valid license from Seattle required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright notice

Trademark notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

Documentation updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To verify you are using the most recent edition of a document, go to https://softwaresupport.softwaregrp.com/group/softwaresupport/search-result?doctype=online help.

This site requires you to sign in with a Software Passport. You can register for a Passport through a link on the site.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your Micro Focus sales representative for details.

Support

Visit the Micro Focus Software Support Online website at https://softwaresupport.softwaregrp.com.

This website provides contact information and details about the products, services, and support that Micro Focus offers.

Micro Focus online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support website to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Access the Software Licenses and Downloads portal
- Download software patches
- Access product documentation
- Manage support contracts

- Look up Micro Focus support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require you to register as a Passport user and sign in. Many also require a support contract.

You can register for a Software Passport through a link on the Software Support Online site.

To find more information about access levels, go to https://softwaresupport.softwaregrp.com/web/softwaresupport/access-levels.

Contents

Part I: Introduction

This section introduces the Micro Focus ACI API and includes the following topics:

Chapter 1: Introduction to the ACI API

The ACI (Autonomy Content Infrastructure) Client API enables easy communication between custom-built applications and Micro Focus ACI servers, as well as simple manipulation of the returned result sets.

API Versions

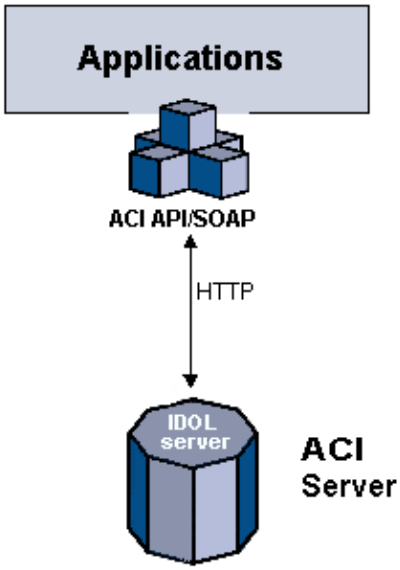
This version of the ACI API Programming Guide is relevant for the following versions of the IDOL SDKs:

- Java Client NG IDOL SDK version 6.0
- C IDOL SDK version 11.0
- .NET IDOL SDK version 10.8

System Architecture

The ACI API uses HTTP to allow custom-built applications (for example C, Java, or .NET applications) to communicate with Micro Focus ACI servers. Communication with the servers is implemented over HTTP using XML and can adhere to SOAP. The following diagram shows the architecture.

Application Server Architecture



Use the ACI API

You can use the ACI API to send actions to ACI servers and retrieve information from their responses. Each IDOL SDK provides APIs that allow you to:

- Configure connections to ACI servers.
- Send requests to ACI servers

For details on the available actions and the parameters they require, refer to the Administration Guide or Reference for the appropriate Micro Focus ACI server.

IDOL SDKs

IDOL SDKs are available for C, Java, and .NET. The SDK to use for your application depends on the application environment, as identified in the following table.

Language Environments

ACI API	Environment
C	C
Java Client NG	• Standalone Java • J2EE • JSP
.NET	.NET (for example, Visual Basic, C#, J#)

Micro Focus ACI Server Action Syntax

You can use either HTTP GET or HTTP POST methods to access Micro Focus ACI servers. The following sections describe syntax differences between the two methods and show examples of each. The ACI API clients construct these requests for you.

HTTP GET

The ACI API allows custom-built applications to access Micro Focus ACI servers using the following HTTP action syntax:

`http://host:port/action=action&parameters`

where,

host	is the IP address (or name) of the machine on which the Micro Focus ACI server is running.
port	is the port number that is used to send actions to the Micro Focus ACI server.
action	is the action that you want the Micro Focus ACI server to run.
parameters	are the required and optional parameters for the action.

Example:

Consider the following example:

```
http://localhost:4000/action=query&text=dinosaurs&maxresults=1
```

This action sends the query text `dinosaur` to an Micro Focus ACI server (in this case IDOL server), which in response returns an XML result, for example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<autnresponse>
   <action>QUERY</action>
   <response>SUCCESS</response>
   <responsedata>
      <autn:numhits>1</autn:numhits>
      <autn:hit>
         <autn:reference>
           http://c.moreover.com/click/here.pl?z16358245&z=28
         </autn:reference>
         <autn:id>101927</autn:id>
         <autn:section>0</autn:section>
         <autn:weight>97</autn:weight>
         <autn:links>DINOSAUR</autn:links>
         <autn:database>0</autn:database>
         <autn:title>Studying dinosaurs</autn:title>
```

```
      </autn:hit>
   </responsedata>
</autnresponse>
```

HTTP POST

The ACI API allows custom-built applications to access Micro Focus ACI servers using the following HTTP action syntax to post the body to *host*:*port*.

`action=`*action*↵

parameter1=parameter_value↵

parameter2=parameter_value↵

> **NOTE:**
> ↵ is a CRLF combination or a CR.

where,

host	is the IP address (or name) of the machine on which the Micro Focus ACI server is running.
port	is the port number that is used to send actions to the Micro Focus ACI server.
action	is the action that you want the Micro Focus ACI server to run.
parameterN	is the name of a required or optional parameter for the action.
parameter_value	is the associated parameter value.

Example:

Consider the following example that posts the following body to `localhost:4000`:

```
action=query
querytext=dinosaurs
maxresults=1
```

This action sends the query text `dinosaur` to an Micro Focus ACI server (in this case an IDOL server), which in response returns an XML result, for example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<autnresponse>
<action>QUERY</action>
<response>SUCCESS</response>
<responsedata>
   <autn:numhits>1</autn:numhits>
   <autn:hit>
      <autn:reference>
```

```
        http://c.moreover.com/click/here.pl?z16358245&z=28
      </autn:reference>
      <autn:id>101927</autn:id>
      <autn:section>0</autn:section>
      <weight>97</weight>
      <links>DINOSAUR</links>
      <database>0</database>
      <autn:title>Studying dinosaurs</autn:title>
   </autn:hit>
</responsedata>
</autnresponse>
```

Display Online Reference for ACI Server Actions

The ACI server online references contain details of the available actions and configuration parameters for an ACI server. You can use the references to find the details of the actions and parameters that you want to call with the ACI API.

You can view the reference for a particular ACI server by sending the following action to the server:

`http://`*`host`*`:`*`port`*`/action=Help`

where,

host	is the IP address (or name) of the machine on which the Micro Focus ACI server is running.
port	is the port number that client machines use to communicate with the Micro Focus ACI server. This port number is specified in the ACI server configuration file `[Server]` section.

> **NOTE:**
> To use the reference, the ACI server must have a `help.dat` file available in the same directory as the server executable file.

The ACI Representation of XML

> **NOTE:**
> The following section is not relevant to the .NET IDOL SDK.

Each XML entity in an XML document is represented by a `Data` ACI object (for the Java SDK, this is an `AciResponse` object), which holds the name of the XML entity and its value (if the XML entity contains plain text between its opening and closing tags, this text is considered to be its value, otherwise no value is stored).

The hierarchical structure of an XML document is preserved through a linked list. For example, if an XML entity contains another XML entity, the ACI object that represents the first XML entity represents the enclosed XML entity through a link to a child ACI object. If an XML entity contains multiple XML entities, the corresponding ACI objects are linked to each other, as well as to the parent ACI object. Each ACI object can point to a child branch and a sibling branch.

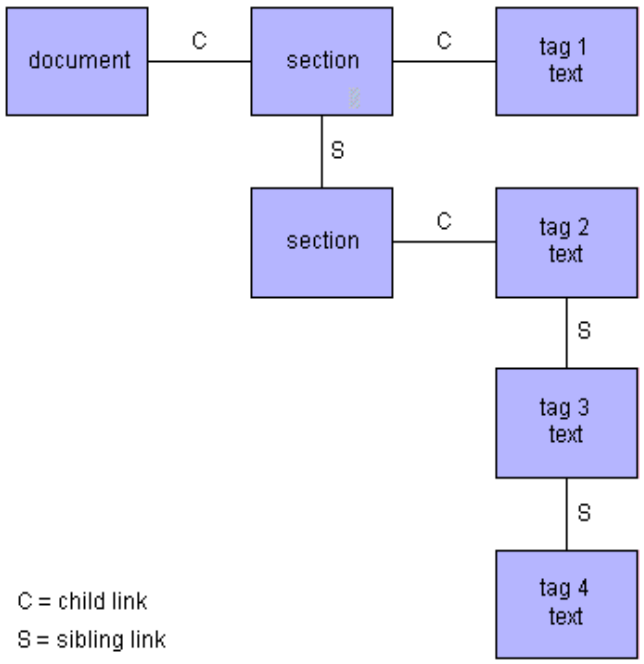
Consider the following example:

```
<document>
  <section>
    <tag 1>text</tag 1>
  </section>
  <section>
    <tag 2>text</tag 2>
    <tag 3>text</tag 3>
    <tag 4>text</tag 4>
  </section>
</document>
```

The XML fragment below is converted to the ACI object linked list as shown in the following diagram.

ACI objects representing XML in linked list

```
<document>
   <section>
      <tag 1>text</tag 1>
   </section>
   <section>
      <tag 2>text</tag 2>
      <tag 3>text</tag 3>
      <tag 4>text</tag 4>
   </section>
</document>
```



Part II: Java Language Interface

This section provides information on using the Java language IDOL SDK and includes the following chapters:

Chapter 2: Use the Java Client NG IDOL SDK

This section describes common ways to use the Java Client NG IDOL SDK.

Java ACI Objects

In the Java Client NG IDOL SDK, the main ACI API functionality is provided by the following objects:

- `AciServerDetails` allows you to set the connection details of an ACI server including encryption settings.
- `AciService` and `AciParameter` objects allow you to construct sets of action parameters and send them to a configured ACI server.
- `DocumentProcessor` and `BinaryResponseProcessor` allow you to process the response from any ACI action into an easy to use, standards compliant, format.
- `Processor<T>` and `AbstractStAXProcessor<T>` allow you to write you own custom ACI response parsers for maximum flexibility.

Create an AciService

All communication with ACI Servers goes through an implementation of the `AciService` interface. To allow for the usage of different HTTP clients, the default implementation of the `AciService` interface requires an implementation of the `AciHttpClient` interface. The default implementation of the `AciHttpClient` interface uses the Apache HTTP Components `HttpClient`, which is included in the distribution.

When creating an `AciService`, you can either specify connection details for a particular server and tie the implementation to that server, or you can specify connection details every time you send an action. Connection details that you specify on a per action basis override those set on the default `AciService` implementation.

The following example shows the simplest way to create an `AciService` implementation:

```
// Create an AciService...
final AciService aciService = new AciServiceImpl(
      new AciHttpClientImpl(new DefaultHttpClient()),
      new AciServerDetails(host, port)
);
```

For more control over how the Apache `HttpClient` is configured, the API contains a factory class, `HttpClientFactory`, for creating and configuring the `HttpClient`. For information about how to configure and use this class, refer to the JavaDocs.

The only way to switch between GET and POST HTTP methods, is to set a flag on the `AciHttpClientImpl` class. The way that this class is normally created means that an `AciService` implementation is normally set to either send everything as GET or everything as POST.

The `AciServerDetails` class allows you to set the HTTP protocol (HTTP or HTTPS), as well as the host, port, character set and any encryption details. It supports only ACI servers secured via the BTEA algorithm. Micro Focus recommends that you use SSL by HTTPS.

The following example shows how to configure an `AciServer` instance when you are using a dependency injection container like Spring, you can configure an `AciServer` instance:

```
<beans xmlns="...">

  <context:property-placeholder location="classpath:idol.properties" />

  <bean id="httpClientFactory"
      class="com.autonomy.aci.client.transport.impl.HttpClientFactory"
      p:maxConnectionsPerRoute="20"
      p:maxTotalConnections="120"
      p:staleCheckingEnabled="true" />

  <bean id="httpClient" factory-bean="httpClientFactory"
      factory-method="createInstance" />

  <bean id="aciService"
        class="com.autonomy.aci.client.services.impl.AciServiceImpl">
    <constructor-arg>
      <bean class="com.autonomy.aci.client.transport.impl.AciHttpClientImpl"
          p:httpClient-ref="httpClient" />
    </constructor-arg>
    <constructor-arg>
      <bean class="com.autonomy.aci.client.transport.AciServerDetails"
          p:host="${idol.host}"
          p:port="${idol.aciPort}" />
    </constructor-arg>
  </bean>
</beans>
```

Alternatively, if you are using Spring Java Configuration, you can use something similar to the following example:

```
@Configuration
@EnableWebMvc
public class AppConfig extends WebMvcConfigurerAdapter {

    @Autowired
    private Environment env;
```

```
    @Bean
    public ResourceBundle idolProperties() {
        return ResourceBundle.getBundle("/idol");
    }

    @Bean
    public HttpClient httpClient() {
        final HttpClientFactory factory = new HttpClientFactory();
        factory.setMaxTotalConnections(toInt(env.getProperty(
              "aci.maxTotalConnections"), 30));
        factory.setMaxConnectionsPerRoute(toInt(env.getProperty(
              "aci.maxConnectionsPerRoute"), 15));
        factory.setConnectionTimeout(toInt(env.getProperty(
              "aci.connectionTimeout"), 10000));
        factory.setLinger(toInt(env.getProperty("aci.linger"), -1));
        factory.setSocketBufferSize(toInt(env.getProperty(
              "aci.socketBufferSize"), 8192));
        factory.setSoTimeout(toInt(env.getProperty("aci.soTimeout"), 30000));
        factory.setStaleCheckingEnabled(toBoolean(env.getProperty(
              "aci.staleChecking", "true")));
        factory.setTcpNoDelay(toBoolean(env.getProperty(
              "aci.tcpNoDelay", "true")));
        factory.setUseCompression(toBoolean(env.getProperty(
              "aci.useCompression", "false")));
        return factory.createInstance();
    }

    @Bean
    public AciService aciService() {
        final ResourceBundle idolProperties = idolProperties();
        final AciServerDetails serverDetails = new AciServerDetails();
        serverDetails.setHost(idolProperties.getString("idol.host"));
        serverDetails.setPort(NumberUtils.toInt(
              idolProperties.getString("idol.aciPort"), 9030));

        return new AciServiceImpl(
              new AciHttpClientImpl(httpClient()), serverDetails);
    }
}
```

Perform an Action

After you create your `AciService`, the next thing you need to do to send an action to an ACI Server, is to produce a set of parameters. You can use any kind of `Set` implementation, and there is a helper class, `ActionParameters`, with various methods to make it easier to populate.

If you were sending a `Query` action to IDOL Server, you would create your `ActionParameters` like:

```
// Create the parameter set...
final ActionParameters parameters = new ActionParameters();
parameters.add(AciConstants.PARAM_ACTION, "Query");
parameters.add("Text", queryText);
parameters.add("combine", "simple");
parameters.add("maxResults", 10);
parameters.add("totalResults", true);
parameters.add("print", "none");
parameters.add("summary", "ParagraphConcept");
parameters.add("characters", 400);
parameters.add("anyLanguage", true);
parameters.add("outputEncoding", "utf8");
```

You can also specify parameters to the constructor, using `varargs` for example:

```
// Create the parameter set...
final ActionParameters parameters = new ActionParameters(
      new AciParameter(AciConstants.PARAM_ACTION, "Query"),
      new AciParameter("text", queryText),
      new AciParameter("combine", "simple"),
      new AciParameter("maxResults", 10),
      new AciParameter("totalResults", true),
      new AciParameter("print", "none"),
      new AciParameter("summary", "ParagraphConcept"),
      new AciParameter("characters", 400),
      new AciParameter("anyLanguage", true),
      new AciParameter("outputEncoding", "utf8")
);
```

The following example also shows a shorthand way to create your `ActionParameters`, where all you need is the action parameter:

```
final ActionParameters params = new ActionParameters("getstatus");
```

When you are passed a `Set<ActionParameter>`, you can easily convert this into an `ActionParameters` object by using the `ActionParameters.convert(Set<? extends ActionParameter>)` method. If the method is passed an `ActionParameters` object, it casts it back and returns it, without creating a new copy.

The default behavior of a `Set` is to not add something if it already exists in the set, so `ActionParameters` contains a series of `put(AciParameter)` methods, which remove the existing value and replace it with the one passed. This option ensures that certain parameters are set to required values, regardless of what has been passed. For example:

```
private void myMethod(final Set<ActionParameter<?>> parameterSet) {
    // Convert the set of parameters...
    final ActionParameters parameters = ActionParameters.convert(parameterSet);

    // Ensure that these parameters are set to the values that are required...
    parameters.put("combine", "simple");
    parameters.put("anyLanguage", true);
    parameters.put("outputEncoding", "utf8");
```

```
    ...
}
```

After you have your set of parameters, you can use the previously configured `AciService` to send them to your ACI Server. For example:

```
// Send the action and process the response into a DOM Document...
final Document response = aciService.executeAction(
      parameters, new DocumentProcessor());
```

As previously stated, you can also supply connection details at this point to override any already set on the default `AciService` implementation. For example:

```
// Send the action and process the response into a DOM Document...
final Document response = aciService.executeAction(
      new AciServerDetails(
            AciServerDetails.TransportProtocol.HTTPS, host, port),
      new ActionParameters("GetVersion"),
      new DocumentProcessor()
);
```

Process an ACI Response

The API uses a processor based system to consume the responses from an ACI Server. It is based around two interfaces: `Processor<T>` and `StAXProcessor<T>`. `StAXProcessor<T>` has an abstract implementation that sets up the StAX parser. You can either write your own processor implementations, or use those that are included with the API.

The default processor that is included with the API is the `DocumentProcessor`. This processor returns a DOM Document object, which you can then parse using either DOM or XPath. However, not all ACI responses are XML. For example, the cluster and spectrograph functionality can also return images. The SDK contains a `BinaryResponseProcessor` for these cases. Internally it uses the `ByteArrayProcessor` to process the binary data, or the `ErrorProcessor` if it detects that the response is actually XML.

The following example shows the use of the `DocumentProcessor` and `XPath` to output query results to the console:

```
// Send the action and process the response into a DOM Document...
final Document response = aciService.executeAction(
      parameters, new DocumentProcessor());

// Get the individual documents in the response...
final XPath xpath = XPathFactory.newInstance().newXPath();
final NodeList nodeList = (NodeList) xpath.evaluate(
      "/autnresponse/responsedata/hit",
      response,
      XPathConstants.NODESET
);
final int documents = nodeList.getLength();
```

```
System.out.println(
      "Displaying " + documents +
      " results from a total of " +
      xpath.evaluate("/autnresponse/responsedata/totalhits", results) + '\n'
);

// Output the details of each document to the console...
for(int ii = 0; ii < documents; ii++) {
    // Get an individual document in the response...
    final Node node = nodeList.item(ii);

    // Output a few key fields...
    System.out.println('\n' + xpath.evaluate("reference", node));
    System.out.println("Title     : " + xpath.evaluate("title", node));
    System.out.println("Relevance : " + xpath.evaluate("weight", node) + '%');
    System.out.println("Database  : " + xpath.evaluate("database", node));
    final String links = xpath.evaluate("links", node);
    if(StringUtils.isNotBlank(links)) {
        System.out.println("Matched on: " + links);
    }
    System.out.println("Summary   : " + xpath.evaluate(
          "summary", node).replaceAll("\n", " "));
}
```

The following example shows the use of the `BinaryResponseProcessor` in a Spring based service implementation:

```
@Service
public class ClusteringServiceImpl implements ClusteringService {

    @Autowired
    @Qualifier("clusterService")
    private AciService clusteringService;

    ...

    @Override
    public byte[] clusterServe2DMap(final String job) {
        return clusteringService.executeAction(
              new ActionParameters(
                    new AciParameter(
                          AciConstants.PARAM_ACTION, "ClusterServe2DMap"),
                    new AciParameter("sourceJobName", job)
              ),
              new BinaryResponseProcessor()
        );
    }
```

```
    ...
}
```

You might want to write your own response processors for increased performance, or to return the response as a collection of `JavaBeans`. In this case, you simply implement one of the two processor interfaces. When you need access to the underlying response stream, implement the `Processor<T>` interface. When you want to parse XML, then extend the `AbstractStAXProcessor<T>` class, which implements the `StAXProcessor<T>` interface.

The following example shows how to return the first instance of a field in a response. This process is useful for pulling out information such as `SecurityInfo` strings, where you do not need the rest of the response.

```
public class ElementValueProcessor extends AbstractStAXProcessor<String> {

    private final String elementName;

    public ElementValueProcessor(final String elementName) {
        this.elementName = elementName;
        setErrorProcessor(new ErrorProcessor());
    }

    @Override
    public String process(final XMLStreamReader xmlStreamReader) {
        try {
            if(isErrorResponse(xmlStreamReader)) {
                processErrorResponse(xmlStreamReader);
            }

            String result = null;

            while(xmlStreamReader.hasNext()) {
                final int elementType = xmlStreamReader.next();

                if((elementType == XMLEvent.START_ELEMENT) &&
                        (elementName.equals(xmlStreamReader.getLocalName()))) {
                    result = xmlStreamReader.getElementText();
                    break;
                }
            }

            return result;
        }
        catch(XMLStreamException e) {
            throw new ProcessorException(
                  "Unable to parse the ACI response.", e);
        }
    }

}
```

The following example processes the response from the `UserReadUserList` action into a `UserList` JavaBean.

```
public class UserReadUserListProcessor extends AbstractStAXProcessor<UserList> {

    public UserReadUserListProcessor() {
        setErrorProcessor(new ErrorProcessor());
    }

    @Override
    public UserList process(final XMLStreamReader xmlStreamReader) {
        try {
            if(isErrorResponse(xmlStreamReader)) {
                processErrorResponse(xmlStreamReader);
            }

            final UserList userList = new UserList();

            while(xmlStreamReader.hasNext()) {
                final int eventType = xmlStreamReader.next();

                // We want the start elements...
                if(XMLEvent.START_ELEMENT == eventType) {
                    if("autn:user".equalsIgnoreCase(
                            xmlStreamReader.getLocalName())) {
                        userList.addUserName(xmlStreamReader.getElementText());
                    }
                    else if("autn:totalusers".equalsIgnoreCase(
                          xmlStreamReader.getLocalName())) {
                        userList.setTotalUsers(NumberUtils.toInt(
                              xmlStreamReader.getElementText()));
                    }
                    else if("autn:numusers".equalsIgnoreCase(
                          xmlStreamReader.getLocalName())) {
                        userList.setNumUsers(NumberUtils.toInt(
                              xmlStreamReader.getElementText()));
                    }
                }
            }

            return userList;
        }
        catch(XMLStreamException xmlse) {
            throw new ProcessorException(
                    "Unable to create a list of users.", xmlse);
        }
    }

}
```

JSP Tag Library

The Java Client NG IDOL SDK has a JSP tag library to provide an alternative method for sending actions.

The SDK includes documentation for these tags, as well as JavaDocs. For more information about the tag library, refer to these documents in your Java Client NG IDOL SDK installation package.

The following example displays the current memory report for an ACI server.

```
<c:catch var="memoryReportException">
 <aci:action var="memoryReportResult" action="MemoryReport"
     aciService="${aciService}" />
</c:catch>

<c:choose>
  <c:when test="${empty memoryReportException}">
   <div id="memoryReport" class="content">
     <div class="title"><fmt:message key="memoryReport.title" /></div>
      <table class="data marginTop10px">
        <tbody>
          <c:set var="bytes">
            <x:out
              select="$memoryReportResult/autnresponse/
                responsedata/memory0/memoryusage" />
          </c:set>
          <tr>
            <td class="bolder"><fmt:message key="memoryReport.total" /></td>
            <td>
              <fmt:formatNumber value="${(bytes div 1024) div 1024}"
                  type="number" pattern="###.##" />
              <fmt:message key="memoryReport.mb" />
            </td>
          </tr>
          <x:forEach
              select="$memoryReportResult/autnresponse/responsedata/memory0//*[
                starts-with(name(),'memory')
                and not(starts-with(name(),'memoryusage'))]">
            <tr>
              <x:set var="padding" select="count(ancestor::*)" />
              <td class="bolder" style="padding-left: ${(padding - 2) * 25}px">
                <x:out select="name" />
              </td>
              <td>
                <c:set var="usage1"><x:out select="memoryusage" /></c:set>
                <c:choose>
                  <c:when test="${((usage1 div 1024) div 1024) gt 1}">
                    <fmt:formatNumber value="${(usage1 div 1024) div 1024}"
                        type="number" pattern="###.##" />
```

```
                    <fmt:message key="memoryReport.mb" />
                  </c:when>
                  <c:when test="${(usage1 div 1024) gt 1}">
                    <fmt:formatNumber value="${usage1 div 1024}"
                        type="number" pattern="###.##" />
                    <fmt:message key="memoryReport.kb" />
                  </c:when>
                  <c:otherwise>${usage1}
                    <fmt:message key="memoryReport.b" />
                  </c:otherwise>
                </c:choose>
              </td>
              <td>
                <x:if select="components[text()!='0']">
                  <c:set var="usage2">
                    <x:out select="noncomponentusage" />
                  </c:set>
                  <c:choose>
                  <c:when test="${((usage2 div 1024) div 1024) gt 1}">(
                    <fmt:formatNumber value="${(usage2 div 1024) div 1024}"
                        type="number" pattern="###.##" />
                    <fmt:message key="memoryReport.mb" />)
                  </c:when>
                  <c:when test="${(usage2 div 1024) gt 1}">(
                    <fmt:formatNumber value="${usage2 div 1024}"
                        type="number" pattern="###.##" />
                    <fmt:message key="memoryReport.kb" />)
                  </c:when>
                  <c:otherwise>(${usage2}
                    <fmt:message key="memoryReport.b" />)
                  </c:otherwise>
                </c:choose>
                </x:if>
              </td>
            </tr>
          </x:forEach>
        </tbody>
      </table>
    </div>
  </c:when>
  <c:otherwise>
    <div class="content error">
      <fmt:message key="memoryReport.error">
        <fmt:param value="${memoryReportException.message}" />
      </fmt:message>
    </div>
  </c:otherwise>
</c:choose>
```

Part III: .NET Language Interface

This section provides information on using the .NET language IDOL SDK.

Chapter 3: Use the .NET Interface

This section describes common ways to use the IDOL SDK for .NET.

> **NOTE:**
> The classes defined in the .NET IDOL SDK are in the `Autonomy.Aci` namespace. To use the examples in this document you must include the directive:
>
> ```
> using Autonomy.Aci;
> ```


.NET IDOL SDK Objects

In the .NET IDOL SDK, the main functionality is provided by the following objects:

- The `AciClient` class provides methods for creating new connections.
- `Command` objects represent requests to an ACI server. They can be run by any connection returned by `AciClient`.
- `Response` objects represent responses from an ACI server. They provide access to the XML of the response, which is represented by a `System.Xml.XmlDocument` object.

Create a Connection

To send actions to an ACI server, you must first create a connection. When you create the connection, you specify the host and port of the server you want to send the actions to. You can specify other connection settings, such as the number of times to retry a request that fails before abandoning it. For an example of how to create a connection, see Create a Connection Using the .NET IDOL SDK., below.

Create a Connection Using the .NET IDOL SDK.

The following example shows how to create various types of connection.

```
// Create an unencrypted connection using the default settings.
IConnection unsecuredConnection =
    AciClient.CreateUnsecuredConnection("localhost", 9900);

// Create a connection secured using BTEA encryption.
uint[] key = { 123, 234, 345, 456 };
IConnection encryptedConnection =
```

```
    AciClient.CreateBteaConnection("localhost", 9900, key);

// Create an unencrypted connection, overriding the number of
// retries.
HttpSettings settings = new HttpSettings();
settings.Retries = 10;
IConnection anotherConnection =
    AciClient.CreateUnsecuredConnection(
        "localhost", 9900, settings);
```

Perform an Action

To send an action to the ACI server, you must create a `Command` object. This object stores information about the action that you want to run, including all the parameters that the action requires. For an example of how to create a `Command` object, see Create an Action, below.

Create an Action

The following example shows how to create a `Command` object for the IDOL server query action. The action is `Query` and it sets the parameters `Text`, `LanguageType`, and `TotalResults`.

```
Command query = new Command("query");
query.Set("text", "Reducing heart attack risk");
query.Set("LanguageType", "English");
query.Set("TotalResults", true);
```

Manipulate a Response

The `Execute` method of an `IConnection` sends an action to an ACI server and returns a structure containing the response.

For an example of how to obtain this structure, see Obtain a Response to a Query, below.

Obtain a Response to a Query

The following example shows how to run a command and retrieve the response.

```
Response response = connection.Execute(query);
```

If the request is successful, `response.Data` provides access to an `XMLDocument` containing the response from the server. If the request failed, an exception is thrown from `connection.Execute`.

Consider the following example, which might be the result of the `Query` action performed in Create an Action, above:

```
<autnresponse>
  <action>QUERY</action>
  <response>SUCCESS</response>
  <responsedata>
```

```
    <autn:numhits>2</autn:numhits>
    <autn:hit>
      <autn:reference>http://www.example.com/2</autn:reference>
      <autn:id>8</autn:id>
      <autn:section>0</autn:section>
      <autn:weight>96.00</autn:weight>
      <autn:database>Default</autn:database>
    </autn:hit>
    <autn:hit>
      <autn:reference>http://www.example.com/1</autn:reference>
      <autn:id>7</autn:id>
      <autn:section>0</autn:section>
      <autn:weight>96.00</autn:weight>
      <autn:database>Default</autn:database>
    </autn:hit>
  </responsedata>
</autnresponse>
```

For an example of how to obtain a list of the references returned by the `Query` action, see Obtain Response Data From an XML Document, below.

Obtain Response Data From an XML Document

The following example shows how to obtain a list of references returned by the `Query` action by selecting the reference of each hit in the ACI response

```
XmlNamespaceManager namespaces =
    new XmlNamespaceManager(response.Data.NameTable);
namespaces.AddNamespace(
    "autn", "http://schemas.autonomy.com/aci/");
XmlNodeList nodes = response.Data.SelectNodes(
    "/autnresponse/responsedata/autn:hit/autn:reference",
    namespaces);
foreach (XmlNode node in nodes)
{
    Console.WriteLine(node.InnerText);
}
```

Part IV: C Language Interface

This section provides information on using the C language IDOL SDK and includes the following chapters:

Chapter 4: Use the C Interface

This chapter describes common ways to use the C-language IDOL SDK.

C ACI Objects

The fundamental construct in the C IDOL SDK is the ACI object. ACI objects have the following types:

- `Connection` objects allow you to represent details of the ACI server.
- `Command` objects allow you to set details of the requests made to the server; for example, the server action and the parameters that it requires.
- `Data` objects allow you to access the HTML or XML reply to a server request as a plain text buffer (for later parsing or outputting to a browser, for example) or to parse and map XML replies to a linked list of ACI objects. The API provides functions for user-friendly accessing and manipulating the ACI object linked list.

ACI Object Structure

Each ACI object contains:

- a value indicating its type (`Connection`, `Command` or `Data`)
- a collection of key-value pairs used to store parameters and their values
- a link to a child ACI object
- a link to the next ACI object in the linked list (sibling object)

Create a Connection

You must create a connection object (type `ACI_CONNECTION`) to establish a connection with the server. You must set the server host details in this object before making a request. You can specify other connection settings, such as the number of times a connection to the server is attempted and fails before the request is abandoned. For an example of how to create connections, see Create a Connection Using the C-language IDOL SDK, below.

Create a Connection Using the C-language IDOL SDK

The following example shows how to create a connection object.

```
// Create the connection object
t_aciObject* pConnection = NULL;
aciObjectCreate(&pConnection, ACI_CONNECTION);

// Set host details
aciObjectParamSetString(pConnection, ACI_HOSTNAME, "12.3.4.56");
aciObjectParamSetInt(pConnection, ACI_PORTNUMBER, 4001);
aciObjectParamSetInt(pConnection, ACI_CONN_RETRIES, 1);
```

Perform an Action

A command object (type `ACI_COMMAND`) stores information about the action that you want to run, as well as all the parameters that this action requires, such as:

- the type of HTTP request (GET/POST) that is made when the action runs.
- security details for a data repository.

There is no limit to the number of parameters that can be stored, but there can only be one action.

Create an Action

The following example shows how to create a command object.

```
// Create the command object
t_aciObject* pCommand = NULL;
aciObjectCreate(&pCommand, ACI_COMMAND);

// Set command to execute
aciObjectParamSetString(pCommand, ACI_COM_COMMAND, "QUERY");

// Set http method
aciObjectParamSetBool(pCommand, ACI_COM_USE_POST,TRUE);

// Set action parameters
aciObjectParamSetString(
  pCommand, "Text", "Reducing heart attack risk");
aciObjectParamSetString(pCommand, "LanguageType", "English");
aciObjectParamSetBool(pCommand, "TotalResults", TRUE);

// Create security string
aciObjectSetUserSecurityInfo(pCommand, "notes", ACI_SECURITY_USERNAME, "jbloggs");
aciObjectSetUserSecurityInfo(
  pCommand, "notes", ACI_SECURITY_GROUP, "techusers");
```

Process a Response

There are two ways to execute the request, depending on how you want to process the response.

Obtain and Process an Unparsed Response

The `aciObjectExecuteToString` function returns the server response as an unparsed data buffer:

```
char *szResponse = NULL;
int nResponseLen = 0;
char *szContentType = NULL;

aciObjectExecuteToString(pConnection, pCommand, &szResponse, &nResponseLen,
&szContentType);

// For most actions, szResponse now holds the raw xml and szContentType will be
"application/xml"
// Process the xml in whatever way you wish now (e.g. third-party XML processing
API)

// Remember to tidy up afterwards
free(szResponse);
free(szContentType);
```

Obtain and Process a Parsed Response

The `aciObjectExecute` function returns a `t_aciObject` that holds a parsed representation of the XML, which you can manipulate by using the `acio` set of functions in the C IDOL SDK. For example if the XML returned was from the `UserReadAgentList` action and the following response was retrieved:

```
<?xml version="1.0" encoding="UTF-8" ?>
<autnresponse>
  <action>USERREADAGENTLIST</action>
  <response>SUCCESS</response>
  <responsedata>
  <maxagents>5</maxagents>
  <numagents>2</numagents>
  <agent>Housing</agent>
  <agent>Rock climbing</agent>
  </responsedata>
</autnresponse>
```

Then you can obtain the list of agent names for the user as follows:

```
t_aciObject *pResult = NULL;
char **szAgentNames = NULL;
aciObjectExecute(pConnection, pCommand, &pResult);

if (aciObjectCheckforSuccess(pResult))
{
  // Successful request so find out how many agents this user has
  t_aciObject* pNumAgent = NULL;
  pNumAgent = acioFindFirstOccurrence(pResult, "numagents");
  if(pNumAgent)
```

```
  {
    int nNumAgents = 0;
    acioParamGetInt(pNumAgent, ACI_DATA_NODE_VALUE, &nNumAgents);
    if(nNumAgents > 0)
    {
      // This user has agents, construct array to hold names and
      // retrieve agent names from response
      t_aciObject* pAgentName = NULL;
      int nAgent = 0;

      saAgentNames = (char**)malloc(sizeof(char*)*nNumAgents);

      // Find first agent entry
      pAgentName = acioFindFirstOccurrence(pResult, "agent");
      while(pAgentName)
      {
        acioParamGetString(pAgentName, ACI_DATA_NODE_VALUE,
        &saAgentNames[nAgent]);
        // Move to next agent entry
        pAgentName = aciObjectNextEntry(pAgentName);
        nAgent++
      }
    }
  }
}

// Remember to tidy up afterwards
aciObjectDestroy(&pResult);
```

> **NOTE:**
> Although it is imperative to free the memory that is associated with an ACI structure in C, you must not free the structure returned by the `acioFindFirstOccurrence` function.

Chapter 5: C-Language Reference

This section describes the C-language functions that are part of the C IDOL SDK.

Function Summary

The C-language functions can be divided into several categories as identified in the table below.

C-language functions

Function	Description
Initialization and housekeeping	
`aciInit`	Initializes the ACI API.
`aciInitEncryption`	Initializes the encryption keys to use for subsequent ACI connections.
`aciShutdown`	Cleans up at the end of an ACI client session.
`aciObjectCreate`	Creates an ACI object.
`aciObjectDestroy`	Destroys an ACI object.
`aciObjectIsAlive`	Determines whether the object is alive.
`aciObjectDeleteObject`	Deletes a single ACI object.
`aciObjectGetVersion`	Returns the version string of ACI client.
`aciClientFreeMemory`	Frees memory allocated with ACI client.
`acioGetErrorDescription`	Returns an error description.
Execution	
`aciObjectExecute`	Executes an action.
`aciObjectExecuteToString`	Executes an action to a string buffer.
`aciObjectCheckForSuccess`	Determines whether the execution was successful.
Access	
`aciObjectParamSetString`	Sets a string parameter in an object.
`aciOjbectParamSetStringN`	Sets a string parameter with a length.
`aciOjbectParamSetStringB`	Sets a binary data parameter with a length.
`acioParamGetString`	Gets a string parameter in an object.
`acioParamGetStringMoveOriginal`	Gets a string parameter in, and removes it from, an object.
`aciObjectParamSetInt`	Sets an integer parameter in an object.
`acioParamGetInt`	Gets an integer parameter in an object.

C-language functions, continued

Function	Description
aciObjectParamSetUnsignedInt	Sets an unsigned integer parameter in an object.
acioParamGetUnsignedInt	Gets an unsigned integer parameter in an object.
aciObjectParamSetLong	Sets a long parameter in an object.
acioParamGetLong	Gets a long parameter in an object.
aciObjectParamSetBool	Sets a boolean parameter in an object.
acioParamGetBool	Gets a boolean parameter in an object.
aciObjectParamSetDouble	Sets a double parameter in an object.
acioParamGetDouble	Gets a double parameter in an object.
aciErrorStatusCreate	Creates an error status structure.
aciErrorStatusDestroy	Destroys an error status structure.
aciErrorStatusReset	Resets an error status structure.
aciResponseGetErrorStatus	Extracts error status information.
acioGetFirstVariable	Accesses the parameter list of an ACI object.
acioGetNextVariable	Accesses the next variable (parameter) in the list after the one passed in.
acioGetVariableName	Accesses the name of an ACI object variable.
acioGetVariableValue	Accesses the value of an ACI object variable.
Search and iteration	
aciObjectFirstEntry	Locates the first child entry of a result.
aciObjectNextEntry	Locates the next child entry of a result.
acioNextNamedEntry	Locates the next child entry of a result that matches the name.
aciObjectParentEntry	Returns a pointer to the parent entry of the object.
acioFindFirstOccurrence	Locates the first occurrence of a tag.
acioFindFirstOccurrenceFromRoot	Locates the first occurrence of a tag.
acioFindFirstEnclosedOccurrenceFromRoot	Locates the first occurrence of a tag from the object's immediate children.
aciObjectToString	Prints an ACI object.

C-language functions, continued

Function	Description
`acioGetTagValue`	Obtains the value of a tag.
`aciObjectGetTagValueWithDefault`	Obtains the value of a tag.
`acioGetValue`	Obtains the content from an ACI object.
`acioGetName`	Obtains the name of an ACI object.
`aciObjectToXMLString`	Converts an ACI object to XML format.
`aciObjectToXMLStringNoFormat`	Converts an ACI object to XML without any formatting.
`aciObjectAddAttribute`	Adds an `aciObjectVariable` to the object's attribute linked list.
`aciObjectAttributeGetNames`	Populates an array of strings with the names of all the attributes of the XML tag represented by the object.
`aciObjectAttributeGetNamesAndValuesNoCopy`	Retrieves an array of attribute names and values of an ACI object.
`acioAttributeGetValue`	Gets the value of an attribute.
`aciObjectResponseToStringArray`	Returns all values of a tag in an ACI response object.
Security	
`aciObjectSetSecurityKeys`	Sets the security keys.
`aciObjectSetUserSecurityInfo`	Sets the user-level security for documents.

Constants

The constants that are used in the C language SDK are outlined in the following table.

Constants used in C-language functions

Constant	Description
ACI_COMMAND	Indicates that you want to create an ACI_COMMAND type ACI object. This constant is used by only the aciObjectCreate function.
ACI_COM_COMMAND	Indicates that the specified parameter is an action command. This constant is used by only the aciObjectParamSetString function on an ACI_COMMAND type ACI object.
ACI_COM_USE_POST	Indicates whether to use GET or POST when sending data to the server. This constant is used by only the aciObjectParamSetBool function on an ACI_COMMAND type ACI object.
ACI_CONNECTION	Indicates that you want to create an ACI_CONNECTION type ACI object. This constant is used by only the aciObjectCreate function.
ACI_CONN_ENCRYPTION	Indicates whether to encrypt communications with the ACI server. This constant is used by only the aciObjectParamSetBool function on an ACI_CONNECTION type ACI object. (You must also specify an encryption type if you want to encrypt communications.)
ACI_CONN_ENCRYPTION_TYPE	Indicates that the specified parameter is the encryption type to use for communications with the ACI server. This constant is used by only the aciObjectParamSetString function on an ACI_CONNECTION type ACI object.
ACI_CONN_ENCRYPTION_TYPE_GSS	Sets the GSS encryption type with which to encrypt communications with the ACI server. This constant is used by only the aciObjectParamSetString function that sets the ACI_CONN_ENCRYPTION_TYPE on an ACI_CONNECTION type ACI object.
ACI_CONN_ENCRYPTION_TYPE_TEA	Sets the TEA encryption type with which to encrypt communications with the ACI server. This constant is used by only the aciObjectParamSetString function that sets the ACI_CONN_ENCRYPTION_TYPE on an ACI_CONNECTION type ACI object.
ACI_CONNECTION_REALM_KEY	Sets the realm key of GSS encryption with which to encrypt communications with the ACI server. This constant is used by only the aciObjectParamSetString function on an ACI_CONNECTION type ACI object with ACI_CONN_ENCRYPTION_TYPE_GSS encryption type.
ACI_CONNECTION_SERVICE_KEY	Sets the service key of GSS encryption with which to encrypt communications with the ACI server. This constant is used by only the aciObjectParamSetString function on an ACI_CONNECTION type ACI object with ACI_CONN_ENCRYPTION_TYPE_GSS encryption type.

Constants used in C-language functions, continued

Constant	Description
ACI_CONN_HTTPHEADER_XFORWARDEDFOR	Sets the X-Forwarded-For HTTP header.
ACI_CONN_RETRIES	Indicates that the specified parameter is the number of times a connection is retried. This constant is used by only the aciObjectParamSetInt function on an ACI_CONNECTION type ACI object.
ACI_CONN_TCP_RCVBUF	Sets the size of the socket receive buffer SO_RCVBUF.
ACI_CONN_USE_DEFLATE	Indicates that the specified parameter specifies whether to use deflate compression in HTTP requests. This constant is used by only the aciObjectParamSetBool function on an ACI_CONNECTION type ACI object.
ACI_CONN_TIMEOUT	Indicates that the specified parameter is the timeout used in the connection. This constant is used by only the aciObjectParamSetInt function on an ACI_CONNECTION type ACI object.
ACI_CONN_TIMEOUT_READ	Indicates that the specified parameter is the timeout for how long the connection waits to read data. If you do not set this constant, the connection uses the ACI_CONN_TIMEOUT value as the timeout for both the initial connect or send request phase and the retrieving data phase of the ACI request. This constant is used by only the aciObjectParamSetInt function on an ACI_CONNECTION type ACI object.
ACI_CONN_TIMEOUT_READ_ERROR	Indicates that the specified parameter specifies whether to use a separate error for connection timeouts at the read stage. You can use this parameter to return separate ACICONTENT_ERRORTIMEOUTREAD and ACICONTENT_ERRORTIMEOUT return codes. This constant is used by only the aciObjectParamSetBool function on an ACI_CONNECTION type ACI object.
ACI_CONN_USE_SSL	Indicates that the specified parameter specifies whether to use HTTPS requests. This constant is used by only the aciObjectParamSetBool function on an ACI_CONNECTION type ACI object.
ACI_CONN_SSL_CERTIFICATE	Indicates that the specified parameter gives the location on disk of the SSL certificate to be used for HTTPS requests. This constant is used by only the aciObjectParamSetString function on an ACI_CONNECTION type ACI object.
ACI_CONN_SSL_PRIVATEKEY	Indicates that the specified parameter gives the location on disk of the SSL private key file matching the SSL certificate to be used for HTTPS requests. This constant is used by only the aciObjectParamSetString function on an ACI_CONNECTION type ACI object.
ACI_CONN_SSL_CHECKCOMMONNAME	Indicates that the specified parameter specifies whether to check that the CommonName (CN) attribute of the SSL certificate returned by the peer resolves to the same IP address of the hostname. This constant is used by only the aciObjectParamSetBool function on an ACI_CONNECTION type ACI object.

Constants used in C-language functions, continued

Constant	Description
ACI_CONN_SSL_CACERTIFICATE	Indicates that the specified parameter gives the location on disk of the SSL Certificate Authority (CA) certificate of a trusted authority to be used for HTTPS requests. Communication with a peer is only trusted if it provides a certificate signed by the specified CAs. This constant is used by only the aciObjectParamSetString function on an ACI_CONNECTION type ACI object.
ACI_DATA	Indicates that you want to create an ACI_DATA type ACI object. This constant is used by only the aciObjectCreate function.
ACI_DATA_DATA	Indicates that the specified parameter is all data that the server has returned. This constant is used by only the aciObjectParamGetString function on an ACI_DATA type ACI object.
ACI_DATA_NODE_NAME	Indicates that the specified parameter is the node name. This constant is used by only the aciObjectParamGetString function on an ACI_DATA type ACI object.
ACI_DATA_NODE_VALUE	Indicates that the specified parameter is the node value. This constant is used by only the aciObjectParamGetString function on an ACI_DATA type ACI object.
ACI_HOSTNAME	Indicates that the specified parameter is the IP address (or name) of the ACI server. This constant is used by only the aciObjectParamSetString function on an ACI_CONNECTION type ACI object.
ACI_PORTNUMBER	Indicates that the specified parameter is the port number that is used to communicate with the ACI server. This constant is only used by the aciObjectParamSetInt function on an ACI_CONNECTION type ACI object.
ACI_SECURITY_DOMAIN	Indicates that the specified value is a domain name. This constant is used by only the aciObjectSetUserSecurityInfo function on an ACI_COMMAND type ACI object.
ACI_SECURITY_EXTENDED	Indicates that the specified value is the value for an additional field used for security in this repository. This constant is used by only the aciObjectSetUserSecurityInfo function on an ACI_COMMAND type ACI object.
ACI_SECURITY_GROUP	Indicates that the specified value is a user group name. This constant is used by only the aciObjectSetUserSecurityInfo function on an ACI_COMMAND type ACI object.
ACI_SECURITY_PASSWORD	Indicates that the specified value is a password. This constant is used by only the aciObjectSetUserSecurityInfo function on an ACI_COMMAND type ACI object.
ACI_SECURITY_USERNAME	Indicates that the specified value is a user name. This constant is used by only the aciObjectSetUserSecurityInfo function on an ACI_COMMAND type ACI object.

aciClientFreeMemory

Call the `aciClientFreeMemory` function to free memory allocated with the ACI client.

Syntax

```
aciError aciClientFreeMemory(
   void** ppMem)
```

Arguments

Arguments	Type/Description
`ppMem`	**`void**`** A pointer to the memory to free.

Discussion

This function provides a mechanism to free arbitrary memory allocated from within the ACI client API.

Returns

An ACI error code.

aciErrorStatusCreate

Call the `aciErrorStatusCreate` function to create an error status structure for `aciResponseGetErrorStatus` to use.

Syntax

```
aciError aciErrorStatusCreate(
   t_aciErrorStatus** ppErrorStatus)
```

Arguments

Arguments	Type/Description
`ppErrorStatus`	**`t_AciErrorStatus**`** The ACI error status structure to create.

Discussion

This function creates the ACI error status structure. You must call this function before using `aciResponseGetErrorStatus`.

Returns

An ACI error code.

aciErrorStatusDestroy

Call the `aciErrorStatusDestroy` function to destroy an error status structure created by `aciErrorStatusCreate`.

Syntax

```
aciError aciErrorStatusDestroy(
   t_aciErrorStatus **ppErrorStatus)
```

Arguments

Arguments	Type/Description
`ppErrorStatus`	**`t_AciErrorStatus**`** The ACI error status structure to destroy.

Discussion

This function destroys the ACI error status structure. You must call this function after using `aciResponseGetErrorStatus`.

Returns

An ACI error code.

aciErrorStatusReset

Call the `aciErrorStatusCreate` function to reset an error status structure that the `aciResponseGetErrorStatus` uses.

Syntax

```
aciError aciErrorStatusReset(
   t_aciErrorStatus *pErrorStatus)
```

Arguments

Arguments	Type/Description
`pErrorStatus`	**`t_AciErrorStatus*`** The ACI error status structure to reset.

Discussion

This function resets the ACI error status structure.

Returns

An ACI error code.

aciInit

Call the `aciInit` function to initialize the ACI API.

Syntax

```
void aciInit ( )
```

Discussion

You must call this function at the beginning of an application.

aciInitEncryption

Call the `aciInitEncryption` function to initialize the encryption keys to use for subsequent ACI connections.

Syntax

```
aciError aciInitEncryption (
    BOOL bOEMLicensed,
    char* szEncType,
    char* szTEAKeys)
```

Arguments

Arguments	Type/Description
`bOEMLicensed`	**`BOOL`** One of the following values: • `TRUE` • `FALSE`
`szEncType`	**`char*`** The encryption type. Set this to `TEA` to use OEM encryption.
`szTEAKeys`	**`char*`** The TEA keys to use for OEM encryption.

Discussion

You must call this function at the beginning of an application when you want to use OEM encryption. If the application is OEM licensed, the API attempts to encrypt all requests.

For more information about using OEM encryption, see OEM Encryption, on page 111.

acioAttributeGetValue

Call the `acioAttributeGetValue` function to get the value of an attribute.

Syntax

```
aciError acioAttributeGetValue(
   t_aciObject* pObject,
   const char* szAttribName,
   char** pszAttribValue)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to the ACI object whose attribute value you want to obtain.
`szAttributeName`	**`char*`** The attribute name.
`pszNumAttribValue`	**`char**`** A pointer to a string to store the attribute value.

Discussion

This function gets the value of the specified attribute (returned in `pszAttributeValue`). It searches through the linked list `pAttributeVars` to find the entry which matches `szAttribName`.

Returns

An ACI error code.

aciObjectAddAttribute

Call the `aciObjectAddAttribute` function to add a new attribute to an ACI object.

Syntax

```
aciError aciObjectAddAttribute(
   t_aciObject* pObject,
   const char* szAttributeName,
   const char* szAttributeValue)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to the ACI object to add the attribute to.
`szAttributeName`	**`const char*`** The attribute name.
`szAttributeValue`	**`const char*`** The attribute value.

Discussion

This function adds an `aciObjectVariable` to the object's attribute linked list.

Returns

An ACI error code.

aciObjectAttributeGetNames

Call the `aciObjectAttributeGetNames` function to get an array of the attributes of the current object.

Syntax

```
aciError aciObjectAttributeGetNames(
   t_aciObject* pObject,
   char*** paszAttribNames,
   int* pnNumAttributes)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to the ACI object whose attribute names you want to obtain.
`paszAttributeNames`	**`char***`** A pointer to an array of strings to store the attribute names.
`pnNumAttributes`	**`int*`** A pointer to an integer to store the number of attributes.

Discussion

This function populates an array of strings with the names of all the attributes of the XML tag represented by the object.

Returns

An ACI error code.

aciObjectAttributeGetNamesAndValuesNoCopy

Call the `aciObjectAttributeGetNamesAndValuesNoCopy` function to retrieve arrays of attribute names and values of an ACI object.

Syntax

```
aciError aciObjectAttributeGetNamesAndValuesNoCopy(
   t_aciObject* pObject,
   const char*** paszAttrNames,
   const char*** paszAttrValues,
   int* pnNumAttrs);
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an ACI object.
`paszAttrNames`	**`char***`** A pointer to an array of strings containing the attribute names.
`paszAttrValues`	**`char***`** A pointer to an array of strings containing the attribute values.
`pnNumAttrs`	**`int*`** A pointer to the number of attributes.

Discussion

This function populates the arrays with names and values of the attributes of the object `pObject`. The two arrays are allocated but the actual attribute names and values are not.

Returns

An ACI error code.

aciObjectCheckForSuccess

Call the `aciObjectCheckForSuccess` function to check whether an ACI server reply object contains a successful response entry.

Syntax

```
BOOL aciObjectCheckForSuccess (
   t_aciObject* pACIOResult)
```

Arguments

Arguments	Type/Description
`pACIOResult`	**`t_aciObject*`** A pointer to an ACI server reply object you want to check.

Discussion

You can only call this function for an `ACI_DATA` type ACI object.

Returns

`TRUE` if this object is an `ACI_DATA` type ACI object and holds a response object with the value `SUCCESS`; otherwise it returns `FALSE`.

aciObjectCreate

Call the `aciObjectCreate` function to create an ACI object.

Syntax

```
aciError aciObjectCreate (
   t_aciObject** ppObject,
   aciObjectType nType);
```

Arguments

Arguments	Type/Description
`ppObject`	**`t_aciObject**`** On return, the ACI object you created.
`nType`	**`aciObjectType`** One of the following data type constants: • `ACI_CONNECTION`. A connection object. • `ACI_COMMAND`. A command object. • `ACI_DATA`. A data object.

Discussion

This function creates and allocates memory for an ACI object of type `nType`. Typically, you do not create `ACI_DATA` type objects; call the `aciObjectExecute` function instead. For information about the `aciObjectExecute` function, see aciObjectExecute, on page 61.

Examples

```
aciObjectCreate(&pCommand, ACI_COMMAND);
aciObjectCreate(&pConnection, ACI_CONNECTION);
aciObjectCreate(&pResponse, ACI_DATA);
```

aciObjectDeleteObject

Call the `aciObjectDeleteObject` function to delete a single ACI object.

Syntax

```
aciError aciObjectDeleteObject(
   t_aciObject* pObject,
   t_aciObject* pPrevious)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** The ACI object to delete.
`pPrevious`	**`t_aciObject*`** ACI object (can be null if `pObject` is first) in the linked list just before `pObject`.

Discussion

This function deletes a ACI object, rearranging the pointers from its parent or sibling, and freeing all its memory.

Returns

An ACI error code.

aciObjectDestroy

Call the `aciObjectDestroy` function to destroy an ACI object.

Syntax

```
aciError aciObjectDestroy (
   t_aciObject** ppObject);
```

Arguments

Arguments	Type/Description
`ppObject`	**`t_aciObject**`** The ACI object to destroy.

Discussion

This function frees memory from the ACI object and all objects in its linked lists. This is a deep destroy.

Returns

An ACI error code.

aciObjectExecute

Call the `aciObjectExecute` function to run an ACI action.

Syntax

```
aciError aciObjectExecute (
   t_aciObject* pConnection,
   t_aciObject* pToExecute,
   t_aciObject** ppResult);
```

Arguments

Arguments	Type/Description
`pConnection`	**`t_aciObject*`** A pointer to an `ACI_CONNECTION` object.
`pToExecute`	**`t_aciObject*`** A pointer to an ACI object that contains the action.
`ppResult`	**`t_aciObject**`** An ACI object created by this function to hold the result.

Discussion

This function opens the `pConnection` connection, run the action that is contained in the `pToExecute` ACI object, and then closes the connection. You can call this function only for a `Connection` type ACI object.

The output `ppResult` ACI object either contains the response in plain text or contains the top node of a tree structure, which represents the XML response.

Returns

An ACI error code.

aciObjectExecuteToString

Call the `aciObjectExecuteToString` function to run an ACI action to a string buffer, rather than to an ACI response object.

Syntax

```
aciError aciObjectExecuteToString(
   t_aciObject* pConnection,
   t_aciObject* pToExecute,
   char** pszResult,
   int* pnBufferLength,
   char** pszContentType);
```

Arguments

Arguments	Type/Description
`pConnection`	**`t_aciObject*`** A pointer to an `ACI_CONNECTION` object.
`pToExecute`	**`t_aciObject*`** A pointer to an ACI object that contains the action.
`pszResult`	**`char**`** A pointer to the response string buffer.
`pnBufferLength`	**`int*`** A pointer to the length of the response string buffer.
`pszContentType`	**`char**`** A pointer to the string containing the value of the *Content-type* HTTP header field in the response.

Discussion

This function opens the `pConnection` connection, runs the action that is contained in the `pToExecute` ACI object, and then closes the connection. You can call this function only for a `Connection` type ACI object.

The response string buffer contains the body of the response.

Returns

An ACI error code.

aciObjectFirstEntry

Call the `aciObjectFirstEntry` function to locate the first child entry in a result.

Syntax

```
t_aciObject* aciObjectFirstEntry (
   t_aciObject* pObject)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that contains an XML result.

Discussion

You can only call this function for an `ACI_DATA` type ACI object.

Returns

The first child entry or `NULL` if no child entry exists.

aciObjectGetTagValueWithDefault

Call the `aciObjectGetTagValue` function to obtain the value associated with the first occurrence of a tag.

Syntax

```
aciError aciObjectGetTagValueWithDefault (
   t_aciObject* pObject,
   char* szTagName,
   char** pszTagValue,
   char* szDefault)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` type object.
`szTag`	**`char*`** A pointer to the requested tag.
`pszTagValue`	**`char**`** The contents associated with the tag.
`szDefault`	**`char*`** A pointer to the default string.

Discussion

This function finds the first occurrence of the specified `szTagName`, extracts its character value and sets the pointer `pszTagValue` to the memory space (allocated by this function) where it stores the character value. If there is no tag with the specified name or if that tag does not contain character data, `pszTagValue` points to the `szDefault` string.

Returns

An ACI error code.

aciObjectGetVersion

Call the `aciObjectGetVersion` function to return the version string of the ACI client.

Syntax

```
char* aciObjectGetVersion()
```

Discussion

This function gets a human-readable string describing the version of the ACI Client API. Memory is allocated to the returned pointer and must be freed by the caller.

Returns

A string containing the version number.

aciObjectIsAlive

Call the `aciObjectIsAlive` function to determine whether the ACI server is accessible.

Syntax

```
BOOL aciObjectIsAlive (
   t_aciObject* pACIOConnection)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_CONNECTION` type object.

Discussion

This function checks whether the ACI server is available and accessible. You can only call this function for an `ACI_CONNECTION` type ACI object.

Returns

`TRUE` if the ACI server is running; otherwise, it returns `FALSE`.

aciObjectNextEntry

Call the `aciObjectNextEntry` function to locate the next child entry in a result.

Syntax

```
t_aciObject* aciObjectNextEntry (
   t_aciObject* pObject)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that contains an XML result.

Discussion

You can only call this function for an `ACI_DATA` type ACI object.

Returns

The next entry or `NULL` if the object is not found or no next entry exists.

aciObjectParamSetBool

Call the `aciObjectParamSetBool` function to set the contents of a Boolean parameter.

Syntax

```
aciError aciObjectParamSetBool (
   t_aciObject* pObject,
   char* szName,
   BOOL bValue)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that contains an XML result.
`szName`	**`char*`** A pointer to the parameter name.
`bValue`	**`BOOL`** One of the following values: • `TRUE` • `FALSE`

Discussion

This function adds the `szName` Boolean parameter with the specified `bValue` to the `pObject`.

Returns

An ACI error code.

Examples

```
aciObjectParamSetBool(pCommand,ACI_COM_USE_POST,TRUE);
aciObjectParamSetBool(pConnection, ACI_CONN_ENCRYPTION, TRUE);
```

aciObjectParamSetDouble

Call the `aciObjectParamSetDouble` function to set the contents of a double-precision floating point parameter.

Syntax

```
aciError aciObjectParamSetDouble (
   t_aciObject* pObject,
   char* szName,
   double dValue,
   int nDecimalPlaces)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that contains an XML result.
`szName`	**`char*`** A pointer to the parameter name.
`dValue`	**`double`** A double-precision floating point value.
`nDecimalPlaces`	**`int`** The number of digits to preserve to the right of the decimal point.

Discussion

This function adds the `szName` double parameter with the specified `dValue` to the `pObject`.

Returns

An ACI error code.

aciObjectParamSetInt

Call the `aciObjectParamSetInt` function to set the contents of an integer parameter.

Syntax

```
aciError aciObjectParamSetInt(
   t_aciObject* pObject,
   char* szName,
   int nValue)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that contains an XML result.
`szName`	**`char*`** A pointer to the parameter name.
`nValue`	**`int`** An integer value.

Discussion

This function adds the `szName` integer parameter with the specified `nValue` to the `pObject`.

Returns

An ACI error code.

Examples

```
aciObjectParamSetInt(pConnection, ACI_PORTNUMBER, 4001);
```

aciObjectParamSetLong

Call the `aciObjectParamSetLong` function to set the contents of a long integer parameter.

Syntax

```
aciError aciObjectParamSetLong (
   t_aciObject* pObject,
   char* szName,
   long lnValue);
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that contains an XML result.
`szName`	**`char*`** A pointer to the parameter name.
`lnValue`	**`long`** A long integer value.

Discussion

This function adds the `szName` long parameter with the specified `lnValue` to the `pObject`.

Returns

An ACI error code.

aciObjectParamSetString

Call the `aciObjectParamSetString` function to set the contents of a string parameter.

Syntax

```
aciError aciObjectParamSetString (
   t_aciObject* pObject,
   char* szName,
   char* szValue);
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that contains an XML result.
`szName`	**`char*`** A pointer to the parameter name.
`szValue`	**`char*`** A string value.

Discussion

This function adds the `szName` string parameter with the specified `szValue` to the `pObject`.

Returns

An ACI error code.

Example

```
aciObjectParamSetString(pCommand, ACI_COM_COMMAND, "Query"

aciObjectParamSetString(pConnection, ACI_CONN_ENCRYPTION,
   ACI_CONN_ENCRYPTION_TYPE_TEA);

aciObjectParamSetString(pConnection, ACI_HOSTNAME, "12.3.4.56");
```

aciObjectParamSetStringB

Call the `aciObjectParamSetStringB` function to set the contents of a binary data parameter with a length.

Syntax

```
aciError aciObjectParamSetStringB (
   t_aciObject* pObject,
   size_t nLength,
   const char *szName,
   const char *szValue);
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to the ACI object.
`nLength`	**`size_t`** The number of bytes to copy from `szValue`.
`szName`	**`char*`** The parameter name.
`szValue`	**`char*`** The parameter value.

Discussion

This function adds the binary data parameter `szName` with `nLength` bytes copied from value `szValue` to the ACI object `pObject`.

Returns

An ACI error code.

aciObjectParamSetStringN

Call the `aciObjectParamSetStringN` function to set the contents of a string parameter with a length.

Syntax

```
aciError aciObjectParamSetStringN (
   t_aciObject* pObject,
   int nLength
   const char *szName,
   const char *szValue);
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to the ACI object.
`nLength`	**`int`** The length of the parameter value to set to.
`szName`	**`char*`** The parameter name.
`szValue`	**`char*`** The parameter value.

Discussion

This function adds the string parameter `szName` with the specific value `szValue` to the ACI object `pObject`. If the length of `szValue` is greater than `nLength`, only `nLength` bytes of it is set.

Returns

An ACI error code.

aciObjectParamSetUnsignedInt

Call the `aciObjectParamSetUnsignedInt` function to set the contents of an unsigned integer parameter.

Syntax

```
aciError aciObjectParamSetUnsignedInt(
   t_aciObject* pObject,
   const char* szName,
   unsigned int nValue)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that contains an XML result.
`szName`	**`char*`** A pointer to the parameter name.
`nValue`	**`unsigned int`** An unsigned integer value.

Discussion

This function adds the `szName` unsigned integer parameter with the specified `nValue` to the `pObject`.

Returns

An ACI error code.

Examples

```
aciObjectParamSetUnsignedInt(pConnection, ACI_PORTNUMBER, 4001);
```

aciObjectParentEntry

Call the `aciObjectParentEntry` function to return a pointer to the parent entry of the object.

Syntax

```
t_aciObject* aciObjectParentEntry(
   t_aciObject* pObject)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that contains an XML result.

Discussion

This function accesses the parent of the current object.

Returns

The parent ACI object.

aciObjectResponseToStringArray

Call the `aciObjectResponseToStringArray` function to return all the values of a tag in an ACI response object.

Syntax

```
aciError aciObjectResponseToStringArray(
   t_aciObject *pResponse,
   char *szTagName,
   char *szNumberName,
   char ***paszValues,
   long *plnValues );
```

Arguments

Arguments	Type/Description
`pResponse`	**`t_aciObject*`** A pointer to an ACI response object.
`szTagName`	**`char*`** The name of the tag values to retrieve.
`szNumberName`	**`char*`** The name of the tag, where available, giving the number of tag values to retrieve.
`paszValues`	**`char***`** A pointer to an array of strings containing the tag values.
`plnValues`	**`long*`** A pointer to the number of values retreived.

Discussion

This function populates the array pointed to by `paszValues` with the values of the tags named `szTagName`. Memory is allocated for the array as well as the values.

Returns

An ACI error code.

aciObjectSetSecurityKeys

Call the `aciObjectSetSecurityKeys` function to specify the security keys for communication with an ACI server.

Syntax

```
aciObjectSetSecurityKeys (
   t_aciObject* pObject,
   int nKey1,
   int nKey2,
   int nKey3,
   int nKey4)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_CONNECTION` type object.
`nKey1`	**`int`** Value of the first security key.
`nKey2`	**`int`** Value of the second security key.
`nKey3`	**`int`** Value of the third security key.
`nKey4`	**`int`** Value of the fourth security key.

Discussion

This function sets the four security keys needed for secure communication with an ACI server. Enter the value of the first security key with `nKey1`, the value of the second security key with `nKey2`, and so on. You can call this function only for an `ACI_CONNECTION` type ACI object.

Returns

An ACI error code.

aciObjectSetUserSecurityInfo

Call the `aciObjectSetUserSecurityInfo` function to set the user-level security for documents in a repository.

Syntax

```
aciObjectSetUserSecurityInfo (
   t_aciObject* pObject,
   char* szRepository,
   char* szName,
   char* szValue)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_CONNECTION` type object.
`szRepository`	**`char*`** A pointer to a repository.
`szName`	**`char*`** A pointer to a security field name, which is one of the following values: • `ACI_SECURITY_USERNAME`. User name. • `ACI_SECURITY_PASSWORD`. Password. • `ACI_SECURITY_GROUP`. Group. • `ACI_SECURITY_DOMAIN`. Domain. • `ACI_SECURITY_EXTENDED`. Other information.
`szValue`	**`char*`** The value to associate with the security field.

Discussion

This function sets security information for a user, which is used to determine access to documents in an IDOL server. Enter the repository type with `szRepository` (this must match the name of the IDOL server configuration section in which you make the security settings for this type of repository). To enter the security information, set the security field with `szName` and the value for the field with `szValue`.

For information about the `szValue` arguments to set for various repositories, see Security Information Values, on page 107.

The information that you set with this function is encrypted and automatically added as the security string that is sent with an action to the ACI server. You do not need to set this parameter separately with a call to the `aciObjectParamSetString` function.

You can only call this function for an `ACI_COMMAND` type ACI object.

Returns

An ACI error code.

Examples

```
aciObjectSetUserSecurityInfo(pCommand, "netware",
   ACI_SECURITY_EXTENDED, "x");

aciObjectSetUserSecurityInfo(pCommand, "nt",
   ACI_SECURITY_DOMAIN, "Office");

aciObjectSetUserSecurityInfo(pCommand, "nt",
   ACI_SECURITY_GROUP, "techusers");

aciObjectSetUserSecurityInfo(pCommand, "notes",
   ACI_SECURITY_PASSWORD, "pass1234");

aciObjectSetUserSecurityInfo(pCommand, "notes",
   ACI_SECURITY_USERNAME, "jsmith");
```

aciObjectToString

Call the `aciObjectToString` function to print an ACI object.

Syntax

```
void aciObjectToString (
   t_aciObject* pObject)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an ACI object.

Discussion

This function prints a string representation of the ACI object to the standard output.

aciObjectToXMLString

Call the `aciObjectToXMLString` function to convert an object into an XML-formatted string.

Syntax

```
char* aciObjectToXMLString (
   t_aciObject* pObject)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an ACI object.

Returns

The ACI object as a string containing XML formatting.

aciObjectToXMLStringNoFormat

Call the `aciObjectToXMLStringNoFormat` function to convert an object into an XML string without newlines or tabs formatting.

Syntax

```
char* aciObjectToXMLStringNoFormat(
   t_aciObject* pObject)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an ACI object.

Discussion

This function converts the XML beneath `pObject` and return it as XML without formatting the output with tabs or newlines.

Returns

A string.

acioFindFirstEnclosedOccurrenceFromRoot

Call the `acioFindFirstEnclosedOccurrenceFromRoot` function to locate the first occurrence of the specified tag in an XML result from the immediate children.

Syntax

```
t_aciObject* acioFindFirstEnclosedOccurrenceFromRoot(
   t_aciObject* pObject,
   const char* szTag)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that contains an XML result.
`szTag`	**`const char*`** A pointer to the requested tag.

Discussion

This function finds the first occurrence of the specified `szTag` in the XML structure represented by `pObject`, searching only in `pObject` itself and its immediate child objects. It does not search `pObject`'s siblings, nor its descendents beyond its immediate children.

You can only call this function for an `ACI_DATA` type ACI object.

Returns

A pointer to the tag's contents or `NULL` if the tag is not found.

acioFindFirstOccurrence

Call the `acioFindFirstOccurrence` function to locate the first occurrence of the specified tag in an XML result.

Syntax

```
t_aciObject* acioFindFirstOccurrence (
   t_aciObject* pObject,
   char* szTag)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that is part of an XML result.
`szTag`	**`char*`** A pointer to the requested tag.

Discussion

The function finds the first occurrence of the specified `szTag` in the XML structure represented by `pObject`, using a *child first* search through the linked lists. You can call this function only for an `ACI_DATA` type object.

Returns

A pointer to the tag's contents or `NULL` if the tag is not found.

acioFindFirstOccurrenceFromRoot

Call the `acioFindFirstOccurrenceFromRoot` function to locate the first occurrence of the specified tag in an XML result.

Syntax

```
t_aciObject* acioFindFirstOccurrenceFromRoot (
   t_aciObject* pObject,
   char* szTag)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that is part of an XML result.
`szTag`	**`char*`** A pointer to the requested tag.

Discussion

This function finds the first occurrence of the specified `szTag` in the XML structure represented by `pObject`, searching only in `pObject` itself and its child object tree. If it does not find the specified `szTag`, it does not search `pObject` siblings.

You can call this function only for an `ACI_DATA` type ACI object.

Returns

A pointer to the tag's contents or `NULL` if the tag is not found.

acioGetErrorDescription

Call the `acioGetErrorDescription` function to return an error description.

Syntax

```
onst char* acioGetErrorDescription(
   aciError nError);
```

Arguments

Arguments	Type/Description
`nError`	**`aciError`** An ACI error code.

Discussion

This function returns an error description of the error code `nError`.

Returns

An error description of the error code `nError`.

acioGetFirstVariable

Call the `acioGetFirstVariable` function to access the parameter list of an ACI object.

Syntax

```
t_aciObjectVariable* acioGetFirstVariable(
   t_aciObject* pObject)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to the ACI object.

Discussion

This function accesses the parameter list of an ACI object. You cannot de-reference this pointer directly (the type is opaque), but you can can pass it to `acioGetVariableName()` or `acioGetNextVariable()`.

Returns

An ACI object variable.

acioGetName

Call the `acioGetName` function to obtain the name of an ACI object.

Syntax

```
char* acioGetName (
   t_aciObject* pObject)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to the ACI object whose name you want to obtain.

Returns

A pointer to the name of the ACI object.

acioGetNextVariable

Call the `acioGetNextVariable` function to access the next variable (parameter) in the list after the one passed in.

Syntax

```
t_aciObjectVariable* acioGetNextVariable(
   t_aciObjectVariable* pVariable)
```

Arguments

Arguments	Type/Description
`pVariable`	**`t_aciObjectVariable*`** A pointer to an ACI object variable.

Discussion

This function accesses the next variable (parameter) in the list after the one passed in.

Returns

An ACI object variable or `NULL` if this is the last parameter.

acioGetTagValue

Call the `acioGetTagValue` function to obtain the value associated with the first occurrence of a tag.

Syntax

```
aciError acioGetTagValue (
   t_aciObject* pObject,
   char* szTagName,
   char** pszTagValue)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` type object.
`szTag`	**`char*`** A pointer to the requested tag.
`pszTagValue`	**`char**`** The contents associated with the tag.

Discussion

This function finds the first occurrence of the specified `szTagName`, extracts its character value, and sets the pointer `pszTagValue` to the memory space (allocated by this function) where it stores the character value. If there is no tag with the specified name or if that tag does not contain character data, `pszTagValue` points to a `NULL` character.

Returns

An ACI error code.

acioGetValue

Call the `acioGetValue` function to obtain the content from an ACI object.

Syntax

```
char* acioGetValue (
   t_aciObject* pObject)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an ACI object.

Returns

Returns the content associated with the object.

acioGetVariableName

Call the `acioGetVariableName` function to access the name of an ACI object variable.

Syntax

```
const char* acioGetVariableName(
   t_aciObjectVariable* pVariable)
```

Arguments

Arguments	Type/Description
`pVariable`	**`t_aciObjectVariable*`** A pointer to an ACI object variable.

Discussion

This function accesses the name of an ACI Object variable. The data is not duplicated, and must not be freed by the caller.

Returns

A string containing the variable name.

acioGetVariableValue

Call the `acioGetVariableValue` function to access the value of an ACI object variable.

Syntax

```
const char* acioGetVariableValue(
   t_aciObjectVariable* pVariable)
```

Arguments

Arguments	Type/Description
`pVariable`	**`t_aciObjectVariable*`** A pointer to an ACI object variable.

Discussion

This function accesses the value of an ACI Object variable. The data is not duplicated, and must not be freed by the caller.

Returns

A string containing the variable value.

acioNextNamedEntry

Call the `acioNextNamedEntry` function to locate the next child entry in a result that matches the name.

Syntax

```
t_aciObject* acioNextNamedEntry(
   t_aciObject* pObject,
   const char* szName)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that contains an XML result.
`szName`	**`const char*`** The name of the tag to match.

Discussion

Find the next entry in the same manner as `acioNextEntry`, but skip tags that do not match `szName`.

Returns

The next entry that matches or `NULL` if the object is not found or no next entry exists.

acioParamGetBool

Call the `acioParamGetBool` function to obtain the contents of a Boolean parameter.

Syntax

```
aciError acioParamGetBool (
   t_aciObject* pObject,
   char* szName,
   BOOL *pbResult)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that contains an XML result.
`szName`	**`char*`** A pointer to the parameter name.
`pbResult`	**`BOOL*`** A pointer to the parameter's content.

Discussion

This function retrieves the specified `szName` boolean parameter from the object.

Returns

An ACI error code.

acioParamGetDouble

Call the `acioParamGetDouble` function to obtain the contents of a double-precision floating-point parameter.

Syntax

```
aciError acioParamGetDouble (
   t_aciObject* pObject,
   char* szName,
   double *pfResult)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that contains an XML result.
`szName`	**`char*`** A pointer to the parameter name.
`pfResult`	**`double*`** A pointer to the parameter's content.

Discussion

This function retrieves the specified `szName` double parameter from the object.

Returns

An ACI error code.

acioParamGetInt

Call the `acioParamGetInt` function to obtain the contents of an integer parameter.

Syntax

```
aciError acioParamGetInt (
   t_aciObject* pObject,
   char* szName,
   int *pnResult);
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that contains an XML result.
`szName`	**`char*`** A pointer to the parameter name.
`pnResult`	**`int*`** A pointer to the parameter's content.

Discussion

This function retrieves the specified `szName` integer parameter from the object.

Returns

An ACI error code.

acioParamGetLong

Call the `acioParamGetLong` function to obtain the contents of an long integer parameter.

Syntax

```
aciError acioParamGetLong (
   t_aciObject* pObject,
   char* szName,
   long *plnResult)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that contains an XML result.
`szName`	**`char*`** A pointer to the parameter name.
`plnResult`	**`long*`** A pointer to the parameter's content.

Discussion

This function retrieves the specified `szName` long parameter from the object.

Returns

An ACI error code.

acioParamGetString

Call the `acioParamGetString` function to obtain the contents of a string parameter.

Syntax

```
aciError acioParamGetString (
   t_aciObject* pObject,
   char* szName,
   char** szResult)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that contains an XML result.
`szName`	**`char*`** A pointer to the parameter name.
`szResult`	**`char**`** Address of the parameter's content.

Discussion

This function retrieves the specified `szName` string parameter from the object.

Returns

An ACI error code.

Examples

```
aciObjectParamGetString(pResult, ACI_DATA_DATA, &pcResults);
aciObjectParamGetString(pResult, ACI_DATA_NODE_NAME, &szNodeName);
aciObjectParamGetString(pResult, ACI_DATA_NODE_VALUE, &szNodeValue);
```

acioParamGetStringMoveOriginal

Call the `acioParamGetStringMoveOriginal` function to obtain the contents of a string parameter and removes it from the ACI object.

Syntax

```
aciError acioParamGetStringMoveOriginal(
   t_aciObject* pObject,
   const char* szName,
   char** szResult)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that contains an XML result.
`szName`	**`char*`** A pointer to the parameter name.
`szResult`	**`char**`** Address of the parameter's content.

Discussion

This function retrieves the specified `szName` string parameter from the object and removes it from the ACI object.

Returns

An ACI error code.

acioParamGetUnsignedInt

Call the `acioParamGetUnsignedInt` function to obtain the contents of an unsigned integer parameter.

Syntax

```
aciError acioParamGetUnsignedInt(
   t_aciObject* pObject,
   const char* szName,
   unsigned int *pnResult)
```

Arguments

Arguments	Type/Description
`pObject`	**`t_aciObject*`** A pointer to an `ACI_DATA` object that contains an XML result.
`szName`	**`char*`** A pointer to the parameter name.
`pnResult`	**`int*`** A pointer to the parameter's content.

Discussion

This function retrieves the specified `szName` unsigned integer parameter from the object.

Returns

An ACI error code.

aciResponseGetErrorStatus

Call the `aciResponseGetErrorStatus` function to extract any error information from the response of an ACI server.

Syntax

```
aciError aciResponseGetErrorStatus(
   t_aciObject* pResult,
   t_aciErrorStatus* pError)
```

Arguments

Arguments	Type/Description
`pResult`	**`t_aciObject*`** An ACI object that contains the result returned by an ACI server.
`pError`	**`t_aciErrorStatus*`** An ACI error status structure to hold the error information extracted.

Discussion

This function extracts any error information from an ACI object that contains the result of a response by an ACI server. It extracts this information to an ACI error status structure.

Returns

An ACI error code.

Example

```
t_aciErrorStatus* pErrorStatus = NULL;
aciErrorStatusCreate(&pErrorStatus);
aciResponseGetErrorStatus(pResult, pErrorStatus); if (
   pErrorStatus->eResponse != eResponseSuccess) {
   fprintf(stderr, "The error description is %s\n",
pErrorStatus->szErrorDescription);
}
aciErrorStatusDestroy(&pErrorStatus);
```

aciShutdown

Call the `aciShutdown` function to clean up at the end of an ACI client session.

Syntax

```
void aciShutDown()
```

Discussion

This function must be called at the end of the ACI client session.

Appendixes

This section includes the following appendixes:

- Security Information Values
- Error Codes
- OEM Encryption

Appendix A: Security Information Values

You must use the values detailed in the table below to enter security information for the following functions:

- The C function `aciObjectSetUserSecurityInfo`.
- The Java function `SetUserSecurityInfo`.

The action parameters can be required or optional depending on the type of security that you implement. For each security type the required values are:

Security Information Values for Different Repositories

Security Type	Unmapped security	Mapped security
Documentum	• user: Documentum user name • pass: Documentum password	• user: Documentum user name • group: comma-separated list of groups to which the user belongs
eRoom	N/A	• user: eRoom user name • group: comma-separated list of groups to which the user belongs
Exchange	• user: Exchange user profile name • domain: Exchange server domain name	• user: Exchange user name • domain: Exchange domain name
FileNet	N/A	• user: FileNET user name • group: comma-separated list of groups to which the user belongs
iManage	N/A	• user: iManage user name • group: comma-separated list of groups to which the user belongs
Lotus Notes	• user: Notes user name • group: comma-separated list of groups that the user belongs to	• user: Notes user name • group: comma-separated list of groups to which the user belongs
Microsoft NT	• user: NT user name • pass: NT password • domain: NT domain name	If your security type is `AUTONOMY_SECURITY_V4_NT_MAPPED`: • user: *NT_domain_name\user_name* • group: *NT_domain_name\group* **NOTE:** The security string must be escaped.

Security Information Values for Different Repositories, continued

Security Type	Unmapped security	Mapped security
		If your security type is `AUTONOMY_SECURITY_NT_MAPPED`: • user: NT user name • domain: NT domain name • group: comma-separated list of groups to which the user belongs
NetWare	• user: NetWare user name • pass: NetWare password • domain: NetWare domain name	• user: NetWare user name • domain: NewWare domain name • group: comma-separated list of groups to which the user belongs
ODBC	Dependent on the stored function/procedure	N/A
OpenText	• user: OpenText user name • pass: OpenText password	• user: OpenText user name • group: comma-separated list of groups to which the user belongs
Oracle	Dependent on the stored function/procedure	N/A
PCDocs	N/A	• user: PCDocs user name • group: comma-separated list of groups to which the user belongs
SharePoint	N/A	• user: SharePoint user name • group: comma-separated list of groups to which the user belongs
UNIX	N/A	• user: UNIX user name • group: comma-separated list of groups to which the user belongs

Example:

```
t_aciObject* acioCommand = NULL;

aciObjectCreate(&acioCommand, ACI_COMMAND);
aciObjectSetUserSecurityInfo(acioCommand, "nt", ACI_SECURITY_DOMAIN,"AUTONOMY");
aciObjectSetUserSecurityInfo(acioCommand, "nt", ACI_SECURITY_GROUP,"Users");
aciObjectSetUserSecurityInfo(acioCommand, "nt", ACI_SECURITY_USERNAME,"johnsmith");
```

Appendix B: Error Codes

The following error codes can be returned by functions that return an ACI error code.

0	SUCCESS	-8	Encoding error
-1	Bad parameter	-9	Parsing error
-2	Cannot connect	-10	Encryption error
-3	Memory allocation error	-11	Decryption error
-4	Not found	-12	Not implemented
-5	Invalid tag value	-13	Connection Timed Out
-6	Multipart error	-14	Read Timed Out
-7	File not found	-15	Unclassified error

Appendix C: OEM Encryption

This section describes how to use OEM licensing with IDOL products.

Overview

The OEM licensing scheme restricts access to IDOL by applying ACI encryption to all requests, so that only the OEM application, which knows the encryption keys, can communicate with IDOL. This enables Micro Focus to provide a single license with no host or port restrictions that the OEM can distribute with all copies of their software. Even though the OEM license can be used to start IDOL on any machine, end users (customers of the OEM) cannot abuse this license because they do not have the encryption keys required to communicate with IDOL.

Restrictions

- Unlike a standard License Server-based license, there are no host, IP, MAC address or port restrictions on the OEM license.
- ACI communication is possible only between the OEM application and the IDOL component, or between the IDOL components that share the OEM license. This is enforced by using ACI encryption.
- Index and service port actions are not restricted by any form of encryption.

Setting Up Licensing

All ACI communication actions sent to IDOL components in an OEM-licensed environment must be ACI encrypted. IDOL components access the `licensekey.dat` file to determine the encryption keys required to decode encrypted action.

> **NOTE:**
> You must not use the `CommsEncryptionType` and `CommsEncryptionTEAKeys` (deprecated) configuration parameters to encrypt ACI communications because the key would be publicly available.

To set up licensing in an OEM environment

- Encrypt ACI communications between IDOL components and the front-end application by making the appropriate API call in your application and passing in the OEM encryption keys that were provided to you. For example:

  In the C API, you could make the following call:

```
char* szKeys = "MjR8CJCUGcb4RbRdNDKbK9RXX3pEswAiZ";
aciInitEncryption(TRUE, "TEA", szKeys);
```

where the value of `szKeys` is the encryption key provided with the license.

In the Java API, you could make the following call:

```
TEAEncryptionDetails encryptionDetails = new TEAEncryptionDetails();
encryptionDetails.setEncryptionKeys(MjR8CJCUGcb4RbRdNDKbK9RXX3pEswAiZ);
encryptionDetails.setEncrypting(true);
aciConnection.setEncryptionDetails(encrytionDetails);
```

- Redistribute the `licensekey.dat` file with your application by copying it to the working directory of each IDOL component. The IDOL components will then read the license from the `licensekey.dat` instead of the `[License]` section of the component's configuration file.

  The `licensekey.dat` file is generated by Micro Focus and provided to you along with your license.
- Unlike standard ACI encryption, the ACI responses from the IDOL component are not encrypted by default. You can optionally encrypt the response returned by an IDOL component by setting the `EncryptResponse` action parameter to **`True`** in the ACI action you run. However, in most cases HPE recommends that you use SSL/TLS for secure communications.

Glossary

A

ACI API
The programming interface for HPE IDOL products. You can use this interface to create custom applications that use ACI servers, such as the IDOL Content component. IDOL SDKs are available for several different programming languages to allow you to develop with the ACI API.

ACI Server
A server component that runs on the Autonomy Content Infrastructure (ACI). The ACI API controls communications with ACI servers.

action
A request that can be sent to ACI Servers, for example a query.

Autonomy Content Infrastructure (ACI)
A technology layer that automates operations on unstructured information for cross-enterprise applications. ACI enables an automated and compatible business-to-business, peer-to-peer infrastructure. The ACI allows enterprise applications to understand and process content that exists in unstructured formats, such as email, Web pages, Microsoft Office documents, and IBM Notes.

C

connector
An IDOL component (for example File System Connector) that retrieves information from a local or remote repository (for example, a file system, database, or Web site).

D

data index
An IDOL Server index that stores content data. You can customize how to store data in the Data index by configuring appropriate settings in the IDOL Server configuration file.

database
An IDOL Server data pool that stores indexed information. The administrator can set up one or more databases, and specifies how data is fed to the databases. By default contains the databases Profile, Agent, Activated, Deactivated, News and Archive.

I

IDOL SDK
DIOL Software Development Kit. A programming language-specific binding that allows you to develop applications that use the ACI API to communicate with ACI servers. HPE releases IDOL SDKs for C, Java, and .NET.

IDOL Server
The Intelligent Data Operating Layer (IDOL) Server, which integrates unstructured, semi-structured and structured information from multiple repositories through an understanding of the content. It delivers a real-time environment in which operations across applications and content are automated.

IDX
A structured file format that can be indexed into IDOL Server. You can use a connector to import files into this format or you can manually create IDX files.

indexing
The process of storing data in IDOL Server. IDOL Server stores data in different field types (such as, index, numeric and ordinary fields). It is important to store data in appropriate field types to ensure optimized performance.

Intellectual Asset Protection System (IAS)
An integrated security solution to protect your data. At the front end, authentication checks that users are allowed to access the system that contains the result data. At the back end, entitlement checking and authentication combine to ensure that query results contain only documents that the user is allowed to see, from repositories that the user has permission to access.

Q

query
A string that you submit to IDOL Server, which analyzes the concept of the query and returns documents that are conceptually similar to it. You can submit queries to to perform several kinds of search, such as natural-language, Boolean, bracketed Boolean, and keyword.

X

XML
Extensible Markup Language. XML is a language that defines the different attributes of document content in a format that can be read by humans and machines. In IDOL Server, you can index documents in XML format. IDOL Server also returns action responses in XML format.

Send documentation feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on Programming Guide (Micro Focus ACI API 11.0)

Add your feedback to the email and click **Send**.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to AutonomyTPFeedback@hpe.com.

We appreciate your feedback!