# CA Output Management Web Services

## Developer Guide

Version 2.0

# CA Technologies Product References

This document references the following CA Technologies products:

- CA View® Output Archival and Viewing (CA View)
- CA Bundl®
- CA Dispatch™ Output Management (CA Dispatch)
- CA View
- CA Deliver

# Contact CA Technologies

**Contact CA Support**

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

**Providing Feedback About Product Documentation**

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

# Contents

# Chapter 1: Introduction

The CA Output Management Web Services (CA OM Web Services) provides several web services. These web services enable you to access programmatically, extract, and download the reports that are stored in CA mainframe report archival, distribution, and viewing products. Benefits of the CA OM Web Services are:

- Accessing content and reports in Host Output Management products:
  - CA View® Output Archival and Viewing (CA View)
  - CA Bundl®
  - CA Dispatch™ Output Management (CA Dispatch)
- Easy to modify and extend, reducing long development cycles.
- Proprietary software is not required.
- Supportive of industry standard technologies including HTTP/HTTPS, SOAP, UDDI, WSDL, and XML.
- Platform independence.
- Custom integration of Host Output Management product content.

This *Developer Guide* is intended for the CA OM Web Services programmers.

This section contains the following topics:

# The Web Services

The services that the OM Web Services provide are:

- Report List
- Report Data
- Index List
- Ping

The *Report List Web Service* returns report attributes for one or more reports in a repository of Host Output Management product system. The input request specifies the desired attributes that are returned and the selection criteria that is used to select the reports.

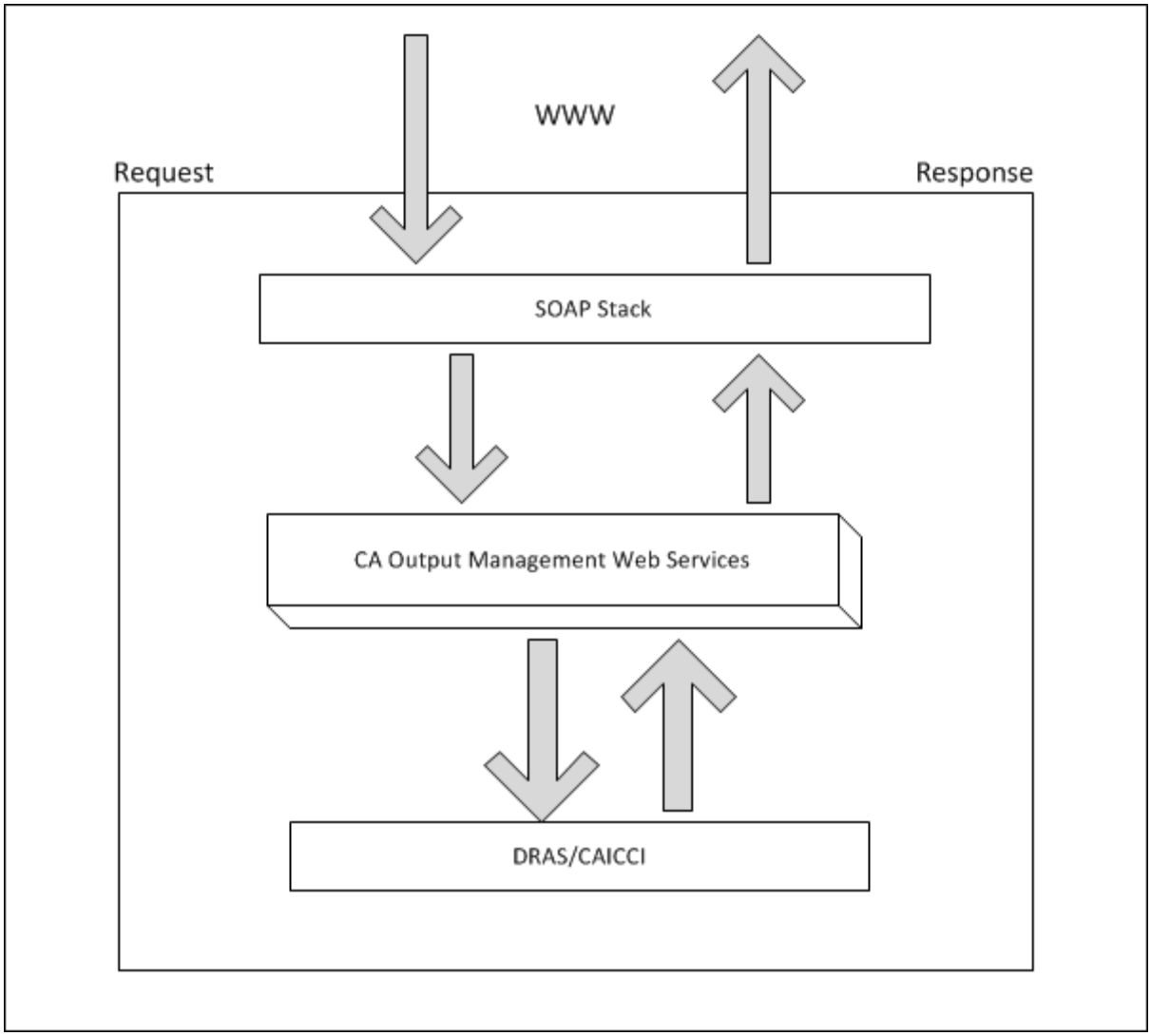
The *Report Data Web Service* returns a report. If the report is a text report the entire report or selected pages are retrieved. If the report is AFP or binary, the entire report is returned. If the report is AFP retrieval by an index the entire section of the report matching the index is retrieved and page selection is not supported. If the report is AFP retrieval with a report type AFP2PDF for transformer support, then the PDF report is retrieved and returned as an attachment. The AFP2PDF report type is only valid if the underlying Output Management product supports the AFP to PDF transformation. The Report Data Web Service retrieves report data for processing and does not provide display facilities.

The *Index List Web Service* returns index information for repositories and reports. Index list requests are only valid if the underlying Output Management product supports indexing. For example, CA Dispatch does not provide indexing; Index List requests are not performed if the repository specified is for CA Dispatch. The Index List Web Service request supports two types of indexes—cross-report indexes and regular indexes. To return cross-report index information, the underlying repository must support them. In this case, CA View is the only application that supports cross-report indexing. The Index List request returns the cross-report index names for a repository, the reports that match a particular cross-report index value, index names for a report and the sections of a report that match a particular index value.

The *Ping Web Service* returns a string indicating the Web Service has started. *Ping Web Service* is the simplest Web Service, provided to test for the correct Web Service operation and setup.

# Processing Data with Web Services

The following diagram is an overview of the OM Web Services that shows how data flows into the Web Service, is processed and then returned:



1. The SOAP receives the request using HTTP or HTTPS.
2. The SOAP stack receives the data and invokes one of the OM Web Services.
3. The OM Web Services process the request. These Web Service requests send requests to the Host Output Management product to retrieve data using the Distributed Repository Access System (DRAS) and CA International Common Communications Interface (CAICCI) components.
4. Data that is retrieved by the OM Web Services request is formatted for return to the client application.
5. The CA OM Web Service returns to the SOAP stack.
6. The SOAP stack transforms the data into SOAP syntax for the transmission back to the client application and transmits the data.

**Note:** For each Web Service API call, an individual DRAS connection is created. A DRAS connection will be established to execute each API call and be disconnected after the call, which can incur a significant performance overhead.

To reduce the connection cost, a local cache method is recommended to store the returned report lists and report data in the local memory of the Web Service client applications. Depending on the timeliness of your application data, you can construct this cache to be refreshed periodically.

**Note:** For more information, see Terms Used in this Guide (see page 11).

# Terms Used in this Guide

The following list contains definitions, acronyms, and abbreviations that are used in this guide:

- CA International Common Communications Interface (CAICCI)

  CAICCI offers a simple yet flexible approach enabling the CA products to communicate with one another. This facility provides a layer that isolates the application software from the specifics of the communications environment.

- Distributed Repository Access System (DRAS)

  DRAS is the Output Management component that makes requests to the Host Output Management product (CA View, CA Bundl, or CA Dispatch). DRAS has a client and server component. The client runs on the same platform as the OM Web Servicesand the server runs on the mainframe.

- Host Output Management product

  Host Output Management product refers to the CA OM mainframe products as a group rather than individually. These products are CA View, CA Dispatch and CA Bundl.

- Hypertext Transfer Protocol (HTTP)

  HTTP, an application protocol running on top of the TCP/IP protocol suite, is a set of rules for transferring files. HTTPS (Hypertext Transfer Protocol over Secure Socket Layer) encrypts and decrypts requests.

- Java Archive (JAR)

  A JAR file allows multiple files to be placed into a single archive file. (The JAR files are similar in the concept to .ZIP files, except they are a standard file format for holding Java file types.) A JAR file usually contains the class files and auxiliary resources that are associated with applets and applications.

- Simple Object Access Protocol (SOAP)

  SOAP is an industry standard from the World Wide Web Consortium organization (W3C) (WWW.W3 .ORG). SOAP is an XML-based language that provides a way for two or more applications to communicate together over the Internet. SOAP is used to encode data that is sent between two applications. All data that are sent between the OM Web Services and client programs is formatted as SOAP messages.

- Web Application Archive (WAR)

  A Web application can be packaged and run on another server when it is in a WAR. In addition to web components, a WAR usually contains other files including the following:

  - Server-side utility classes (database beans, shopping carts, and so on) often conform to the JavaBeans component architecture.
  - Static Web content (HTML, image, and sound files, and so on)
  - The Client-side classes (applets and utility classes)

- Web Services Description Language (WSDL)

  WSDL is an industry standard from the W3C (WWW.W3.ORG). WSDL is an XML-based language that defines web services capabilities. It defines Web service requests and the details of each request's input and output parameters. Most tools that create a client access to Web Services do so using the WSDL for the Web Service.

- Extensible Markup Language (XML)

  XML is an industry standard from the W3C (WWW.W3.ORG). XML, a markup language that is designed to describe data, is similar in format to HTML, rather than being made up of a fixed set of predefined tags, XML allows the developer to design their own. SOAP and WSDL are the XML languages. The OM Web Services return data in XML syntax.

# Chapter 2: Preparing to Write Code

This chapter contains design and code preparation items for writing code to access OM Web Services.

This section contains the following topics:



## Accessing Web Services WSDL

Once you have the OM Web Services that are installed and configured, you can view its WSDL through your browser. Enter the following URL:

http://servername:port/caomws20/services/OmWebServiceService?wsdl

The first Web Service request that is issued processes the Web Service's initialization code. The first request has more overhead than subsequent requests.

## Access Web Services through a Code

The choice of technology for writing programs that access the OM Web Services depends on corporate standards and the type of application being built. After selecting a technology to implement an application, there are often many stack technologies or interfaces available for creating code to access a Web Service. For example, in Java several technologies are available for writing code to access Web Services. One choice, the Apache Axis2 SOAP stack, includes classes to invoke Web Services and is included with the OM Web Services

**Note:** The Apache Axis2 SOAP stack is also the SOAP stack that is discussed in this guide.

The following table provides a list of technologies that can be used for writing applications that access OM Web Services:

| Technology Choice | Technologies and Interfaces Available |
|---|---|
| Java | Apache Axis2<br>Apache CXF |
| Perl | SOAP:Lite for Perl |
| PHP | PHP SoapClient (built-in) |

| Technology Choice | Technologies and Interfaces Available |
|---|---|
| .NET MS Office | Microsoft WCF |
| C++ | gSOAP |

**Note:** This is not an endorsement of those technologies or an indication that we have tested and certified them for this use.

These technologies help programmers build applications that use the OM Web Services. They often hide the SOAP request generation and SOAP response processing by converting structures and classes that are passed to them into SOAP requests (serialization) and converting the SOAP response into structures and classes in the native language (de-serialization). These technologies provide tools for using the OM Web Services WSDL statement as input and generate code to access the OM Web Services.

Two of the technologies available for writing applications that access the OM Web Servicesare discussed in this guide – Java and .NET platform. The Java programmers use Java with the Apache Axis2 engine. Microsoft Visual Studio uses built-in Web Services support to generate Visual Basic or C# classes that can be accessed from custom code and integrated into Office applications as a COM component.

## Java Programs and Automatic Code Generation using Apache Axis

Apache Axis2 provides a utility (WSDL2Java) that generates the client code to invoke OM Web Services. WSDL2Java receives the name of the WSDL file for the Web service as an input parameter. The WSDL2Java utility generates Java programs that access the web services that are defined in the WSDL. The generated code can then be used to invoke the OM Web Services.

Apache Axis2 generates the following files when processing the CA OM WSDL file:

| File Name | Description |
|---|---|
| ExtensionMapper.java | Helper class for nested parameter types |
| OmIndexListParameter.java | Index list request parameter |
| OmReportDataParameter.java | Report data request parameter |
| OmReportSelectionArray.java | Array of selection criteria |
| OmReportSelectionItem.java | Selection criteria type |
| OmReportSelectionParameter.java | Report list request parameter |
| OmResponse.java | All operations response type |
| OmStringArray.java | Nested response type |

| File Name | Description |
|---|---|
| OmVector.java | Nested response type |
| OmWebServiceServiceStub.java | Client-side service stub |

Programmers writing code in Java to access OM Web Services can choose to use the automatic code generation using Apache Axis2 or generate these classes. They can then use these classes in their code.

An alternative to automatic code generated using Apache Axis2 is a set of classes and wrapper classes that are provided with the OM Web Services. A wrapper class invokes each OM Web Services. The wrapper classes and the classes from the previous table are in the client.jar installed during the OM Web Services installation. Classes in the client.jar file facilitate writing Java code to access the CA OM Web Services.

**Note**: For more information about these wrapper classes, see the chapter, "Wrapper Classes" in this document.

## Visual Basic and C# Programs

Microsoft Visual Studio Express 2012 For Windows Desktop provides a standard utility that can be pointed at the WSDL and can generate Visual Basic or C# code to access the OM Web Services. Using the Microsoft Visual Studio removes most of the complexity that is associated with writing a Visual Basic or c # application.

Microsoft Visual Studio Tools for Office provides an official solution for further work with data retrieved from Web Services in Microsoft Office.

The NetOffice provides an open source alternative for the Microsoft Office integration without version limitations (2000, 2002, 2003, 2007, 2010, and 2013).

# Chapter 3: Understanding the Web Services

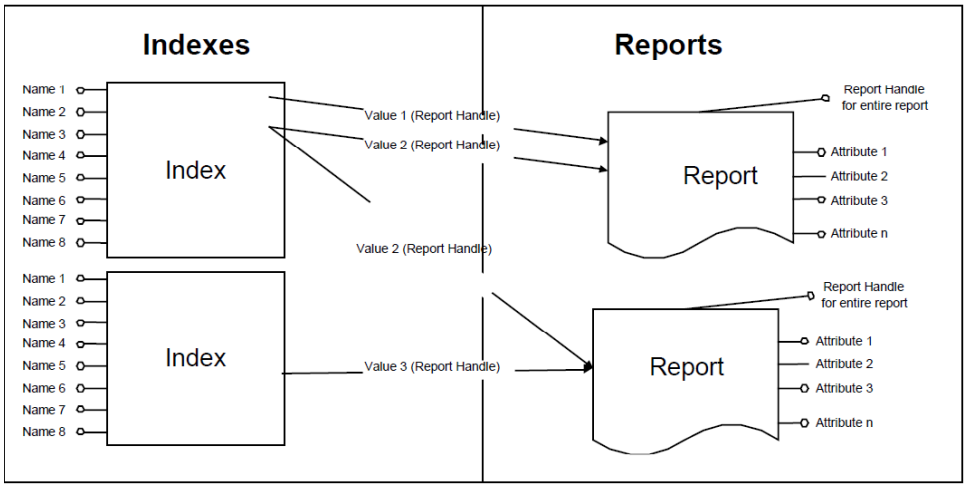
This chapter describes the relation between repositories, indexes, and reports. It also provides descriptions for each of the OM Web Services, their input parameters, and data returned.

This section contains the following topics:



## Repositories, Indexes and Reports

Repositories can hold reports and indexes. The following diagram represents a repository:



## Repositories

Repositories are entities in the Host Output Management product environment where reports are stored. The repositories that a web service can access are defined in the repository definition file.

The repository definition file is created during the post-installation of the Web service or when making changes to the Host Output Management product (such as applying a service pack, installing a new release of the product, or adding, removing, or changing repositories for the product).

## Reports

Reports consist of data. The Report Data Web Service retrieves the data for a report or the pages of a report.

Reports have attributes; a set of fields defines these attributes. The Report List Web Service selects a series of reports and retrieves the values for selected report attributes.

**Note:** For more information about the attributes of the Report List Web Service, see the Customizing the Report List Web Service (see page 75).

Every report has a unique report handle attribute assigned to reference it. If the repository supports indexes, an extra report handle references those pages that are associated with an indexed value. The Report List Web Service and Index List Web Service retrieve a unique report handle for every report in a repository. These handles are then used to access the reports in the Report Data Web Service.

## Indices

Indexes are entities that enable referencing of sections (one or more pages) of reports. For example, you can reference the section of the checking statement report for a particular account number. Locating the pages of the report requires matching an index name that is based on a particular value.

Each index is assigned a unique name; the name can have up to eight different parts. In the preceding diagram, index names are represented as Name1, Name 2, Name 3, and so forth. For example, an index can be named "ACCOUNT#" or "ACCOUNT#, MONTH, YEAR".

An index that spans multiple reports is called as a cross-report index. Index 1 in the diagram is a cross-report index; it is an index for both Report 1 and Report 2. An index references a section of a report. A report handle references as an index. The Index List Web Service returns report handles to pages of reports that are based on index or cross-report indexes.

The Index List Web Service retrieves the list of cross-report indexes in a repository, the report handles for sections of reports for an index matching a specific value, the list of indexes for a report, and the section of a report that is indexed by a specific value.

**Note:** The CA Dispatch repositories do not hold indices.

# The Report List Web Service

The Report List Web Service returns a list of attributes for one or more reports in a repository. The request specifies an input parameter that includes the attributes to return and the selection criteria. Data can be returned in XML or native formats.

The following table describes input parameters for the Report List Web Service:

| Request Name | Report List | Required | Case sensitive |
|---|---|---|---|
| Request Format | userid: string | Y | Y |
| | password: string | N | Y |
| | returnas: (values can be: "vector", "stringarray", "xmlattachment", "xmlstring") | N<br>default:vector | N |
| | repository: string with name of repository entry in repository definition file | Y | Y |
| | fields: an array of one or more strings specifying report attribute fields | Y<br>(atleast one field) | N |
| | selectionCriteria: array of name-value pairs | N | N(field names)<br>Y(values) |
| Request Response | Based on return type selected | | |

## Input Parameters

A Report List request returns report information for reports that are assigned or accessible to the user ID in the Host Output Management product. If the user ID can view only a particular report, only information for that report gets retrieved.

**Userid**

Specifies the user ID associated with the request. The userid specifies a user ID defined to the Host Output Management product where the report is stored. A Report List request requires the userid parameter.

**Password**

Specifies the password that is associated with the user ID for the request. The password matches the password that is associated with the user ID defined to the Host Output Management product. If no password is assigned, specify null.

**Repository**

Specifies the name of an entry in the repository definition file and identifies a specific repository in the Host Output Management product. The Report List Request retrieves information for reports that are held in that repository only. If information is required for reports on other repositories, make secondary requests. This case-sensitive parameter must match the case of the repository name in the repository definition entry. Contact your System Administrator for the name of the repository definition entry.

**Returnas**

Specifies the format of the data returned. For example, a returnas value of "xmlattachment" returns the data from the Report List request as a file attachment in XML format. The Returnas values that are supported for the Report List Web Service are:

- *Vector* data is returned as a vector. The client code supports the vector data types. If a vector is specified as a returnas value. Java supports Vector data.
- StringArray data is returned as a string array. The client code supports the data that is returned as string arrays. Java and the Visual Basic client code support a string array.
- *XMLString* data is returned as a single string containing the data retrieved by the Web Service request. The string is a fully formatted XML document.
- *XMLAttachment* data is returned as a file attachment. The file is in XML format and contains the data that is retrieved by the Web Service request.

The returnas parameter that is specified is not case-sensitive; it can be upper, lower, or mixed case. If the returnas parameter is omitted its default value is set to "Vector".

**Fields**

Specifies the attributes (fields) to be returned and is based on the requirements of the application being developed. This parameter must pass at least one attribute (field name) or the request fails. The fields parameter that is passed to the Web Service is a list of the attributes indicating the information to be returned for each report in a repository. In Java, each field would be delimited with double quotation marks and separated by commas. For more information about the attributes that can be specified in the fields parameter, see the appendix "Customizing the Report List Web Service" in this guide.

**Selection criteria**

Specifies a series of rules to determine the reports (in the specified repository) for which information is retrieved. If selection criteria are omitted from the parameter list, all of the reports accessible by the user ID in the repository are selected. For example, selection criteria can be specified to return information for a specific report name, such as the number of lines in the report named "creditcardsummary".

Selection criteria are name-value pairs; the name is a field name (uses same field name keywords as specified for the field parameter) and the value is the value that must be matched. Values in the parameters are specified as character strings; numeric data values, dates, and times are specified as numeric values, such as "1", "01/01/01", "12:00:00:00".

Use the following name-value pair to return the number of lines for a report named "creditcardsummary":

| Name | Value |
|---|---|
| RID | "creditcardsummary" |

The keyword RID designates the report name field and creditcardsummary is the report name in the repository.

There are several special rules for the selection criteria specification:

1. If the field is comprised of characters, the value that is specified must be an exact match or a partial match using an "*" or "?" character to create a mask character:

| Name | Value |
|---|---|
| RID | "creditcardsummary" |
| RID | "c*" |

"c*" returns information for all reports whose report names begin with *c*.

2. If the field is numeric, date or time, the value must be an exact match:

| Name | Value |
|---|---|
| LINES | "50" |
| ARCDATE | "09/05/2003" |

Specifying "50" for the LINES criteria returns information for all reports that have 50 lines. Specifying "09/05/2003" for the ARCDATE criteria retrieves information for reports that are archived on September 5, 2003.

3. If the field is numeric, date or time, the value can be in a range of values:

| Name | Value |
|---|---|
| LINES | "50,100" |
| ARCDATE | "01/01/2004,09/05/2004" |

Specifying "50,100" for the LINES criteria returns information for all reports that have from 50 to 100 lines inclusive. Specifying "01/01/2003, 09/05/2004" for the ARCDATE criteria retrieves information for reports that are archived between January 1, 2004 and September 5, 2004.

4. CRXNAME, CRXVALUE are special criteria names that relate to cross-report index names and value specification rather than report information fields. For the Report List request, specifying a CRXNAME requires also specifying a CRXVALUE.

   **Note:** CRXNAME and CRXVALUE are only supported for CA View. CRXNAME specifies one to eight parts of a cross-report index name. CRXVALUE specifies a cross-report index value. If the CRXNAME criteria specify a two-part index, then the CRXVALUE must specify two values to match.

   For example, a group of reports can support a three-part cross-report index (ACCOUNT#, MONTH, YEAR). To retrieve information for all March 2004 reports for account number 12345, the CRXNAME and CRXVALUE are:

| Name | Value |
|---|---|
| CRXNAME | "ACCOUNT#,MONTH,YEAR" |
| CRXVALUE | "12345,MARCH,2004" |

To retrieve information for all reports for account number 12345 for all months of all years, the CRXNAME for specific index names and CRXVALUE with wildcards are:

| Name | Value |
|---|---|
| CRXNAME | "ACCOUNT#,MONTH,YEAR" |
| CRXVALUE | "12345,*,*" |

5. Specify dates values as *DD/MM/YYYY.*
6. Specify time values as *HH:MM:SS:nn* or *HH:MM:SS* which is the same value as *HH:MM:SS:00*.
7. Specify a selection criteria that specifies a recipient using the RECIPIENT field, if the Host Output Management Product is CA Dispatch.
8. Specify a selection criteria that specifies a mailcode using the MAILCODE field, if the Host Output Management Product is CA Bundl.

## The Data Returned

The Report List Web Service request returns lists of attributes for selected reports in repositories. The attributes that are returned are requested by the client application using the *fields* input parameter. The client application can request a single field of information or multiple fields. If more than 50 field names are specified the request fails with a WEBSER051E error. The selection criteria can select a single report or multiple reports.

Based on the request being made and the number of reports that are selected, the data content that is returned differs as follows:

- If the returnas value is "Vector", there is a vector entry for each report. The vector entry is a string array with an entry for each field.
- If the returnas value is "StringArray", there is a string array entry for each report. The string array entry is another string array with an entry for each field.
- If the returnas value is "XMLString", there is an entry for each report. The entry includes entries for each field.
- If the returnas value is XMLAttachment, it contains a file with the same data that is returned by XMLString.

For example, a request to return data fields for the text reports beginning with the letter *c* in the repository that defines the repository definition file named "TestRepository" contains the following parameters:

| Request Parameter | Value |
|---|---|
| Userid | "omuser" |
| Password | "ompwd" |
| Repository | "TestRepository" |
| ReturnAs | "vector" |
| Fields | "RID","GEN","SEQ","DOC_TYPE","LPAGES","LINES", "ARCDATE","ARCTIME" |
| Selection Criteria | DOC_TYPE="TEXT"<br>RID="C*" |

The request returns the following data:

| Item | RID | GEN | SEQ | DOC_TYPE | LPAGES | LINES | ARCDATE | ARCTIME |
|---|---|---|---|---|---|---|---|---|
| 1 | CHARRPT3 | 19 | 19 | TEXT | 1 | 41 | 3/1/00 | 8:17:55:29 |
| 2 | CHARRPT4 | 19 | 18 | TEXT | 1 | 41 | 2/29/00 | 11:22:17:03 |

Specifying a returnas value of "xmlstring" returns the following data for the text reports beginning with the letter *c* in the repository that defines the repository definition file named "TestRepository":

```
<reportlistdata>
    <report>
        <rid>CHARRPT3</rid>
        <gen>19</gen>
        <seq>19</seq>
        <doc_type>TEXT</doc_type>
        <lpages>1</lpages>
        <lines>41</lines>
        <arcdate>3/1/00</arcdate>
        <arctime>8:17:55:29</arctime>

<rpt_handle>c3c8c1d9d9d7e3f340404040404040404040404040ffecffec00000000000000000000

000</rpt_handle>
    </report>
    <report>
        <rid>CHARRPT4</rid>
        <gen>19</gen>
        <seq>18</seq>
        <doc_type>TEXT</doc_type>
        <lpages>1</lpages>
        <lines>41</lines>
        <arcdate>2/29/00</arcdate>
        <arctime>11:22:17:03</arctime>

<rpt_handle>c3c8c1d9d9d7e3f340404040404040404040404040ffecffed00000000000000000010

000</rpt_handle>
    </report>

</reportlistdata>
```

# The Report Data Web Service

If the reports are binary or AFP, the Report Data Web Service returns pages of a report or the entire report. A Text report can be returned in an XML attachment or in native data formats. Binary and AFP reports are returned as attachments. Support for page range selection varies by report type; AFP and distributed reports (binary) do not support retrieval by page numbers while text reports do. Support for AFP to PDF transformation with a new report type, AFP2PDF, has been introduced in this release.

The following table describes request and response formats for the Report Data Web Service:

| Request Name | Report Data | Required | Case-sensitive |
|---|---|---|---|
| Request Format | userid:string | Y | Y |
| | password:string | N | Y |
| | respository:string with name of respository entry in repository definition file | Y | Y |
| | type:string (TYPE, BINARY, AFP, AFP2PDF) | N(default TEXT) | N |
| | textreportformat:(PAGE,LINE) | N(default PAGE) | N |
| | pageto:string | N(default999999999) | N |
| | pagefrom:string | N(default1) | N |
| | returnas: (value can be: "vector","stringarray","xmlattachment", "xmlstring","attachment") | N(default vector) | N |
| | reporthandle:string | Y | N |
| Request Response | Based on report type selected | | |

## Report Data Web Service Input Parameters

**Userid**

Specifies the user ID associated with the request. Userid specifies a user ID defined to the Host Output Management product. The Report Data Web Service request requires the userid parameter. The Report Data request returns reports assigned or accessible to the specified user ID through the Host Output Management product. If the user ID can only view a particular report, Report Data requests for a different report fail because the user is not defined for that report in the host product.

**Password**

Specifies the password that is associated with the user ID for the request. The password matches the password that is associated with the user ID defined to the Host Output Management Product. If no password is assigned, specify null.

**Repository**

Specifies the name of an entry in the repository definition file and identifies a specific repository in the Host Output Management Product. The Report Data request retrieves information for reports that are held on that repository only. If information is required for reports on other repositories, make secondary requests. This parameter is case-sensitive and must match the case of the repository name in the repository definition entry. Contact your System Administrator for the name of the repository definition entry.

**Returnas**

Specifies the format of the data returned. (For example, a returnas value of "xmlattachment" returns the data from the report list request as a file attachment in XML format.) The Returnas values that are supported for the Report Data Web Service are:

- *Vector* data is returned as a vector. If the vector is specified as a returnas value, the client code supports vector data types. Java supports the Vector data.
- *StringArray* data is returned as a string array. The client code supports the data that is returned as string arrays. The Java and Visual Basic client code supports the String arrays.
- *XMLString* data is returned as a single string containing the data retrieved by the Web Service request. The string is a fully formatted XML document.
- *XMLAttachment* data is returned as a file attachment. The file is in XML format and contains the data gets retrieved from the Web Service request.
- Attachment data is returned as a file attachment. Returning these file attachments is supported for binary and AFP reports, but is not supported for text reports. The attached file holds the report data. An AFP report returns binary data for the report in proprietary AFP formatting and can only be accessed by programs that understand the AFP format. AFP reports can return as PDF reports if the host Output Management product, CA View, supports the AFP transformation. A Binary report holds binary data and based on the initial source of the data it can only be processed by programs that understand the data source. For example, a binary file can be a JPEG image and Microsoft Paint can read and display the file.

  The returnas parameter that is specified is not case-sensitive; it can be upper, lower, or mixed case. If, the returnas parameter is omitted its default value is set to "Vector".

**Type**

Specifies the type of report requested. Supported report types are text, binary, AFP, and AFP2PDF. The type parameter is not case-sensitive; it can be upper, lower, or mixed case.

The following list describes each report type:

**TEXT**

The Host Output Management product stores the report being retrieved in text form.

**BINARY**

The Host Output Management product stores the report being retrieved as a distributed file. This file can be loaded using the LPD command.

**AFP**

An AFP report that is originally destined for an IBM AFP printer is the report being retrieved.

If the type parameter, is omitted it defaults to "TEXT".

**AFP2PDF**

An AFP report transformed to PDF is retrieved as PDF reports if the Host Output Management product supports the AFP to PDF transformation.

**TextReportFormat**

Applies to requests where the Type is specified as TEXT or the Type has defaulted to TEXT because the Type was not specified. The TextReportFormat parameter allows the client application to specify the data to be returned for text reports. PAGE and the LINE options are supported. The TextReportFormat parameter is not case-sensitive; it can be upper, lower, or mixed case.

The following table describes the PAGE and LINE options that are supported for the TextReportFormat parameter:

**PAGE**

Retrieves the text report as formatted, composed pages. Each page is broken into multiple lines. The original line carriage control is evaluated and used to compose the page.

**LINE**

Retrieves the text report as a row of data. Data that are returned for each row includes:

- Report Data Line
- Page Number
- Line Number
- Line in Page
- Carriage Control

If the TextReportFormat parameter is omitted, it defaults to "PAGE".

**Pagefrom**

Applies to requests where the Type is specified as TEXT. If the Type is not specified, then the type has defaulted to TEXT. The Pagefrom parameter specifies the lower page range for the report selection. If the Pagefrom parameter is omitted, it defaults to 1. The Pagefrom parameter must be a numeric string whose value is greater than 0. The Pagefrom value must be lower than the Pageto value. If you specify a Pagefrom value higher than the actual number of pages in the report, the request returns no data.

**Pageto**

Parameter applies to requests where the Type is specified as TEXT. If the Type is not specified, then the type has defaulted to TEXT. The Pageto parameter specifies the upper page range for the report selection. If the Pageto parameter is omitted, it defaults to 999999999. The Pageto parameter must be a numeric string whose value is greater than 0. When retrieving data, the Pageto value specifies the highest page to be retrieved from the report. Requests that specify a large page range take more time to retrieve. The Pageto value must be greater than the Pagefrom value. If you specify a Pageto value greater than the number of the pages in the report, the request stops after the last page of the report.

**Reporthandle**

Specifies a required parameter for the Report Data requests. The report handle is a string that uniquely identifies every report in a Host Output Management repository. The Report List Web Service request or the Index List Web Service request retrieve report handle and passes them to Report Data requests through this parameter.

## Data Returned by the Report Data Web Service

The Report Data Web Service request returns the contents of a report or distributed file from a repository. If the report is text, the client application specifies the pages of the report to be returned. If the report is AFP, or binary, the entire report is returned in a file attachment.

The data that returned is based on the following parameter specifications:

- TextReportFormat specified as "Page"
- TextReportFormat specified as "Line"
- Type specified "AFP", "AFP2PDF" "BINARY", "TEXT"

### Returning TextReportFormat Specified as "Page"

TextReportFormat specified as "Page" can have returnas values of "Vector", "StringArray", "XMLString" or "XMLAttachment":

- If the returnas value is "Vector", there is an entry in the vector for each page and each line of the report. Each vector entry contains a two-entry string array.
  - The first entry in the string array indicates whether the entry is for a page (has constant of "page") or a line (has a constant of "line").
  - If the first entry in the string array is for a page, the second entry is the page number.
  - If the first entry in the string array is for a line, the second entry is the report line.
  - If the first entry is "/page" or the end of the vector has occurred, then the page has ended.

- If the returnas value is "StringArray", there is a string array entry for each report. Each string array entry contains another two-entry string array.
  - The first entry in the string array indicates whether the entry is for a page (has constant of "page") or a line (has a constant of "line").
  - If the first entry in the string array is for a page, the second entry is the page number.
  - If the first entry in the string array is for a line, the second entry is the report line.
  - If the first entry is "/page" or the end of the vector has occurred, then the page has ended.
- If the returnas value is "XMLString", then an XML-formatted string is returned. Pages are designated by a <Page> tag and lines are designated within the page with a <Line> tag. The XML String includes entries for all pages that are requested through the input request parameters.
- If the returnas value is "XMLAttachment", it contains a file with the same data that is returned by "XMLString".

## Returning TextReportFormat Specified as "Line"

TextReportFormat specified as "Line" can have returnas values of "Vector", "StringArray", "XMLString" or "XMLAttachment":

- If the returnas value is "Vector", there is an entry in the vector for each line of the report requested. Each vector entry contains a string array with five entries; there is one vector entry for each line in the report for each of the following entries:
  - Text
  - Page Number
  - Line Number

    (Specifies the actual line number in the report and spans page boundaries.)
  - Line in Page

    (Specifies the line number on the page and is reset on each page.)
  - Carriage Control
- If the returnas value is "StringArray", there is a string array entry for each report. The string array entry is, in turn, another string array with an entry. For each line in the report, there is one string array entry.
- If the returnas value is "XMLString", then an XML-formatted string is returned. A <reportline> tag represents each report line. Within he <reportline></reportline> tags are the following tags:

  <reportdataline> Text

  <pagenumber> Page Number

  <linenumber> Line Number

  (Specifies the actual line number in the report and spans page boundaries).

  <pageline> Line in Page

  (Specifies the line number on the page and is reset on each page).

  <cc> Carriage Control
- If the returnas value is "XMLAttachment", it contains a file with the same data as returned by "XMLString".

## Returning AFP or Binary Reports

If the report is AFP, or binary, it can be returned in the following formats:

- If the returnas value is "Attachment" a file with the entire AFP, or binary report is returned to the client application. The client application must be coded to accept an attachment.
- If the returnas value is "Vector" there is an entry in the vector for each binary line of the report. (These lines have no correspondence to the printed or viewable report.) Each vector entry contains a single entry string array. The single entry holds a displayable representation of the binary data. The client program converts the data to its binary format and then written to a file. For example, a data line of "5a0008d3a8cd000001" is a character string representing the hex string 0x5a0008d3a8cd000001. The Data lines that are returned are of varying length.
- If the returnas value is "StringArray", a string array entry for each binary line of the report is returned. These lines have no correspondence to the printed or viewable report. Each string array entry contains a single entry string array. This second string array entry contains a displayable representation of the binary data. The client program converts the data back to its binary format and then written to a file. For example, a data line of "5a0008d3a8cd000001" is a character string representing the hex string 0x5a0008d3a8cd000001. Returned data lines vary in length.
- If the returnas value is "XMLString", then an XML-formatted string is returned. A <binaryline> tag represents each binary line of the report. Within the <binaryline></binaryline> tag is the <reportdataline> tag.

  The <reportdataline> tag is a displayable representation of the binary data line. The client program converts the data back to its binary format and then written to a file. For example, a data line of "5a0008d3a8cd000001" is a character string representing the hex string 0x5a0008d3a8cd000001. Returned data lines vary in length.
- If the returnas value is "XMLAttachment", it contains a file with the same data as returned by "XMLString".

# The Index List Web Service

The Index List Web Service request retrieves the following types of index information:

- Cross-report index names for a repository
- Report handles for sections of a report that match cross-report index values for a specified cross-report name
- The Index names for a specified report
- Report handle for a section of a report that matches index values for a specified index name
- Data that are returned as XML attachment or native formats

The Index List request is supported for CA View and CA Bundl. CA View also supports cross-report indexing. CA Dispatch does not provide indexing capabilities.

**Note:** The current release of Web Services does not support Logical View with the Index List Web Service request.

The following table describes request and response formats for the Index List Web Service:

| Request Name | Index List | Required | Case-sensitive |
|---|---|---|---|
| Request Format | userid:string | Y | Y |
| | password:string | N | Y |
| | repository: string with name of repository entry in repository definition file | Y | Y |
| | type: String with one of the following values:CROSSREPORTNAMES,CROSSREPORTVALUES, INDEXNAMES, INDEXVALUES. | Y | N |
| | returnas: (values can be: "vector", "stringarray", "xmlstring', "xmlattachment") | N (default "vector") | N |
| | | | N (field) |
| | | Y | Y (value) |
| | selectionCriteria: array of name-value pairs | | |

| Request Name | Index List | Required | Case-sensitive |
|---|---|---|---|
| Request<br>Response | Based on return type selected | | |

## Index List Web Service Input Parameters

**Userid**

Specifies the user ID associated with the request. Userid specifies a user ID defined to the Host Output Management product where the report is stored. The Host Output Management product is defined by the repository definition file entry for the repository that is specified as the repository parameter to the Index List request. The userid parameter is required for an Index List request. An Index List request only returns index information for reports that are assigned or accessible to the user ID in the Host Output Management product. If the user ID can view a single report, only information for that report gets retrieved, even if other parameters are specified on the Index List request.

**Password**

Specifies the password that is associated with the user ID for the request. The password matches the password that is associated with the user ID defined to the Host Output Management product. If no password is assigned, specify null.

**Repository**

Specifies the name of an entry in the repository definition file. The repository definition file is created during the post-installation of the Web Service or when the Host Output Management product is changed (such as applying a service pack, installing a new release of the product, or adding, changing or removing repositories for the product).

The repository definition entry identifies a specific repository in the Host Output Management Product. The Index List Web Service retrieves information for reports that are held on that repository only. If you require information for reports on other repositories, make secondary requests. The repository parameter is case-sensitive and must match the case of the repository entry exactly.

**Returnas**

Allows the client application to specify the format of the data returned. For example, a returnas value of "xmlattachment" returns the data from the Index List request as a file attachment in XML format. Returnas values that are supported for the Index List Web Service are:

- *Vector* data is returned as a vector. The client code supports vector data types if vector is specified as a returnas value. Java supports Vector data.
- *StringArray* data is returned as a string array. The client code supports the data that is returned as string arrays. Java and the Visual Basic client code support the string arrays.
- *XMLString* data is returned as a single string containing the data retrieved by the Web Service request. The string is a fully formatted XML document.
- *XMLAttachment* data is returned as a file attachment. The file is in XML format and contains the data that is retrieved by the Web Service request. The returnas parameter that is specified is not case-sensitive; it can be upper, lower, or mixed case.

The returnas parameter that is specified is not case-sensitive; it can be upper, lower, or mixed case. If the returnas parameter is omitted, its default value is set to "Vector".

**Type**

Specifies the type of index list being requested. Index List types include CROSSREPORTNAMES, CROSSREPORTVALUES, INDEXNAMES, and INDEXVALUES. The following table describes each Index List type:

**CROSSREPORTNAMES**

Request is to return cross-report index names for a repository.

**CROSSREPORTVALUES**

Request is to return the report handles for cross-report names matching cross-report values.

**INDEXNAMES**

Request is to return index names for a report in a repository.

**INDEXVALUES**

Request is to return the report handles for sections of a report with index names matching index values.

The type parameter is not case-sensitive; it can be upper, lower, or mixed case. This parameter is required.

**Selection Criteria**

Specifies a series of rules that are used when determining the reports (in the specified repository) for which index information is retrieved.

The selection criteria are name-value pairs, where the name is a field name (that uses a selected set of field name keywords) and the value is the value that must be matched (to select reports for which index information is to be returned). Values are specified as character strings; numeric data values, dates, and times are specified as numeric values such as "1", "01/01/01", "12:00:00:00".

For the selection criteria specification, the special rules are as follows:

1. If the field consists of characters, the value that is specified must be an exact match or a partial match when using an "*" or "?" character to create a mask character:

| Name | Value |
|---|---|
| RID | "creditcardsummary" |
| RID | "c*" |

"c*" returns information for all reports whose report names begin with *c*.

2. If the field is numeric, date or time, the value must be an exact match:

| Name | Value |
|---|---|
| EQ | "50"" |
| ARCDATE | "09/05/2003" |

Specifying "50" for the LINES criteria returns information for all reports that have a sequence number of 50. Specifying "09/05/2003" for the ARCDATE criteria retrieves information for reports that are archived on September 5, 2003.

3. If the field is numeric, date or time, the value can be in a range of values:

| Name | Value |
|---|---|
| LINES | "50,100" |
| RCDATE | "01/01/2003,09/05/2003" |

Specifying "50,100" for the LINES criteria returns information for all reports that have from 50 to 100 lines inclusive. Specifying "01/01/2003, 09/05/2003" for the ARCDATE criteria retrieves information for reports that are archived between January 1, 2003 and September 5, 2003.

4. CRXNAME and CRXVALUE are special criteria names relating to cross-report index names and value specification.

   - The CROSSREPORTNAMES type requires the CRXNAME selection criteria. The value that is specified selects cross-report index names for a repository. For example, if a CRXNAME with a value of A* is specified, all cross-report index names that begin with *A* are returned. The CRXNAME criteria name specifies one to eight parts of a cross-report index name. CA View supports an index name of up to eight parts.
   - The CROSSREPORTVALUES type requires CRXNAME and CRXVALUE selection criteria. CRXNAME specifies a cross-report name. The CRXVALUE criteria indicate the values to match for selecting sections of one or more reports matching the cross-report index and value. If the CRXNAME criteria specify a two-part index, then the CRXVALUE specifies two values to match. For example, to locate the reports for travel agency's names beginning with the letter *A* in the city of Dallas, specify a CRXNAME of "AGENCY", "CITY" and the CRXVALUE of "A*", "DALLAS".

     **Note:** Wildcard search cannot be used for both CRXNAME and CRXVALUE at the same time.

5. IDXNAME and IDXVALUE are special criteria names relating to index names and value specification.

   - If the type is INDEXNAMES then the selection criteria IDXNAME is required. The value that is specified is used to select index names for a report.

   For example, if an IDXNAME with a value of A* is specified, all of the index names for the specified report beginning with A are returned. The IDXNAME criteria name specifies one to eight parts of an index name; CA View supports an index name of up to eight parts.

   - The INDEXVALUES type requires the IDXNAME and IDXVALUE selection criteria. IDXNAME specifies an index name. The IDXVALUE criteria indicate the values to match for selecting sections of a report that match the index name and value. If the IDXNAME criteria specify a two-part index, then the IDXVALUE specifies two values to match.

     For example, to locate the portions of a report for names of travel agencies beginning with the letter A in the city of Dallas, specify IDXNAME of "AGENCY", "CITY" and the IDXVALUE of "A*", "DALLAS". A report handle selection criteria, also required, specifies the report to use for locating the index information.

     - The INDEXNAMES type requires the RPT_HANDLE selection criteria. RPT_HANDLE specifies a valid report handle for a report in the repository that the specified user ID can access.
     - The INDEXVALUES type requires the RPT_HANDLE selection criteria. RPT_HANDLE specifies a valid report handle for a report in the repository that the specified user ID can access.

6. If the type is CROSSREPORTNAMES or CROSSREPORTVALUES, the valid selection criteria fields are RID, GEN, VERSION, ARCDATE, CRXNAME, and CRXVALUE. Specifying any other selection criteria fields generates an error. Specification of the RID, GEN, VERSION, and the ARCDATE parameters is optional; if specified, they improve the performance.
7. Specify date values as *DD/MM/YYYY*.
8. Specify time values as *HH:MM:SS:nn* or *HH:MM:SS* which is the same value as *HH:MM:SS:00*.

# The Data Returned by the Index List Web Service

The data that is returned by the Index List Web Service depends on the type of request. For the Index List Web Service requests, the following are true:

- If the returnas value is "Vector", a vector entry for each matching entry is returned. The vector entry contains a string array with entries for returned information appropriate to the type of request.
- If the returnas value is "StringArray", a string array entry for each matching entry is returned. The string array entry contains another string array with an entry for returned information appropriate to the type of request.
- If the returnas value is "XMLString", entries for each matching entry are returned. The entry includes entries for returned information appropriate to the type of request.
- If the returnas value is "XMLAttachment", it contains a file with the same data that is returned by "XMLString".

## Specifying the CROSSREPORTNAMES Parameter

If the returnas value is "Vector" or "StringArray, a vector entry for each matching entry is returned. The vector entry contains a string array with entries for returned information appropriate to the type of request.

The following table describes the string array entries identifying the index names for each cross-report index:

| Entry | Description |
|---|---|
| Index Name Part 1 | Part 1 of the index name |
| Index Name Part 2 | Part 2 of the index name or "" if there is no part 2 |
| Index Name Part 3 | Part 3 of the index name or "" if there is no part 3 |
| Index Name Part 4 | Part 4 of the index name or "" if there is no part 4 |
| Index Name Part 5 | Part 5 of the index name or "" if there is no part 5 |
| Index Name Part 6 | Part 6 of the index name or "" if there is no part 6 |
| Index Name Part 7 | Part 7 of the index name or "" if there is no part 7 |

| Entry | Description |
|---|---|
| Index Name Part 8 | Part 8 of the index name or "" if there is no part 8 |
| Index Handle | Index handle to retrieve index values (interim value – do not use to access an index) |

If the returnas value is "XMLString" or "XMLAttachment", entries for each matching entry are returned. The entry includes entries with the index names for each cross-report index in the repository. Each entry is a <crossreportname> </crossreportname> pair. Within the pair are entries for each part of the index name. For each part, there is one entry and if there are any additional parts, they are present.

An entry with four parts resembles the following:

```
<crossreportname>
    <crossreportname1>INDEX1</crossreportname1>
    <crossreportname2>INDEX2</crossreportname2>
    <crossreportname3>INDEX3</crossreportname3>
    <crossreportname4>INDEX4</crossreportname4>

</crossreportname>
```

## Specifying the CROSSREPORTVALUES Parameter

If the returnas value is "Vector" or "StringArray", a vector entry for each matching entry is returned. The vector entry contains a string array with entries for returned information appropriate to the type of request. The string array has an entry for each report that is indexed by the cross-report index that is named in the request. The entry includes the matching values, the report name, the report generation number, the report sequence number, and a report handle. The report handle can be used to access the portion of the report that matches the values that are specified in the entry.

The following table describes the vector entries identifying the index names for each cross-report index:

| Entry | Description |
|---|---|
| Index Value 1 | Part 1 of the index value |
| Index Value 2 | Part 2 of the index value or "" if there is no part 2 |
| Index Value 3 | Part 3 of the index value or "" if there is no part 3 |

| Entry | Description |
|---|---|
| Index Value 4 | Part 4 of the index value or "" if there is no part 4 |
| Index Value 5 | Part 5 of the index value or "" if there is no part 5 |
| Index Value 6 | Part 6 of the index value or "" if there is no part 6 |
| Index Value 7 | Part 7 of the index value or "" if there is no part 7 |
| Index Value 8 | Part 8 of the index value or "" if there is no part 8 |
| Report ID | Name of the report |
| Gen | Generation number |
| Seq | Sequence number |
| Report Handle | Report handle that can be used to retrieve the report |

If the returnas value is "XMLString" or "XMLAttachment", entries for each matching entry are returned. The entry includes entries for returned information appropriate to the type of request. Each entry represents a report that is indexed by the cross-report index named. The entry is enclosed in <crossreportvalue> </crossreportvalue> tags. Within the tags are entries for each matching value, the report name, the report generation number, the report sequence number, and a report handle. An entry resembles the following:

```
< crossreportvalue>
    < crossreportvalue1>STATION000001</ crossreportvalue1>
    < crossreportvalue2> 000001</ crossreportvalue2>
    < crossreportvalue3>  0008</ crossreportvalue3>
    < crossreportvalue4> 000000008</ crossreportvalue4>
    <reportname>L2-VI-A</reportname>
    <generation>18</generation>
    <sequence>3</sequence>

<reporthandle>d3f260e5c960c1404040404040404040404040ffedfffc0c0a000300000001000

00000</reporthandle>

 </crossreportname>
```

## Specifying the INDEXNAME Parameter

If the returnas value is "Vector" or "StringArray", a vector entry for each matching entry is returned. The vector entry contains a string array with entries for returned information appropriate to the type of request. The string array has entries identifying the index names assigned to the specified report.

The following table describes the vector entries identifying the index names for each cross-report index:

| Entry | Description |
|---|---|
| Index Name Part 1 | Part 1 of the index name |
| Index Name Part 2 | Part 2 of the index name or "" if there is no part 2 |
| Index Name Part 3 | Part 3 of the index name or "" if there is no part 3 |
| Index Name Part 4 | Part 4 of the index name or "" if there is no part 4 |
| Index Name Part 5 | Part 5 of the index name or "" if there is no part 5 |
| Index Name Part 6 | Part 6 of the index name or "" if there is no part 6 |
| Index Name Part 7 | Part 7 of the index name or "" if there is no part 7 |
| Index Name Part 8 | Part 8 of the index name or "" if there is no part 8 |
| Index Handle | Index handle to retrieve index values interim value – do not use to access an index |

If the returnas value is "XMLString" or "XMLAttachment", entries for each matching entry are returned. The entry includes entries with the index names assigned to the specified report. Each entry is an <indexname> </indexname> pair. Entries for each part of the index name are in the pair. For each part, there is one entry and if there are any additional parts, they are present.

The following sample shows returned data:

```
<indexname>
    <indexname1>INDEX1</indexname1>
    <indexname2>INDEX2</indexname2>
    <indexname3>INDEX3</indexname3>
    <indexname4>INDEX4</indexname4>

</indexname>
```

## Specifying INDEXVALUEs

If the returnas value is "Vector" or "StringArray", a vector entry for each matching entry is returned. The vector entry contains a string array with entries for returned information appropriate to the type of request. The string array has an entry for each portion of the report matching the index value for the index name specified. The entry includes the matching values, the report name, the report generation number, the report sequence number, and a report handle. You can use the report handle to access the portion of the report that matches the values that are specified in the entry.

The vector entries identifying the index names for each cross-report index are as follows:

| Entry | Description |
|---|---|
| Index Value 1 | Part 1 of the index value |
| Index Value 2 | Part 2 of the index value or "" if there is no part 2 |
| Index Value 3 | Part 3 of the index value or "" if there is no part 3 |
| Index Value 4 | Part 4 of the index value or "" if there is no part 4 |
| Index Value 5 | Part 5 of the index value or "" if there is no part 5 |
| Index Value 6 | Part 6 of the index value or "" if there is no part 6 |
| Index Value 7 | Part 7 of the index value or "" if there is no part 7 |

| Entry | Description |
|---|---|
| Index Value 8 | Part 8 of the index value or "" if there is no part 8 |
| Report ID | Name of the report |
| Gen | Generation number |
| Seq | Sequence number |
| Report Handle | Report handle that can be used to retrieve the report |

If the returnas value is "XMLString" or "XMLAttachment", entries for each matching entry are returned. Each entry represents a section of a report that is indexed by the cross-report index name. The entry is enclosed in <indexvalue > </ indexvalue > tags. Within the tags are entries for each matching value, the report name, the report generation number, the report sequence number, and a report handle. An entry resembles the following:

```
<indexvalue>
    <indexvalue1>STATION000001</indexvalue1>
    <indexvalue2>000003</indexvalue2>
    <indexvalue3>0008</indexvalue3>
    <indexvalue4>000000028</indexvalue4>
    <reportname>L2-VI-A</reportname>
    <generation>18</generation>
    <sequence>3</sequence>

<reporthandle>d3f260e5c960c1404040404040404040404040ffedfffc0c0a000300000003000

00000</reporthandle>

  </indexvalue>
```

# The Ping Web Service

The Ping request provides the following functions:

- Tests connectivity to the Web Service.

  If a client application issues a Ping Web Service request and they receive the string "Started" back, the OM Web Services is executing and operational. Ping only returns the string Started and does not require repository definitions, Report Info Data dictionary entries or connectivity to any Host Output Management product.

- Processes the Web Service initialization code and allows all other Web Service requests to process without this additional overhead when a client application starts and makes its first Web Service request.

The following table describes the request and response format for the Ping Web Service:

| **Request Name** | **Ping Request** |
|---|---|
| Request Format | No input data |
| Request Response | "Started" |

# Chapter 4: Wrapper Classes

The Wrapper classes in both Visual Basic and Java are provided with the OM Web Services.

This section contains the following topics:



## Java Wrapper Classes

A set of Java wrapper classes are provided in the client.jar file that is installed by the CA OM Web Services installation; these wrapper classes invoke the Web services. The following wrapper classes are provided:

- OmReportSelectionItem
- OmReportList
- OmReportData
- OmIndexList
- OmPing

You can customize these classes by writing Java code to access the OM Web Services. All Java sample code uses these classes to invoke the OM Web Services.

The code for these wrapper classes can be found in the samples/java directory on the OM Web Services CD.

More jar files are copied and added to the class path for the applications are being developed and can be found in the caomws20.war in the WEB-INF\lib directory.

**Note:** In the case of accessing web services over HTTPS protocol, the optional protocol parameter in the wrapper classes is required. The Java runtime variables for Java SSL truststore need to be setup in your Java client application.

**For example:**

```
System.setProperty("javax.net.ssl.trustStore", "keypath/yourkeystore.jks");

System.setProperty("javax.net.ssl.trustStorePassword", "password");

System.setProperty("javax.net.ssl.trustStoreType","JKS");
```

# Report Selection Item Class

The Report Selection Item class, OmReportSelectionItem, holds selection criteria. An object of the type OmReportSelectionItem represents a single selection criteria that are specified by a name and a value or a range of values that are specified as a two-part string delimited by a comma.

The Report List and Index List Web Services support the optional specification of selection criteria. If selection criteria are specified and the web services are invoked using the provided Java wrappers, the wrappers expect the criteria that are specified as an array of one or more OmReportSelectionItem.

Programs that use the OmReportSelectionItem class must include the following import statement in Java code:

```
import com.ca.omgmt.client.OmReportSelectionItem;
```

## Methods

The following list describes the methods of the OmReportSelectionItem Class:

**constructor**

The default constructor for the class.

Parameter**:** none

**constructor**

The constructor for class that creates the object of type OmReportSelectionItem and initializes the name and the value.

Parameter:name: String value: String

**setName**

Sets the name for this selection criteria object.

Parameter**:**name: String

**getName**

Returns a string with the name specified for this selection criteria object.

Parameter**:**none

**setValue**

Sets the value for this selection criteria object

Parameter:value: String

**getValue**

Returns a string with the value specified for this selection criteria object.

Parameter:none

### Code Samples

The following code sample creates two selection criteria, one for the RID field and the other with a cross- report index name using the OmReportSelectionItem constructor, to set the name and values for each selection criteria:

```
OmReportSelectionItem[] byCriteria =   new OmReportSelectionItem[2];

byCriteria[0] = new OmReportSelectionItem("RID","C*");

byCriteria[1] = new OmReportSelectionItem("CRXNAME","ACCOUNT,MONTH");
```

The following code sample creates two selection criteria, one for LINES and the other for DOC_TYPE using the setName and setValue methods, to set the values for each selection criteria:

```
OmReportSelectionItem[] byCriteria =   new OmReportSelectionItem[2];

byCriteria[0].setName("LINES");

byCriteria[0].setValue("1,50");

byCriteria[1].setName("DOC_TYPE");

byCriteria[1].setValue("TEXT");
```

## Report List Class

The Report List class, OmReportList, makes calls to the Report List Web Service.

Programs that use the OmReportList class must include the following import statement in Java code:

```
import com.ca.omgmt.client.OmReportList;
```

Include the following import statement in Java code, if specifying the selection criteria:

```
import com.ca.omgmt.client.OmReportSelectionItem;
```

### Methods

The following list describes the methods of the OmReportList class:

**constructor**

The default constructor for the class.

Parameter:none

**constructor**

The constructor for the class that sets the default timeout for requests.

Parameter:timeout: int

**constructor**

The constructor for the class that sets the default timeout for requests, path for the caching directory and cache threshold.

Parameter:timeout: int, cacheDirectory: String, cacheThreshold: int

**invoke**

Invokes the Report List Web Service using the parameters that are passed as input.

Parameter**:**

userid: String

password:String

repository: String

returntype: String fields: String[]

criteria: OmReportSelectionItem[]

asAttachment: Boolean

port: String (default value: "8080")

servername: String (default value: "localhost")

protocol: String (default value: "http", other value: "https" for HTTPS protocol)

Last three parameters (port, servername, and protocol) are optional and can be omitted

### Code Samples

The following code sample retrieves a report list for the report name and archive date attributes for reports beginning with *c and* returns data in a Java vector (Attachment that is set to false):

```
OmReportList reportlist = new OmReportList(); String[] fieldlist = {"RID",
"ARCDATE"};
```

```
OmReportSelectionItem[] byCriteria =   new OmReportSelectionItem[1];

byCriteria[0] = new OmReportSelectionItem("RID","C*");

Object value;
value=reportlist.invoke("omuserid","ompwd","omrepository","vector",fieldlist,byCr
iteria,false,

"8080","omwsserver");
```

The following code sample retrieves a report list for the report name, number of lines and number of pages attributes for text reports with one to 50 lines inclusive and returns data in an XML file attachment (as Attachment set to true):

```
OmReportList reportlist = new OmReportList(); String[] fieldlist =  {"RID",
"LINES","PAGES"}; OmReportSelectionItem[] byCriteria =   new
OmReportSelectionItem[2]; byCriteria [0] = new
OmReportSelectionItem("LINES","1,50"); byCriteria [1] = new
OmReportSelectionItem("DOC_TYPE","TEXT");

Object value;
value=reportlist.invoke("omuserid","ompwd","omrepository","xmlattachment",fieldli
st,byCriteria,true,

"8080","omwsserver");
```

## Report Data Class

The Report Data class, OmReportData, makes calls to the Report Data Web Service.

Programs that use the OmReportData class must include the following import statement in Java code:

```
import com.ca.omgmt.client.OmReportData;
```

### Constants

The following list describes the constants of the OmReportData class:

**TEXT**

Report Type is text.

**AFP**

Report Type is AFP.

**AFP2PDF**

Report Type is AFP transformed to PDF.

**BINARY**

Report Type is Binary.

### Methods

The following list describes the methods of the OmReportData class:

**constructor**

The default constructor for the class.

Parameter:None

**constructor**

The constructor for the class that sets the default timeout for requests.

Parameter:timeout: int

**constructor**

The constructor for the class that sets the default timeout for requests, path for the caching directory and cache threshold.

Parameter:timeout: int, cacheDirectory: String, cacheThreshold: int

**invoke**

Invokes the Report Data Web Service using the parameters that are passed as input.

Parameter:

userid: String

password: String

repository: String

returntype: String

pagefrom: String

pageto: String

textreportformat: String

reporttype: String

reporthandle: String

asAttachment: Boolean

port: String (default value: "8080")

servername: String (default value: "localhost")

protocol: String (default value: "http", other value: "https"  for HTTPS protocol)

Last three parameters (port, servername, and protocol) are optional and can be omitted.

### Code Samples

The following code sample retrieves pages one through 5 of a text report in a String in XML syntax (as Attachment is set to false):

```
OmReportData reportdata = new OmReportData(); String reporthandle =
"c4e5d3d960c260c2c2c2404040404040404040fffaffee00000000000000000000000000";
Object value;
value=reportdata.invoke("omuserid","ompwd","omrepository","xmlstring","1","5","PA
GE",

OmReportData.TEXT,reporthandle,false,"8080","omwsserver");
```

The following code sample retrieves an AFP report as an attachment (as Attachment set to true):

```
OmReportData reportdata = new OmRAFPeportData(); String reporthandle =
"c8e3d3d960c260c2c2c2404040404040404040fffaffee00000000000000000000000000 ";
Object value;
value=reportdata.invoke("omuserid","ompwd","omrepository","xmlstring","1","5","PA
GE",

OmReportData.AFP,reporthandle,true,"8080","omwsserver");
```

The following code sample retrieves a transformed AFP report as a PDF file attachment (as Attachment set to true):

```
OmReportData reportdata = new OmRAFPeportData(); String reporthandle =
"c8e3d3d960c260c2c2c2404040404040404040fffaffee00000000000000000000000000 ";
```

DataHandler value:

```
value=(DataHandler)reportdata.invoke("omuserid","ompwd","omrepository","attachmen
t","","","",
OmReportData.AFP2PDF,reporthandle,true,"8080","omwsserver");
```

# Index List Class

The Index List class, OmIndexList, makes calls to the Index List Web Service.

Programs that use the OmIndexList class must include the following import statement in Java code:

```
import com.ca.omgmt.client.OmIndexList
```

Include the following import statement in Java code, if specifying the selection criteria:

```
import com.ca.omgmt.client.OmReportSelectionItem;
```

## Constants

The following table describes the constants of the OmIndexList class:

**CROSSREPORTNAMES**

Index List Request Type is CROSSREPORTINDEXNAMES.

**CROSSREPORTVALUES**

Index List Request Type is CROSSREPORTINDEXVALUES.

**INDEXNAMES**

Index List Request Type is INDEXNAMES.

**INDEXVALUES**

Index List Request Type is INDEXVALUES.

## Methods

The following table describes the methods of the OmIndexList class:

**constructor**

The default constructor for the class.

Parameters:None

**constructor**

The constructor for the class that sets the default timeout for requests.

Parameters**:**timeout: int

**constructor**

The constructor for the class that sets the default timeout for requests, path for the caching directory and cache threshold.

Parameter:timeout: int, cacheDirectory: String, cacheThreshold: int

**invoke**

Invokes the Index List Web Service using the parameters that are passed as input. The web service is accessed using port 8080 and the servername *localhost*.

Parameters:

userid: String

password:String

repository: String

returntype: String

requesttype: String

criteria: OmReportSelectionItem[]

asAttachment: Boolean

port: String (default value: "8080")

servername: String (default value: "localhost")

protocol: String (default value: "http", other value: "https" for HTTPS protocol)

Last three parameters (port, servername, and protocol) are optional and can be omitted.

### Code Samples

The following code sample retrieves cross-report names for a repository as a string array (as Attachment is set to false):

```
OmIndexList indexlist = new OmIndexList();

OmReportSelectionItem[] byCriteriavalue =   new OmReportSelectionItem[1];

byCriteria[0] = new OmReportSelectionItem("CRXNAME","*");

 Object value;

value= indexlist.invoke("omuserid","ompwd","omrepository","stringarray",
OmIndexList.CROSSREPORTNAMES,byCriteria,false,"8080","omwsserver");
```

The following code sample retrieves cross-report values for a cross-report index named "POLICY" as an attachment in XML format (as Attachment is set to true):

```
OmIndexList indexlist = new OmIndexList();

OmReportSelectionItem[] byCriteriavalue =   new OmReportSelectionItem[2];

byCriteria[0] = new OmReportSelectionItem("CRXNAME","POLICY");

byCriteria[1] = new OmReportSelectionItem("CRXVALUE","*");

Object value;

value= indexlist.invoke("omuserid","ompwd","omrepository","xmlattachment",
OmIndexList.CROSSREPORTVALUES,byCriteria,true,"8080","omwsserver");
```

## Ping Web Service Class

The Ping class, OmPing, calls the Ping Web Service.

Programs that use the OmPing class must include the following import statement in the Java code:

```
import com.ca.omgmt.client.OmPing;
```

### Methods

The following list describes the methods of the OmPing class:

**constructor**

The default constructor for the class

Parameters:None

**invoke**

Invokes the Ping Web Service using the parameters that are passed as input. The web service is accessed using port 8080 and the servername *localhost*.

Parameters:

port: String (default value: "8080")

servername: String (default value: "localhost")

protocol: String (default value:"http", other value: "https" for HTTPS protocol)

All parameters are optional and can be omitted (but port and servername together).

### Code Sample

The following code sample invokes the Ping Web Service:

```
OmPing ping = new OmPing ();

value= ping.invoke("8080","omwsserver");
```

# Visual Basic and C# Wrapper Client Proxy Classes

A set of Visual Basic (C#) classes is generated by Visual Studio and can be found in the sample project under project's Service References. Here in the OmWebService namespace can be found OmWebServiceServicePortTypeClient class with following OM Web Services methods:

- reportList
- reportData
- indexList
- ping

Customers writing code in Visual Basic or C# to access the OM Web Services can include the source code.

**Note:** The sample project can be found in the samples/dotNET/VB.NET directory on the OM Web Services CD. Read the README file before running the sample.

The wrapper classes are generated as WCF client proxy classes and require Microsoft Visual Studio 2010 or higher and .NET 3.5 or higher.

## Preparing Visual Studio Solution

If you decide to create .NET project from scratch alternatively, you have to start with generating a new Service Reference using Visual Studio Add Service. It requires a couple of modifications in the generated code also.

**Follow these steps:**

1. Select Add Service Reference in the Visual Studio and enter a valid address pointing to the WSDL file.
2. Complete the steps in the wizard.

   The generated reference can be found under the Service References node in the Visual Studio.
3. Open the App.config or Web.config file and add the following attributes for binding:

   ```
   messageEncoding="Mtom"

   maxReceivedMessageSize="<size in bytes>"

   maxBufferSize="<size in bytes>".

   Example:

   <binding name="OmWebServiceServicePortSoapBinding" messageEncoding="Mtom"
   maxReceivedMessageSize="2147483647" maxBufferSize="2147483647" />
   ```
4. Open the generated class Reference.cs, click the generated Service Reference and open the Object browser and then go to the OmVector class.
5. Replace the *private string[][] arrayField* with *private OmStringArray[] arrayField.*
6. Replace *public string[][] array* with *public OmStringArray[] array*.
7. Delete all the attributes earlier to public OmStringArray[] array property and insert the following attributes there:

   ```
   [System.Xml.Serialization.XmlElementAttribute(("array", Form =
   System.Xml.Schema.XmlSchemaForm.Unqualified, Order = 0)]
   ```

   **Note:** Follow the steps from 4 to 7 whenever the Service Reference is updated.

## Calling Web Services Operations

To call any Web Services 2.0 operation, it is necessary to create an instance of client class. The following examples show how it can be done in C# and VB.NET.

C#:

```
OmWebServiceServicePortTypeClient client = new OmWebServiceServicePortTypeClient();
```

VB.NET:

```
Dim client As New OmWebServiceServicePortTypeClient()
```

The available methods are listed in the following sections.

## OmWebServiceServicePortTypeClient

The OmWebServiceServicePortTypeClient class processes initialization and provides the interfaces that invoke the OM Web Services. Each code accessing web service call methods in this class.

An instance of OmWebServiceServicePortTypeClient can be created each time a web service is called by an application, however there is overhead involved in performing the initialization of the Microsoft WCF environment. We recommended creating this class once for an application and then use the class on all subsequent calls to the CA OM Web Services.

### Methods

The methods of the OmWebServiceServicePortTypeClient class are as follows:

**reportList**

Invokes the Report List Web Service

Parameter: OmReportSelectionParameter

**reportData**

Invokes the Report Data Web Service

Parameter: OmReportDataParameter

**indexList**

Invokes the Index List Web Service

Parameter: OmIndexListParameter

**ping**

Invokes the Ping Web Service

Parameter: None

All the parameter classes can be found in the common OmWebService namespace.

## Microsoft Office Excel

Microsoft Office Excel plugin is created as Visual Studio Tools for Office ( VSTO ) solution. For more information, see the Microsoft documentation *How to: Create Visual Studio Tools for Office Projects.*

The code accessing OM Web Services (WCF client proxy) looks alike as in case of VB/C# samples.

**Note:** The sample project can be found in the samples/dotNET/Excel folder on the OM Web Services CD. It requires Microsoft Visual Studio 2010 Professional SP1 and .NET 4.0 or higher.

**Follow these steps:**

1. Edit app.config - to set the endpoint address to host and port where Output Management Web Services server is running.
2. Right-click the project and click Publish in the Solution Explorer

   Installer is created.
3. Run the installer to add the Excel on the target computer.

   After you open the Excel,"CA Technologies"tab appears This tab has three buttons: Report List, Index List, and Report Data. Input cells are placed in the upper left corner of the sheet.

# Chapter 5: SOAP Requests

This chapter contains SOAP-encoded data that is captured by the TCP Monitor Utility provided with the OM Web Services.

This section contains the following topics:



## SOAP Coding Examples

The TCP Monitor Utility allows you to see the input SOAP request and output SOAP response. This can be useful in debugging applications that use the OM Web Services.

## Report List Web Service

This section contains the request and response format for the Report List Web Service.

### Request

The following sample shows the request format for the Report List Web Service:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:par="parameters.omgmt.ca.com">

    <soapenv:Header/>

    <soapenv:Body>

        <par:OmReportSelectionParameter>

          <userid>user999</userid>

          <repository>view999</repository>

          <returnas>vector</returnas>

          The following fields are available on this

            <!--1 or more repetitions:-->

            <item>RID</item>

            <item>DOC_TYPE</item>

            <item>SOURCE</item>

            <item>USERDATA</item>

            <item>RPT_DESC</item>

            <item>ARCDATE</item>

            <item>ARCTIME</item>

          </fields>

          <selectioncriteria>

            <!--1 or more repetitions:-->

            <omReportSelectionItem>

              <name>USERDATA</name>

              <value>*</value>

            </omReportSelectionItem>
```

```
            </selectioncriteria>
            <password>password999</password>
          </par:OmReportSelectionParameter>
      </soapenv:Body>
</soapenv:Envelope>
```

### Response

The following sample shows the response format for the Report List Web Service:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <ns1:OmResponse xmlns:ns1="parameters.omgmt.ca.com">
         <vector>
            <array>
               <item>AUTOMOBILE</item>
               <item>DVS</item>
               <item>VIEWRPTB</item>
               <item></item>
               <item></item>
               <item>7/19/01</item>
               <item>13:48:35</item>
            </array>
            <array>
               <item>BABYLON5JPG</item>
               <item>.JPG</item>
               <item>VIEWRPTB</item>
               <item></item>
               <item>FROM PC VIA  LPR/LPD</item>
               <item>4/23/02</item>
               <item>10:7:24</item>
```

```
                    </array>
                    <array>
                       <item>BOB</item>
                       <item>DVS</item>
                       <item>VIEWRPTB</item>
                       <item></item>
                       <item></item>
                       <item>2/23/01</item>
                       <item>13:10:55</item>
                    </array>
                 </vector>
              </ns1:OmResponse>
           </soapenv:Body>
        </soapenv:Envelope>
```

## Report Data Web Service

This section contains the request and response format for the Report Data Web Service.

### Request

The following sample shows the request format for the Report Data Web Service:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:par="parameters.omgmt.ca.com">

    <soapenv:Header/>

    <soapenv:Body>

        <par:OmReportDataParameter>


           <userid>user999</userid>

           <repository>view999</repository>

           <returnas>xmlattachment</returnas>

           <reporttype>TEXT</reporttype>

           <textreportformat>PAGE</textreportformat>

           <pagefrom>1</pagefrom>

           <pageto>1</pageto>

           <password>password999</password>



<reporthandle>c1c160e2c5d9c9c5e260d9c5d7d6d9e360f0f0f160c7c5d5c5d9c1d360d9d7e3fff
6fffe</reporthandle>


        </par:OmReportDataParameter>

    </soapenv:Body>

</soapenv:Envelope>
```

### Response

The following sample shows the response format for the Report Data Web Service:

```
------=_Part_1_80833.1056573874349
```

```
Content-Type: text/xml; charset=UTF-8

Content-Transfer-Encoding: binary

Content-Id: <C64395899563A9F88F0A3F21C63AFDAB>

<?xml  version="1.0" encoding="UTF-8"?>

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">

   <soapenv:Body>

      <ns1:OmResponse xmlns:ns1="parameters.omgmt.ca.com">

         <attachment href="cid:EE81942CE50536D4F92EA94FCE4AE667"
xmlns:ns2="OmWebServiceService"/>

      </ns1:OmResponse>

   </soapenv:Body>

</soapenv:Envelope>

------=_Part_1_80833.1056573874349

Content-Type: application/octet-stream

Content-Transfer-Encoding: binary

Content-Id: <EE81942CE50536D4F92EA94FCE4AE667>

<?xml  version="1.0" encoding="ISO-8859-1"?>

<report>

<page  number="1"  >

<line>)(VIEW=REPORT-01  Page 0001   PLine 0001   100  x  50  x  80  = 400,000
Report-01</line>

<line>Report-01   Page 0001  PLine 0002  100 x 50 x 80 = 400,000 Report-01</line>

<line>Report-01   Page 0001  PLine 0003  100 x 50 x 80 = 400,000 Report-01</line>

<line>Report-01   Page 0001  PLine 0004  100 x 50 x 80 = 400,000 Report-01</line>

<line>Report-01   Page 0001  PLine 0005  100 x 50 x 80 = 400,000 Report-01</line>

<line>Report-01   Page 0001  PLine 0006  100 x 50 x 80 = 400,000 Report-01</line>

<line>Report-01   Page 0001  PLine 0007  100 x 50 x 80 = 400,000 Report-01</line>

<line>Report-01   Page 0001  PLine 0008  100 x 50 x 80 = 400,000 Report-01</line>

<line>Report-01   Page 0001  PLine 0009  100 x 50 x 80 = 400,000 Report-01</line>
```

```
<line>Report-01   Page 0001  PLine 0010  100 x 50 x 80 = 400,000 Report-01</line>
<line>Report-01   Page 0001  PLine 0011  100 x 50 x 80 = 400,000 Report-01</line>
<line>Report-01   Page 0001  PLine 0012  100 x 50 x 80 = 400,000 Report-01</line>
<line>Report-01   Page 0001  PLine 0013  100 x 50 x 80 = 400,000 Report-01</line>
<line>Report-01   Page 0001  PLine 0014  100 x 50 x 80 = 400,000 Report-01</line>
<line>Report-01   Page 0001  PLine 0015  100 x 50 x 80 = 400,000 Report-01</line>
<line>Report-01   Page 0001  PLine 0016  100 x 50 x 80 = 400,000 Report-01</line>
<line>Report-01   Page 0001  PLine 0017  100 x 50 x 80 = 400,000 Report-01</line>
<line>Report-01   Page 0001  PLine 0018  100 x 50 x 80 = 400,000 Report-01</line>
</page>
</report>
-----=_Part_1_80833.1056573874349--
```

## Index List Web Service

This section contains the request and response format for the Index List Web Service.

### Request

The following sample shows the request format for the Index List Web Service:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:par="parameters.omgmt.ca.com">

    <soapenv:Header/>

    <soapenv:Body>

        <par:OmIndexListParameter>

          <indextype>CROSSREPORTNAMES</indextype>

          <password>password999</password>

          <repository>view999</repository>

          <returnas>stringarray</returnas>

          <selectioncriteria>

            <!--1 or more repetitions:-->

            <omReportSelectionItem>

              <name>RID</name>

              <value>A*</value>

            </omReportSelectionItem>

          </selectioncriteria>

          <userid>user999</userid>

        </par:OmIndexListParameter>

    </soapenv:Body>

</soapenv:Envelope>
```

### Response

The following sample shows the response format for the Index List Web Service:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">

   <soapenv:Body>
```

```
      <ns1:OmResponse xmlns:ns1="parameters.omgmt.ca.com">
         <vector>
            <array>
               <item>AFPPAGE</item>
               <item></item>
               <item></item>
               <item></item>
               <item></item>
               <item></item>
               <item></item>
               <item></item>

<item>ff120fff38120ffea0000000000000000000000000000000000000000000000000000000000000000</i
tem>

               <item>AGENT</item>
               <item></item>
               <item></item>
               <item></item>
               <item></item>
               <item></item>
               <item></item>
               <item></item>

<item>ff12158dd0120ffea0000000000000000000000000000000000000000000000000000000000000000</i
tem>

            </array>
         </vector>
      </ns1:OmResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

# Chapter 6: Customizing the Report List Web

This chapter contains attributes available for the customization in the Report List Web Service.

This section contains the following topics:

Customizing Attributes (see page 75)

## Customizing Attributes

Attributes can be customized for the Report List Web Service.

**Note:** All repositories cannot use all the fields.

The following table describes attributes available for the customization in the Report List Web Service:

| Attribute | Name | Data Type | Description |
|---|---|---|---|
| ARCDATE | Archive Date (creation date) | Date, ten characters, in the format *mm/dd/yyyy* | The date of the report. Depending on the remote repository, this date may represent the date the report ran (the date of the MVS job) or the date the report was collected by the repository. |
| ARCTIME | Archive Time (creation time) | Time, 11 characters, in the format *hh:mm:ss:nn* | The time of the report. Depending on the remote repository, this time may represent the time the report ran (the time of the MVS job) or the time the report was collected by the repository. **Note:** The time format *hh:mm:ss:nn* is required for CA Bundl. |

| | | | |
|---|---|---|---|
| CHECKSUM | Resource Checksum | Up to eight characters | The checksum of the resource associate with a multiple-part AFP report.<br>**Note:** This attribute has no meaning for other report types; the value will be null.<br>The CHECKSUM attribute, in conjunction with RESRC_SIZE, can be used to uniquely identify AFP resources. This allows for caching of resources.<br>Different reports can use the same resources; using a cached resource may improve report retrieval performance. |
| DOC_TYPE | Document Type | Up to eight characters | The type of report.<br>Values include:<br>**AFP**<br>An AFP/ACIF documents. Retrieves data and resource (if any) in binary mode.<br>**TEXT**<br>A text document. Retrieves data in text mode.<br>**.XXX**<br>*XXX* represents a PC-style file extension. This DOC_TYPE value is the actual PC extension; use this value to determine the application to display the document. Retrieves data in binary mode. |
| GEN | Report Generation | Numeric | The backup generation for the report for CA View.<br>For CA Bundl, the GEN attribute is a flag that indicates whether the report is indexed; GEN=0 indicates the report is not indexed and GEN=1 indicates the report is indexed. |
| JOBID | Job Identifier | One to eight characters | The ID of the job that created the report. |
| JOBNAME | Job Name | One to eight characters | The name of the job that created the report. |

| | | | |
|---|---|---|---|
| LINES | Lines in Report | Numeric | The total number of lines in the report.<br>For DOC_TYPE=.XX, LINES is the number of bytes in the report. |
| LPAGES | Pages in Report | Numeric | The total number of pages in the report. |
| ODEST | Output Destination | Up to 17 characters | The original report destination from the job that produced the report.<br>For CA Bundl, ODEST specifies the report's mailcode. |
| ONDISK | On Disk | One character, Y or N | An indicator flag for report location; indicates if the report has a copy on disk.<br>ONDISK=Y<br>The report is on disk and ready for immediate access.<br>ONDISK=N<br>The report is not on disk and is offline.<br>Used in conjunction with ONOPTICAL and ONTAPE. |
| ONOPTICAL | On Optical | One character, Y or N | An indicator flag for the report location; indicates if the report has a copy on an optical device.<br>ONOPTICAL=Y<br>The report is on optical.<br>ONOPTICAL=N<br>The report is not on optical. |
| ONTAPE | On Tape | One character, Y or N | An indicator flag for the report location; indicates if the report has a copy on a tape or another long-term back-up media.<br>ONTAPE=Y<br>The report is on tape.<br>ONTAPE=N<br>The report is not on tape. |

| | | | |
|---|---|---|---|
| RESRC_SIZE | Resource Size | Numeric | The size, in bytes, of the resource file associated with a multiple part AFP report.<br>**Note:** This attribute has no meaning for other report types; the value will be blank or null.<br>The RESRC_SIZE attribute, in conjunction with CHECKSUM, can be used for uniquely identifying AFP resources to allow resource caching.<br>Different reports can use the same resources; using a cached resource may improve report retrieval performance. |
| RID | Report Identifier | 32 characters for CA View. 20 characters for CA Bundl or CA Dispatch | The name of the report.<br>For CA View version 12.0 and up, RID can contain some special characters (refer to the CA View 12.0 Release Notes for the full list).<br>For CA Bundl, RID contains the CA Bundl application ID and the report name. The first 10 bytes of the RID is the application ID (blank padded); the second 10 bytes of the RID is the report name. |
| RPT_DESC | ReportDescription | Up to 24 characters | The description of the report.<br>For CA View, RPT_DESC contains the CA Deliver description if you are in EXP or EXPO mode; otherwise, RPT_DESC is blank. |
| RPT_HANDLE | Report Handle | Binary | A unique identifier for a report in a particular repository.<br>RPT_HANDLE is the primary key for accessing the report. |
| SEQ | Report Sequence | Numeric | The position of the report in its back-up generation.<br>**Note:** Applies to CA View only. |

| | | | |
|---|---|---|---|
| USERDATA | User Data | Length varies by Host Output Management product | For CA Bundl, USERDATA contains the mailcode and is used to filter the ReportList API request by the CA Bundl mailcode.<br>For CA Dispatch, USERDATA contains the recipient ID and is used to filter the ReportList API request by the CA Dispatch recipient.<br>**Note:** CA Dispatch does not use the ODEST field to return the recipient; CA Bundl does.<br>For CA View, USERDATA contains accounting information from the job that created the report if available. |
| VERSION | Version or Copy of the Report | Numeric | Identifies the version or copy of a particular report based on the report name (RID).<br>**Note:** VERSION applies to CA view only.<br>0 represents the most recent version of a report.1 is the next most recent. |
| XCODE | Exception Code | Up to six characters | The exception code for the job that printed the report (JCL errors, abends, and so forth). |

# Index