SERENA®

ChangeMan® ZMF 8.1

XML Services User's Guide

Publication date: September 2015

Contents

About This Book

The XML Services User's Guide documents the most commonly used features of the XML Services application programming interface (API) to Serena® ChangeMan® ZMF. Tutorials, code examples, use cases, and tips and techniques for applications supplement detailed data structure tables covering 162 functions available for general customer use.

Services used with the Enterprise Release Option (ERO) are not described in this manual but are listed for reference in *ERO XML Services Summary* in *Chapter 2*, *"XML Syntax Basics"*. Refer to the *ChangeMan ZMF ERO XML Services User's Guide* for information on using these services.

After reading this manual, you should be able to do the following:

- Understand the software architecture that underlies ChangeMan ZMF XML Services.
- Create a well-formed XML document that complies with Serena XML syntax.
- Use the Serena XML markup language to build reusable XML documents that invoke functions and retrieve data from ChangeMan ZMF.
- Use the XML batch execution client to issue Serena XML service requests to ChangeMan ZMF and receive Serena XML replies.
- Experiment with the XMLSERV interactive prototyping tool to learn Serena XML syntax, generate prototype request messages, and browse Serena XML replies.

Software Versions

This manual discusses Serena Software's XML Services as implemented in ChangeMan ZMF version 8.1 (GA) and ChangeMan ZDD 8.1 (GA).

Audience

This manual targets experienced ChangeMan ZMF programmers, multi-platform systems integrators, and ChangeMan ZMF administrators.

You should be familiar with your mainframe operating system and security system, and you should understand the operation and administration of ChangeMan ZMF. Some familiarity with basic XML syntax and schemas is helpful. Familiarity with PCs is assumed.

Scope

The XML Services features described in this manual are limited to services and functions available for general customer use. These are sometimes called the “Green” services. “Green” functions address package and component lifecycle management, complex searches and queries, data set management functions, change library management functions, and detailed information retrieval from the ChangeMan ZMF database.

Additional services and functions exist to support advanced systems integration needs. The latter features are known as the “Yellow” services because they pose some risk of database corruption and should be used with caution. These are documented in quick-reference form for customers who attend advanced training in XML Services. This information is available from Serena Customer Support.

Related Topics

You need not become an XML expert to use XML Services. To master its advanced capabilities, however, sound knowledge of XML standards is advised. The authoritative source for this information is the World Wide Web Consortium (W3C). You can find the latest XML specifications on the Web at *http://www.w3c.org*.

The eXtensible Markup Language (XML) standard consists of many components in various stages of development, change, and ratification. Of these, you should become familiar with the core XML specifications that cover XML syntax and schemas. If you want to manipulate and reformat the XML output from XML Services (e.g., for custom reports), you should also study the XML stylesheet (XSL) specifications.

Related Documents

Title	Description
Serena® ChangeMan® ZMF ERO XML Services User’s Guide	Documents the most commonly used ERO features of the XML Services application programming interface to ChangeMan ZMF.
Serena® ChangeMan® ZMF XML Reference Tables	HTML cross-reference tables for “green” and “yellow” service/scope/message combinations supported by XML Services, including ERO, and the XML tags for each. If you have taken Serena’s advanced training course in XML Services, you can contact Customer Support for access to this guide.
Serena® ChangeMan® ZMF Installation Guide	Step-by-step instructions for the initial installation of ChangeMan ZMF. Includes installation instructions for XML Services working data areas.

Title	Description
Serena® ChangeMan® ZMF Administrator's Guide	Includes information on customizing exits to call XML Services.
Serena® ChangeMan® ZMF Web Services Getting Started Guide	Documents the Web Services application programming interface to ChangeMan ZMF

Typographical Conventions

The following textual conventions are used throughout this document to highlight special information:

This convention . . .	Represents . . .
`Monospace`	Serena XML code or keyword.
`Bold Monospace`	Serena XML required tag.
`< >`	Delimiters for XML tag name (e.g., `<package>`).
. . .	Tags omitted from example for clarity.
Italic	URL, file name, function name, or book title.
Blue Italic	Clickable cross-reference or active hyperlink in document.

Manual Organization

This chapter . . .	Contains this information . . .
1	**Introduction and architecture overview.** Introduction to features, functions, and benefits of XML Services. Layered software architecture, dynamic client/server messaging, XML interface language, and modular service objects. Choice of XML, COBOL copybook, or REXX batch execution clients.
2	**Serena XML basics.** XML language extensions and XML schemas. Syntax and structure of a well-formed XML document. High-level structure and syntax of Serena XML message documents. Table of Serena XML service, scope, and message names with corresponding COBOL copybooks.

This chapter . . .	Contains this information . . .
3	**Package management.** Serena XML syntax, data structures and values, code examples, and usage tips for the following package-related tasks: • Package lifecycle functions (e.g., *create, delete, freeze, submit, approve, promote, demote, back out, revert*). • Package-level component change (e.g., *unfreeze, refreeze, list*). • Package control and metadata information management (e.g., *list*).
4	**Component management.** Serena XML syntax, data structures and values, code examples, and usage tips for the following component tasks: • Component lifecycle functions (e.g., *checkout, checkin, browse, compare, build, recompile, relink, scratch, rename, lock, unlock*). • Component staging versions (e.g., *list, retrieve*). • Component control and metadata information management (e.g., *list*). • Component history information (e.g., selective *search* and *list*).
5	**Search, summary, and analysis tasks.** Information retrieval and statistical analysis that crosses package, component, and/or application boundaries. Includes the following: • Multi-package search (e.g., general and limbo *search*). • Multi-package *summary* statistics. • Component impact analysis functions. • Change *log* creation and listing.
6	**Dataset management.** XML Services support for managing sequential and partitioned datasets on the mainframe. Includes PDS/PDSE lifecycle functions (e.g., *create* and *delete* data set, *delete* data set member, and *list* data set information).
7	**Hierarchical file system services.** XML Services support for managing HFS files and directories on the mainframe. Includes: • HFS directory services (e.g., *create*, *delete*, *rename*, or *list* the contents of a directory). • HFS file lifecycle services (e.g. *create*, *delete*, *rename*, or *copy* an HFS file, change certain file attributes, or test for file existence and verify user access permissions). • File conversion services (e.g., *import* a z/OS PDS (Partitioned Data Set) member as an HFS file or *export* an HFS file as a PDS member).

This chapter . . .	Contains this information . . .
8	**Database management for IMS and DB2.** Serena XML syntax and data structures for retrieval of change control metadata about the following: • IMS package-level, application-level, and global settings and data binding information (e.g. control region, ACB build statement, DBD and PSB control statement *list*.) • DB2 application-level and global settings and data binding information (e.g., *list* records for active DB2 applications, logical files, and physical files).
9	**Online forms management.** Serena XML syntax and data structures for retrieving information and submitting and approving custom online forms associated with a package.
10	**ChangeMan ZMF administration tasks.** Serena XML syntax and data structures for retrieving global and application-level information about change libraries, sites, languages, library types, and build procedures. XML access to site calendars and package installer scheduling facilities, approver maintenance, reason code administration, and notifications are also discussed.
11	**System administration tasks.** Serena XML syntax and data structures for retrieving SERNET and ChangeMan ZMF setup information, environment parameters, and started task library concatenation.
Appendix A	**XMLSERV -** Interactive TSO/ISPF prototyping tool for XML Services.
Appendix B	**SERXMLBC** – Serena XML native-XML batch execution client.
Appendix C	**SERXMLAC** – Serena XML ASSEMBLER execution client.
Appendix D	**SERXMLCC** – Serena XML COBOL execution client.
Appendix E	**SERXMLRC** – Serena XML REXX execution client.
Appendix F	**Problem analysis and troubleshooting tools.** How to resolve errors when using XML Services.

Change bars in the left margin identify changes in this publication since it was published on March 21, 2012.

XML Services Concepts and Architecture 1

XML Services offers ChangeMan® ZMF customers and system integrators an enhanced application programming interface (API) based on industry-standard XML (e**X**tensible **M**arkup **L**anguage). XML Services simplifies customization, data interchange, and cross-product interoperability for ChangeMan ZMF and other products. An integrated feature of the base ChangeMan ZMF product, XML Services supports all optional product features, including the DB2 Option, IMS Option, ERO Option, M+R Option, and Load Balancing Option. XML Services is the preferred API for customers and system integrators who work with ChangeMan ZMF.

Functionally, XML Services:

- Offers a unified XML programming interface to ChangeMan ZMF functions.
- Provides open access to ChangeMan ZMF package master, component master, Impact/Analysis repository, and activity log data.
- Interoperates seamlessly with Serena products such as ChangeMan® ZDD and StarTool® DA.
- Enables integration with third-party development tools, databases, and reporting.
- Includes "software developer kit" (SDK) environments to simplify developer access to the XML Services API using native-XML, ASSEMBLER, COBOL, or REXX.

Software Architecture

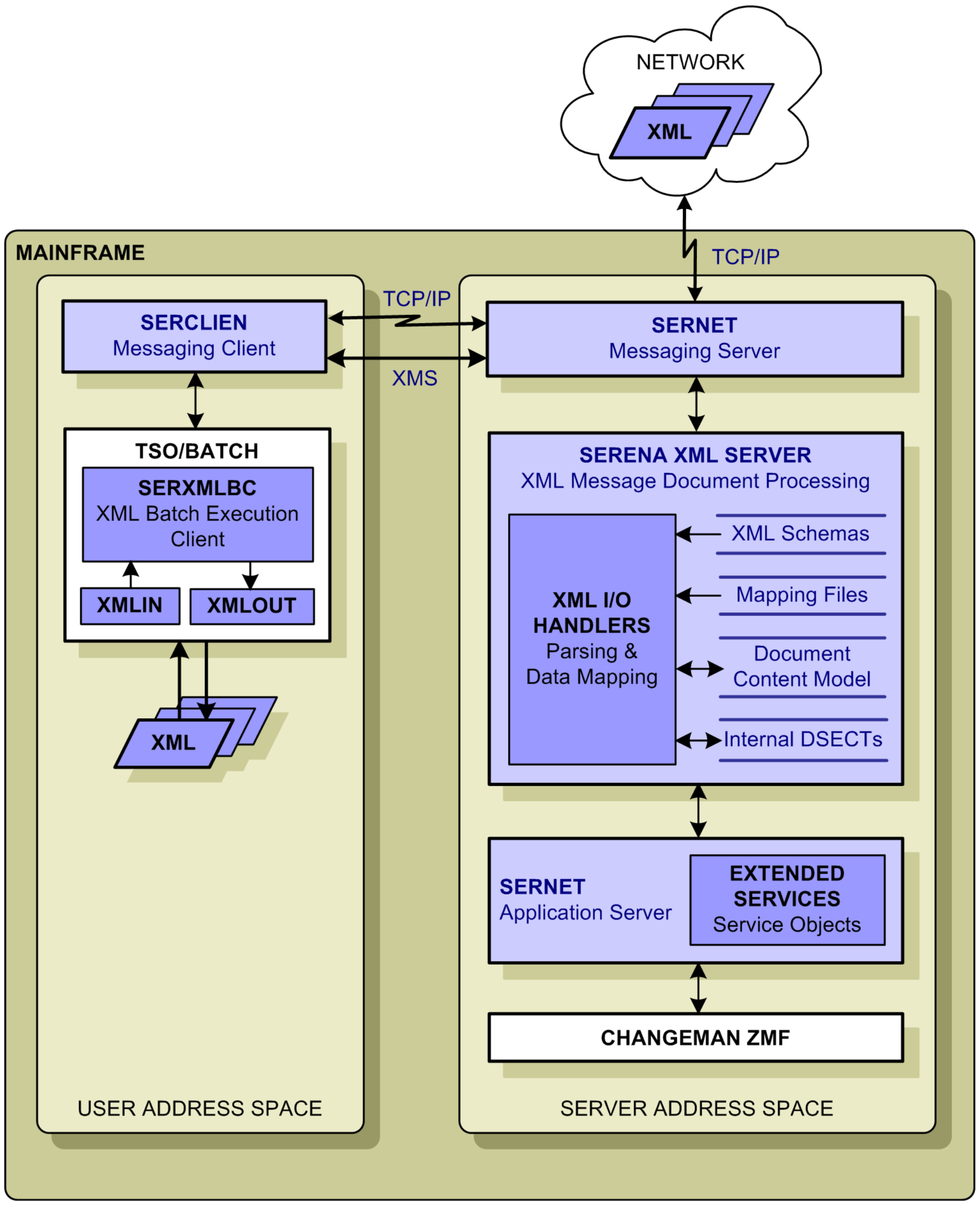
XML Services comprises much more than syntax. It is fully integrated with ChangeMan ZMF and builds on the following architectural keystones:

- ***A layered software architecture*** provides application independence from technology changes in ChangeMan ZMF internals. The low-level "Extended Services" that perform basic ChangeMan ZMF functions are isolated from higher-level interfaces.
- ***Modular service objects*** within the "Extended Services" layer provide a single point of access to ChangeMan ZMF functions. The set of low-level service objects is both comprehensive and extensible.
- ***Dynamic client/server messaging*** uses a shared object-request broker for all ChangeMan ZMF communications. This approach supports asynchronous, stateless, message-based transactions between XML client and server — ideal for network environments and Web-enabled services.

- ***A tag-based XML markup language*** built on industry-standard XML is easily recognized and processed by third-party software. Tag-based markup frees data interchange from bit-offset dependencies and wireline sequence dependencies. It is also inherently extensible, so that custom programs that use the XML Services interface need not be changed when new features or functions are added to ChangeMan ZMF.
- ***Developer-friendly SDK clients*** support COBOL-to-XML, ASSEMBLER-to-XML, and REXX-to-XML API calls as well as interactive XML prototyping.

An overview of the layered XML Services architecture appears in *Exhibit 1-1*.

Exhibit 1-1. XML Services Architecture



Message Processing Cycle

The architectural building blocks of XML Services come together in the Serena XML message processing cycle. The message processing cycle flows through the following steps:

- Serena XML request message issued from client to server
- XML message parsed
- XML data mapped to internal ChangeMan ZMF data formats
- Requested task performed by low-level service object
- Service object success/failure codes and output data mapped to XML data elements
- Serena XML reply message sent by server to client

Every Serena XML request message that reaches the server triggers a Serena XML reply. At minimum, the reply includes a result code that informs the requesting program whether the requested task succeeded, generated a warning message, or failed. Successful requests may trigger several result messages as well — each result representing, for example, a record in a data set or a line in a report. All results generated by a single XML request document are returned in a single XML reply document.

Submitting a Serena XML Request

Serena XML service request messages are issued from the client to the server via a software developer’s “kit” (SDK) or environment optimized for a particular programming language. Batch XML is submitted via the SERXMLBC batch client. Interactive XML can be prototyped in XMLSERV with prompts for required tags and other ease-of-use features, then submitted for execution through SERXMLBC.

The SERXMLCC COBOL-to-XML batch execution client, together with a collection of COBOL copybooks, facilitates XML Services API requests using native COBOL data formats and program calls. Each copybook wraps the proper Serena XML syntax around the contents of predefined COBOL variables populated by your custom COBOL program. Your COBOL program then calls SERXMLCC to generate a true Serena XML request document and place it in the normal XML message processing stream.

The SERXMLRC REXX-to-XML batch execution client similarly facilitates XML Services API requests using native REXX stem data formats and program calls. Your REXX program populates an approximate REXX stem structure, then calls SERXMLRC to generate a Serena XML request document and place it in the normal XML message processing stream.

The SERXMLAC ASSEMBLER-to-XML batch execution client facilitates XML Services API requests using native ASSEMBLER data formats and program calls.

Service, Scope, and Message Syntax

Every Serena XML service request uses a high-level XML syntax that identifies the ChangeMan ZMF *service*, *scope*, and *message* names for the task requested. These values, in combination, uniquely identify the modular service object on the server that must process the request. They also identify the function to be performed and the category of information to perform it against. Their values also must be specified with CAPITAL letters. The batch execution client that submits your request first preprocesses it to ensure that the combination of service, scope, and message names is valid.

Message Routing

If the XML Services service, scope, and message names are valid, the execution client calls the appropriate client messaging program — either SERCLIEN on the mainframe or SERNET Connect on distributed platforms — to initiate a connection to ChangeMan ZMF. The preferred communications protocol for this connection is TCP/IP, but cross-memory services (XMS) is also supported if the client and server both reside on the same mainframe LPAR. The messaging client performs any necessary data compression and packages the XML message with appropriate headers for network addressing, handshaking, and mainframe logon. It then requests a communications session to ChangeMan ZMF via the SERNET messaging server.

The SERNET messaging server resides on the host in the ChangeMan ZMF server address space, where it listens on one or more communication ports for incoming messages. When a message arrives, SERNET completes any network handshaking needed, processes the communications headers, and establishes a conversation. SERNET also decompresses messages and performs any needed data format conversions (e.g. from ASCII to EBCDIC).

If the inbound message contains Serena XML, the SERNET messaging server calls the XML Services input handler to transform that data into internally readable form. The XML input handler then returns the transformed data to the SERNET messaging server, which routes it to the appropriate low-level service object for action.

XML Parsing and Data Mapping

At the core of XML Services are its XML parsing and bidirectional data mapping processes. These interpret Serena XML message streams and map the identified XML data structures of a request to the internal assembler DSECT formats used by the low-level service objects in ChangeMan ZMF. In the reverse direction, the low-level service objects return results that are mapped from their internal assembler DSECT formats to Serena XML data elements, then marshalled into Serena XML reply messages. Serena uses proprietary parsing to achieve faster XML processing.

Generating the Serena XML Reply

After the XML input handler has parsed the Serena XML request message and mapped its data to an appropriate DSECT structure, SERNET queues that DSECT request block for input to the requested low-level service object. The service object receives the request block, performs the requested task, and generates (at minimum) a numeric return code. It may also generate an output message, a report listing, or a set of search results. This output data is stored in one or more output DSECTs populated by the low-level service object. The output is then returned to SERNET for routing to the XML output handler.

The XML output handler marshals a Serena XML reply document from one or more of these output DSECTs. Guided by the permanent object mapping table, the XML output handler maps each field in the DSECT to its corresponding XML tag and creates a document content model for the reply document in a temporary hashed tag pool. The output handler then transforms the document content model into well-formed XML and places the resulting document in a user response area known to the SERNET messaging server.

Control then returns to SERNET, which compresses the XML reply message, packages it with appropriate communications headers, and routes it to the requesting client. Note that, for distributed clients, the SERNET messaging server echoes the original XML request in the XML reply document. For ChangeMan ZMF clients, however, the original XML request is not echoed.

ChangeMan ZMF Interface Comparison

ChangeMan ZMF supports following interfaces:

- Interactive ISPF end-user and administrator panels
- Interactive and batch-mode programming clients (SDKs) for XML Services — including SERXMLAC, SERXMLBC, SERXMLCC, SERXMLRC, and XMLSERV

Of these, the interactive ISPF interface is functionally comprehensive. User tasks are presented at a high level; many low-level software functions might take place behind the scenes to accomplish a “simple” high-level ISPF request. The ISPF interface also builds in robust data validation features on every panel. No other interface provides this level of data validation support.

No one-to-one mapping exists between XML Services interface functions and ISPF interface functions, although similarities are apparent. The XML Services interface targets a lower level of internal function than does ISPF, and is more directly shaped by underlying database implementations and service object technology. Consequently, ISPF-based intuitions may not always apply to XML Services. In addition, XML Services includes no built-in data validation.

Caution

Data validation is the responsibility of XML Services customers. XML Services provides no built-in data validation. All ISPF tables that are available to the ISPF interface to ChangeMan ZMF are not necessarily available to the corresponding functions that are performed with the Serena XML Services. Furthermore, the target XML Services do not need these tables to perform their functions correctly. Using the XMLWARN facility can provide further information concerning data validation, as documented in *“Warn - XML Tag Name Warning” on page 603*.

ChangeMan ZMF interface differences are summarized in *Exhibit 1-2*.

Exhibit 1-2. ChangeMan ZMF Interface Comparison

Interface	Interactive	Reusable Batch Jobs	Functional Coverage	Data Validation
ISPF	Yes	No	Complete, high-level	Yes
XML Services batch clients (SERXMLAC, SERXMLBC, SERXMLCC, SERXMLRC)	No	Yes	XML, COBOL, REXX, Assembler	No
XML Services interactive client (XMLSERV)	Yes	Yes	XML	No

XML Syntax Basics 2

Serena XML is SERENA Software's markup language for Enterprise Change Management (ECM). It is standard XML extended to support the customization, data interchange, and interoperability needs of ChangeMan ZMF customers as they implement change management solutions. Serena XML is the most visible component of XML Services.

The Serena XML markup vocabulary consists of more than a thousand special-purpose XML *tags* used to delimit values in a text file. These tags are defined according to XML's rules for adding new tags to itself. The particular mechanism for defining these special-purpose tags is called an *XML schema*. The Serena XML schemas define not only the tag vocabulary of Serena XML, but also the structure of each data element named by these tags and the syntax used when populating these data elements in an XML document.

Is Serena XML "really" XML, then? The answer is, emphatically, ***yes***. XML stands for *e**X**tensible **M**arkup **L**anguage*. Its reason for being is to provide a standard method for creating special-purpose markup languages — *extensions*, that is, to the base XML tag set. There are two points to remember about XML extensions:

- ***Extensions are not replacements; they are additions.*** XML imposes a discipline on its language extensions that makes them systematically extensible over time. Within broad limits, this discipline prevents the foreclosure of alternatives; future options remain open. Built-in XML extensibility means that Serena XML can grow and change without forcing obsolescence on earlier versions of the language.
- ***Extensions to XML are syntactically consistent with XML.*** All special-purpose extensions to XML follow the same basic syntactic and structural rules. Familiarity with basic XML syntax makes all XML-based markup languages easier to learn and use.

Some knowledge of Serena XML syntax is needed by all users of XML Services. For example, COBOL programmers working with the COBOL-to-XML copybook interface need to know about individual copybook functions and predefined COBOL variable names, data types, and value information — all of which derive from Serena XML. Programmers who work directly with Serena XML need not only data type and value information, but also detailed information about XML language syntax and data structures.

This chapter begins with a discussion of general XML syntax and standards as defined by the World Wide Web Consortium (W3C). It then addresses the basic features of Serena XML. The features discussed are those that apply to all message documents created in Serena XML and to all ChangeMan ZMF user tasks performed via Serena XML. The chapter concludes with a summary of all valid combinations of `<service>`, `<scope>`, and

`<message>` `name` attributes in Serena XML available to customers for general use. This summary includes the names of the corresponding COBOL-to-XML copybooks.

XML Syntax Standards

The body of standards defining XML is actually quite large, but only two core specifications directly concern users of Serena XML. These are the XML Version 1.0 syntax specification and the XML Schema specification. These and other XML specifications are established by the World Wide Web Consortium (W3C) and are published online at *http://www.w3c.org*.

To use the Serena XML programming interface to XML Services, you first need a basic familiarity with this core XML syntax.

XML Tag Names

Programmers familiar with Web markup will note that XML syntax resembles HTML syntax. Like HTML, XML makes use of *tags* (of the form `<tag>`) and *attributes* (of the form `name="value"`). Like HTML tags, XML tags delimit units of content and identify that content by tag name. Generally, XML statements look something like this:

```
<tag attribute="value">data value or structured content</tag>
```

In standard-compliant XML, tag and attribute names are case-sensitive — that is, `<tag>` is not the same as `<Tag>`. Tag and attribute names may include alphanumeric characters, hyphens, underscores, and periods. Other punctuation marks are generally prohibited, since they may have special meanings in XML.

XML Data Elements

Functionally, XML tags mark *data elements* in text. Data elements are of two types:

- ***Simple data elements*** contain basic data types such as integers, dotted decimal numbers, dates, times, fixed-length or variable-length character strings, or the like. Simple data elements cannot be decomposed into subordinate XML data elements; they are, in that sense, “atomic” units of data. Such a tag might look something like this:

  ```
  <package>ACCT000025</package>
  ```

- ***Complex data elements*** contain a *data structure* composed of one or more subordinate XML data elements, each delimited by its own pair of *subtags* within the main tag pair. The subordinate elements may themselves be either simple or complex. Complex tags may be built up from successively simpler tags to form a hierarchical tree structure. A complex tag structure with just one level of subtags might look something like this:

  ```
  <response>
     <statusMessage>CMN8700I - LIST  Package service completed</status
     <statusReturnCode>00</statusReturnCode>
     <statusReasonCode>8700</statusReasonCode>
  </response>
  ```

The contents of an actual data element must conform to whatever data validation restrictions are imposed by the tag definition. For simple data elements, such restrictions would include data type, data pattern, allowable value range, and/or membership in a predefined value list. For complex data elements, the data *structure* must also conform to the tag definition. Restrictions at this level include allowable subtags, subtag sequencing, mutually exclusive subtag choices, and mandatory subtag inclusion. Restrictions on the minimum and maximum number of consecutive tag repetitions, if any, must also be met.

XML Tag Attributes

Attributes qualify the manner in which a tag is used or processed. One tag may have multiple attributes, so each attribute must be explicitly named. The value assigned to an attribute must appear in double quotes and must be a simple data type — such as a date, a character string, or an integer.

Attributes are not (or should not be) used to hold application data. That's what data elements — i.e., tags and subtags — are for! Attributes are used to:

- ***Identify the subtype of a tag*** that is complex enough to have alternative formats, substructures, or validation requirements.
- ***Identify a particular tag instance*** to distinguish it uniquely from other instances of use.
- ***Set a flag for the target application*** to use when choosing among several data interpretations or processing options.

In the case of Serena XML, attributes are used primarily to identify which of many alternative data structures is intended when a particular tag is used. Depending on the value of the attribute, the allowed subtag content and sequence may vary.

Comments

In addition to tags and attributes, standard-compliant XML allows *comments*. XML comments, like those in HTML, begin with `<!--` and end with `-->`. Multi-line comments are permitted. The end-of-comment delimiter must be preceded by a blank or be the first item on a new line. Double hyphens cannot appear anywhere within the comment body.

An XML comment might look something like this:

```
<!-- This is a comment, line 1.
     This is a comment, line 2. -->
```

Character Entities

XML relies on reserved characters (e.g., angle brackets and double quotes) to delimit language-specific constructs (e.g., tags and attribute values). If you include one of XML's reserved characters in your tag data or in attribute values, the XML parser will attempt to treat it as a reserved character — e.g., as the opening angle bracket for a tag name — with unpredictable results. To get around this difficulty, XML provides a mechanism for *escaping* these characters from the special treatment they normally receive, so that they can be included in ordinary data. This is achieved using *character entity codes*.

Character entity codes begin with an ampersand (`&`) and end with a semicolon (`;`). Between these delimiters is a character entity name that identifies the character represented by the entity code. Numeric character entity codes are also allowed in generic XML; however, the XML Services parser does not support numeric character entities at this time.

Five character entities have predefined names in XML. They are listed in *Exhibit 2-1*.

Exhibit 2-1. XML Character Entities

Entity Code	Character Represented
&lt;	Less-than symbol or opening angle bracket (<)
&gt;	Greater-than symbol or closing angle bracket (>)
&amp;	Ampersand (`&`)
&quot;	Straight, double quotation mark (")
&apos;	Apostrophe or straight, single quotation mark (')

For example, you might use ampersands in the names of program modules that you mention in your package implementation instructions. Simply typing an ampersand, in most cases, would generate a parser error. To insert the ampersand without generating an error, use the `&amp;` character entity where you would normally type an ampersand. For example:

```
<packageImplInst>Requires prior execution of USR&amp;001.</packageImplInst>
```

XML parsers vary in their sensitivity to the occurrence of reserved characters in data. You can usually get away with using a regular apostrophe (') instead of the `&apos;` character entity in data strings, for example. But you should always escape any ampersands or angle brackets in your data strings, and escape *all* special characters in attribute values.

Tip

Use character entities instead of special characters in data or attribute values.

XML Documents as Complex Data Elements

XML documents as a whole are themselves defined as complex data elements. The start and end of the document is identified by a *root tag.* Nested within the root tag are the subtags that make up the content of an *instance document* — that is, an actual XML document containing data. There is one and only one root element in an XML document, and the overall structure of the document is always a hierarchical tree. Data structures that loop back upon themselves are forbidden anywhere in an XML document.

The structure of an XML document and its component data elements is defined externally in one of two types of files: a *Document Type Definition (DTD)* or an XML *schema.* XML Services uses the schema approach, because schemas support more sophisticated and rigorous data typing than DTDs. XML documents can be validated against the relevant schema by an XML *parser* to ensure data validity.

Well-Formed Documents

The elements of XML syntax must be combined in a way that conforms to XML rules for a *well-formed document.* If XML Services receives XML input that is not well-formed, it will return an error and make no attempt to process the service request.

XML rules for a well-formed document mirror those in the latest version of HTML. Unlike past practice with HTML, however, the rules for XML are strictly enforced. In particular:

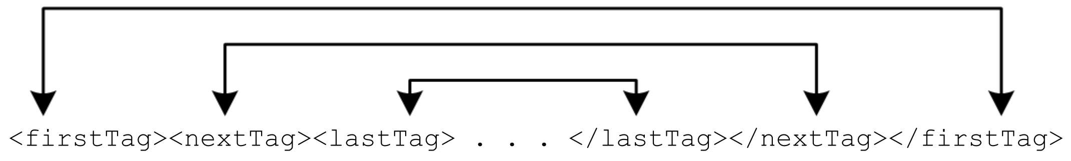
- ***Only one root tag is allowed in a document.*** A well-formed XML document must map to an *n*-way tree data structure. Such a tree has exactly one root node. The root node may have multiple branches to lower-level nodes, each of which may also branch similarly to any depth. Nodes in the tree structure correspond to tags in the XML syntax.
- ***Every opening tag must be matched by a closing tag.*** Closing tags have the same tag name as the opening tag, preceded by a forward slash. For example, the opening tag `<tag>` must be paired with the closing tag `</tag>`.
- ***Standalone tags must be self-closing.*** Standalone tags are defined to mark points in a document rather than contain data; they are explicitly declared to be *“empty”* in the XML schema. Since it contains no data, the standalone opening tag is also the closing tag. As such, it includes a final slash just before the ending angle bracket. For example:

  ```
  <tagname />
  ```

- ***Attribute values must be enclosed in double quotes.*** The quotes are never optional. For example:

  ```
  <tag attribute="value">
  ```

- ***Nested tags must be opened and closed in the proper order.*** The rules for pairing the opening and closing tags in a nested data structure are the same as those for pairing the opening and closing parentheses in a mathematical expression. The first tag opened must be the last tag closed, the next tag opened must be the next-to-last tag closed, and the last tag opened must be the first tag closed. Visually:



```
<firstTag><nextTag><lastTag> . . . </lastTag></nextTag></firstTag>
```

- ***XML comments are comments — and nothing else.*** The frequent HTML practice of embedding non-markup processing instructions in comments is not allowed in XML. Instead, non-XML processing instructions and other non-XML declarations should precede the root tag in the document file.

Strict enforcement of these syntax rules prevents ambiguity when interpreting XML documents. This is vital in XML, because general-purpose XML parsers, unlike their HTML counterparts, can’t rely on the names of tags to help resolve ambiguity.

For example, if you see the tag `'<p>'` in an HTML file, you can assume it marks a paragraph. This works because HTML predefines what each tag and attribute name means in advance and all HTML parsers build in at least some of that knowledge.

However, in XML, you cannot assume anything about the tag `'<p>'`. XML leaves the interpretation of document markup and document content completely to the application that reads it. Tag meaning is defined externally to the document in either a DTD specification or an XML schema specification.

XML Document Declarations

An XML document must identify itself as such to the SERNET messaging server in order to be routed properly to and from XML Services. In addition, once an XML document reaches an XML parser or similar XML processor on either the server or the client, the document must declare the type of XML document it is. This allows the XML parser to interpret the document data structures properly.

Identifying XML Documents

Standard-compliant XML relies on a combination of file naming conventions and declarations in the XML instance document itself to flag XML documents for processing. Conventions for doing this differ somewhat on distributed systems and mainframes.

Distributed systems usually identify XML documents by the Web-style *.xml* file name extension, which is appended to a base file name of up to 8 characters (or more on modern systems). The file name extension identifies the document type immediately for Web browsers and other distributed applications that work with XML. This eliminates the need for these applications to open each document they receive and inspect the contents to determine whether it contains XML. If you access XML Services from a distributed client, you may want to append the *.xml* file extension to any file names when saving reusable Serena XML documents in your local development environment. This facilitates the integration of ChangeMan ZMF with distributed applications.

Mainframes do not support the same file naming conventions used on most distributed systems. The SERNET messaging server therefore cannot rely on file naming conventions to identify XML documents. Instead, SERNET inspects the first line of an incoming message to determine whether or not it contains XML. For this reason, XML Services requires that XML documents always include an `<?xml?>` declaration to identify themselves. This requirement applies regardless of the type of system on which the document originates.

Mainframe users may find it useful to define a library type called “XML” for storing reusable XML documents. However, this is not a requirement of XML Services.

<?XML?> Declaration Syntax

An `<?xml?>` declaration is required on the first line of an XML document. Because it is not properly an XML statement, it precedes the XML root tag of your document. It also precedes any other non-XML declarations or processing instructions that appear before the root tag.

The `<?xml?>` declaration looks something like this:

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

The `version` attribute is required. The `encoding` attribute is optional (the default is UTF-8).

<?XML?> Version Attribute

The `version` attribute in the `<?xml?>` declaration refers to the particular W3C syntax standard followed in your XML document. XML Services recognizes XML Version 1.0, Second Edition, which was published by the W3C in October 2000. This is the latest version of XML. Attempts to use other versions will fail. Consequently, your `<?xml?>` declaration will always have the following `version` attribute:

```
<?xml version="1.0"?>
```

<?XML?> Encoding Attribute

The `encoding` attribute in the `<?xml?>` declaration identifies the character encoding standard used to represent text in your XML document. To ensure both cross-platform and international language compatibility, the W3C specification for XML states that all standard-compliant XML parsers support Unicode. Support for additional character sets is optional.

Unicode is a superset of the 7-bit ASCII character code, with international language and special symbol extensions. The most widely supported variant of Unicode is UTF-8, a variable-length encoding that uses one to four 8-bit bytes to represent characters and symbols. It yields compact files sizes for Latin-based alphabetic text, yet expands to support non-Latin alphabets, ideographic characters, and a wide variety of special symbols on demand. The first 128 code points in UTF-8 — i.e., character codes 0 to 127 — correspond to the same character codes in 7-bit ASCII.

XML Services supports 7-bit ASCII and the full U.S. EBCDIC character set, as well as the subset of UTF-8 that happens to match 7-bit ASCII. Any of the following encoding attributes are therefore valid in the `<?xml?>` declaration for XML Services:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml version="1.0" encoding="US-ASCII"?>
<?xml version="1.0" encoding="EBCDIC-US"?>
```

Note

You may also omit the encoding attribute and it will default to UTF-8.

The values for the `encoding` attribute have the meanings shown in *Exhibit 2-2*.

Exhibit 2-2. XML Character Encoding Attributes

Attribute Value	Character Encoding Description
UTF-8	Variable-length Unicode representation in one to four 8-bit bytes. Supports international languages, including non-Latin and ideographic scripts. The default encoding for XML. XML Services accepts documents with this attribute, but interprets them as 7-bit ASCII at this time. Codes higher than 127 are ignored.

Exhibit 2-2. XML Character Encoding Attributes

Attribute Value	Character Encoding Description
US-ASCII	8-bit ASCII character set. XML Services accepts documents with this attribute, but interprets them as 7-bit ASCII at this time. Codes higher than 127 are ignored.
EBCDIC-US	1987 standard EBCDIC for U.S. English & IBM 3270 terminals. Fully supported by XML Services.

Undefined Character Code Handling

The double-byte variant of Unicode is UTF-16. UTF-16 reserves the range of character codes `E000` – `F8FF` as the Private Use Area (PUA) range. The PUA range is reserved for private use by software vendors.

When converting from EBCDIC to UTF-16 or UTF-8, conversion will fail for characters that are not defined in the EBCDIC code page. To handle characters that fail conversion, SERNET utilizes PUA range `F800` – `F8FF`. For UTF-16, undefined characters are converted to `F8xx`, where `xx` is the hexadecimal value of the undefined EBCDIC character.

For UTF-8, in binary this corresponds to:

```
11101111  101000bb  10bbbbbb
```

where *bbbbbbbb* is the binary value of the undefined EBCDIC character.

When converting from UTF-16 or UTF-8 back to EBCDIC, SERNET will convert the `F8xx` characters back to their original `xx` form.

SERENA XML MESSAGE DOCUMENTS

Every Serena XML request and reply message is an XML document. From a syntactic point of view, each document consists of free-format text delimited by nested markup tags. Tags may be nested to any depth, repeated, or exhibit other forms of structure. The nested tag syntax of an XML document is logically equivalent to a hierarchical *n*-way tree structure.

Serena XML Syntax Example

Syntactically, a Serena XML document begins with a document type declaration, then opens the root `<service> tag`. The document ends with the closing `</service>` tag.

The `name` attribute of the `<service>` tag determines which `<scope>` subtags are valid for nesting within the `<service>` tag for a particular instance document. Similarly, the `name` attribute of the `<scope>` tag determines which `<message>` subtags are valid for nesting within it.

The `<message>` tag completes the trio of nested tags needed to invoke a low-level service object in the Extended Services layer of XML Services. The `name` attribute of the `<message>` tag, in the context provided by the superordinate `<service>` and `<scope>` tags, determines which complex data structures are valid within the `<message>`.

The following Serena XML example illustrates the nested structure of a Serena XML document. The role of the `<service>` tag as the root node is clear from the indentation — although in practice, both indentation and line breaks are optional in XML.

It should also be clear from this example why markup tags in free-format text are so flexible for data interchange. Adding one more tag to some level in the hierarchy does not change the meaning of any other tag in the message.

XML Example — PACKAGE SERVICE CREATE:

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="SERVICE">
  <message name="CREATE">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <applName>ACTP</applName>
    <createMethod>0</createMethod>
    <packageLevel>1</packageLevel>
    <packageType>1</packageType>
    <reasonCode>000</reasonCode>
    <requestorDept>IDD</requestorDept>
    <requestorName>USER24</requestorName>
    <requestorPhone>555 5555</requestorPhone>
    <packageTitle> TEST XML PACKAGE SERVICE CREATE</packageTitle>
    <packageDesc>TEST XML PACKAGE SERVICE CREATE</packageDesc>
    <packageImplInst>TEST XML PACKAGE SERVICE CREATE</packageImplInst>
    <siteInfo>
     <siteName>SERT8</siteName>
     <installDate>20091231</installDate>
     <fromInstallTime>0100</fromInstallTime>
     <toInstallTime>0200</toInstallTime>
     <contactName>USER24</contactName>
     <contactPhone>555 5555</contactPhone>
     <alternateContactName>USER24</alternateContactName>
     <alternateContactPhone>555 5555</alternateContactPhone>
    </siteInfo>
   </request>
  </message>
 </scope>
</service>
```

Logical Document Structure

The logical structure of a ChangeMan ZMF XML Services document can be visualized as an *n*-way hierarchical tree. This structure is illustrated for the high-level nodes common to all services in *Exhibit 2-3*.

Exhibit 2-3. High-Level XML Document Structure

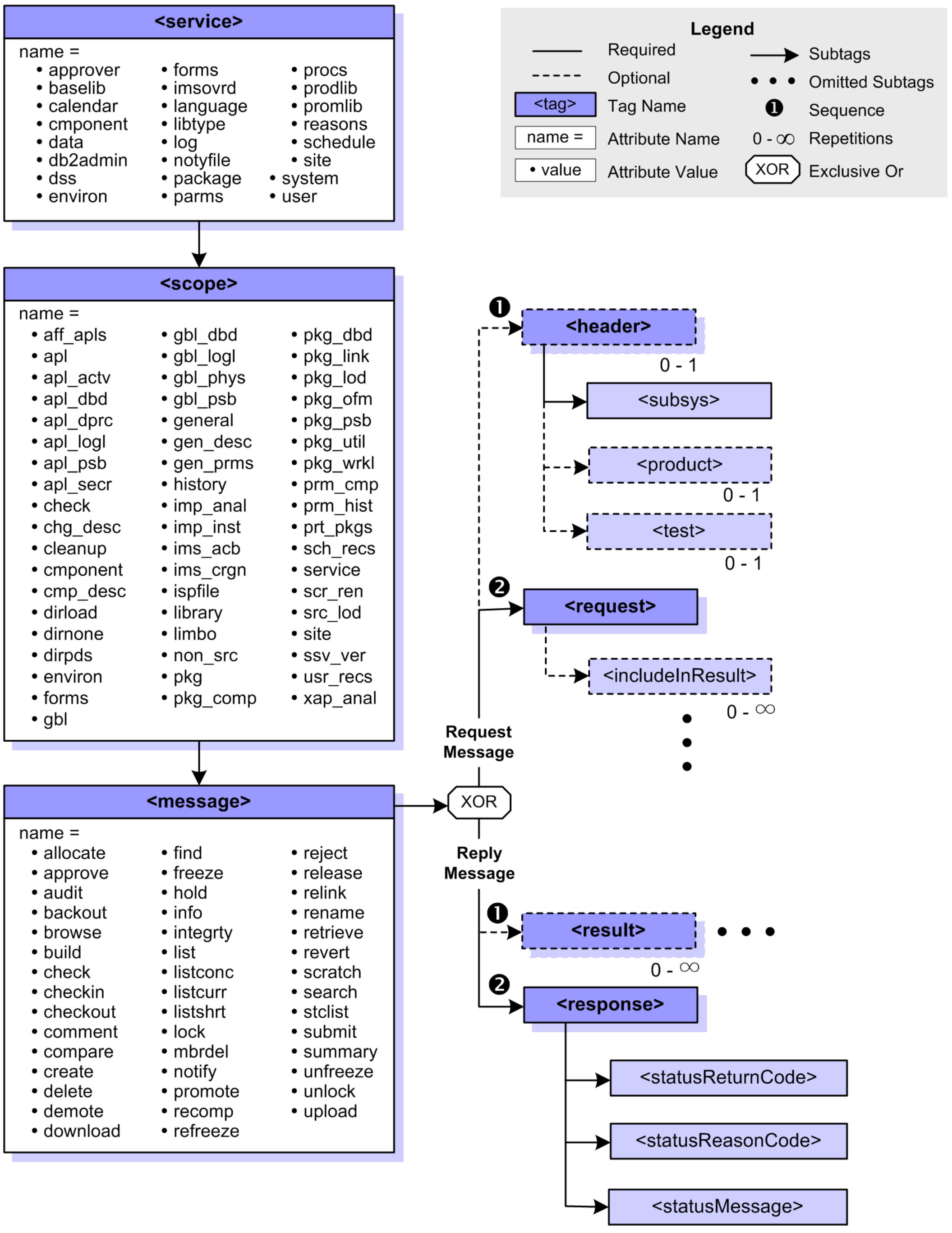


Diagram conventions for Exhibit 2-3 are as follows. Each node of the tree (shown as a rectangle) corresponds to a named data element represented in markup by an XML tag. One or more branches from a node (shown by solid or dashed lines) represent the inclusion of subsidiary nodes in the higher-level node's contents. Dashed borders indicate an optional data element; solid borders indicate that the node is required. Multiple occurrences of a node are allowed — each occurrence of which includes the node's subordinate data structure. Mutually exclusive relationships among nodes is shown by a lozenge-shaped octagon labeled "XOR," from which branches extend to the mutually exclusive nodes with their substructures. Leaf nodes indicate simple data elements containing raw data rather than a substructure of subordinate data elements. An ellipsis (three consecutive dots) indicates the omission of subordinate nodes from the diagram for clarity.

Nodes in the structure diagram are annotated according to the following conventions:

- Tag names appear in the blue region of the node.
- If attributes for the tag exist, their names and permitted values appear in a white region appended to the node.
- If the number of occurrences of a node is variable, the allowed range for the number of repetitions appears below the lower right corner of the node. The number of occurrences can range from zero to unbounded.
- A mandatory sequence for nodes in a data structure is shown by sequence numbers in solid circles at the left of each node in the sequence.

High-Level Tags in Serena XML

A few tags at the highest levels in the Serena XML document hierarchy are used consistently in all XML instance documents. These consistent usage patterns persist despite variations in the low-level service object called, the function requested of that object, or the scope of action requested. These high-level tags are documented below.

<service> Tag: The Root Data Element

The root data element in an XML Services message document is marked by the `<service>` tag. The `<service>` tag identifies the low-level service object that processes the message — such as the approver maintenance service (`name="approver"`) or the package management service (`name="package"`).

The `<service>` tag represents a complex data element with one attribute and one subordinate data element (or subtag). All attributes and subtags are required. The `<service>` tag data structure is summarized in *Exhibit 2-4*.

Exhibit 2-4. Data Structure for Serena XML <service> Tag

Attribute or Subtag	Use	Occurs	Data Type & Length	Description and Values
name	Required	1	String (8), variable	*Attribute.* XML service object name. Actual data length and value fixed for each service object. See *Exhibit 2-10* for allowed values.
<scope>	Required	1	Complex	*Element.* See `<scope>` tag.

<scope> Tag

The `<scope>` tag is the sole subtag of the `<service>` data element. It identifies the types of objects or class of services to be included in the scope of the service object's operations. Example scopes include global records (`name="gbl"`), application records (`name="apl"`), package records (`name="pkg"`), component records (`name="cmponent"`), and service-wide functions (`name="service"`). The chosen scope must be compatible with the requested service. Valid combinations are listed at the end of this chapter in *Exhibit 2-10* and *Exhibit 2-11*.

The `<scope>` tag represents a complex data structure that has one attribute and one subtag. Both are required. The `<scope>` data structure is summarized in *Exhibit 2-5*.

Exhibit 2-5. Data Structure for Serena XML <scope> Tag

Attribute or Subtag	Use	Occurs	Data Type & Length	Description and Values
name	Required	1	String (8), variable	*Attribute.* XML scope name. Must be compatible with service name. Actual data length & value fixed for each service & function. See *Exhibit 2-10* for values.
<message>	Required	1	Complex	*Element.* See `<message>` tag.

<message> Tag

The XML Services `<message>` tag occurs as a subtag of `<scope>`. It identifies the task to be performed by the requested service within the requested scope of action. Example message names include create (`name="create"`), delete (`name="delete"`), update (`name="update"`), list (`name="list"`), and approve (`name="approve"`). Message names must be consistent with the higher-level service and scope names. Valid combinations of service, scope, and message attribute names are listed at the end of this chapter in *Exhibit 2-10* and *Exhibit 2-11*.

The `<message>` tag delimits a complex data element with one attribute and four optional subtags. Subtags must appear in sequence. The use and/or structure of each subtag depends on the service/scope/message combination in the XML document.

The `<message>` tag data structure is summarized in *Exhibit 2-6*.

Exhibit 2-6. Data Structure for Serena XML <message> Tag

Attribute or Subtag	Use	Occurs	Data Type & Length	Description & Values
name	Required	1	String, variable	*Attribute.* XML message type name for service and scope. Actual data length and value fixed for each service object and function. Allowed values appear in *Exhibit 2-10*.
<header>	Required in mainframe batch jobs.	0 - 1	Complex	*Element.* Consistent substructure whenever used. See `<header>` tag below.
<request>	Required in requests. Not used in replies.	0 - 1	Complex	*Element.* Structure varies with service, scope, and message. See particular `<request>` tag for desired user task elsewhere in this manual.
<result>	Optional in replies. Not used in requests.	0 - ∞	Complex	*Element.* Structure varies with service, scope, and message. See particular `<result>` tag for desired user task in *XML Services User Guide*.
<response>	Required in replies. Not used in requests.	0 - 1	Complex	*Element.* Consistent substructure whenever used. See `<response>` tag below.

<header> Tag

The `<header>` tag is a subtag within the `<message>` data structure. It contains routing and test options specific to the ChangeMan ZMF mainframe environment and is required only for TSO batch jobs. It does not appear in reply messages or in request messages submitted interactively.

Syntactically, the `<header>` tag takes the following general form:

```
<header>
   <subsys>P</subsys>
   <product>CMN</product>
   <test>T</test>
</header>
```

Note the absence of a `name` attribute.

Data structure details for the `<header>` tag appear in *Exhibit 2-7*.

Exhibit 2-7. Data Structure for Serena XML <header> Tag

Attribute or Subtag	Use	Occurs	Data Type & Length	Description & Values
<subsys>	Required	1	String (1)	*Element.* One-byte identifier for ChangeMan ZMF instance or subsystem to which request is addressed.

<product>	Optional	0 - 1	String (3)	*Element.* Mnemonic for product to run under subsystem in `<subsys>` tag. Values:
				CMN = ChangeMan ZMF (default)
				XCH = Exchange (ZDD)

<test>	Optional	0 - 1	String (1)	*Element.* Used only at request of Serena Customer Support personnel for diagnostic purposes. Values: **T** = Enable test mode
<warn>	Optional	0 - 1	String (1)	*Element.* Used to enable XML WARN Facility for this XML request. See *"Warn - XML Tag Name Warning" on page 603*. This overrides the XML WARN Facility specification for the started task. Values: **Y** = Enable XML Warning

<request> Tag

The `<request>` tag is a subtag within the `<message>` data structure. It contains the actual content of a Serena XML request message and appears in all requests.

The syntax and structure of the `<request>` tag varies with the service/scope/message combination used in the XML message document. It takes no attributes, and on occasion it may even be empty (i.e., contain no subtags). Further information about specific `<request>` tag structures appears later in this manual.

<result> Tag

The `<result>` tag is a subtag within the `<message>` data structure. It appears only in reply messages and contains the output data, if any, generated by a low-level service object in response to a Serena XML request. It takes no attributes.

The `<result>` tag may be repeated 9,999 times to accommodate multiple result records. For reasons of performance and to minimize memory demands, ZMF limits the maximum number of occurrences of any tag -- including the <results> tag -- to 9999. Each `<result>` tag in a series may contain, for example, a line of code in a browsed component or an item in a list of search results. Alternatively, the tag may not appear at all.

All `<result>` tags in a Serena XML reply message appear before the final `<response>` tag, which contains the return code indicating whether or not the service completed successfully.

The syntax and structure of the `<result>` tag varies by the service/scope/message combination used in the document.

Further information about specific `<result>` tag structures appears later in this manual.

<response> Tag

The `<response>` tag is a subtag of the `<message>` data structure. It contains a mainframe return code, reason code, and/or message concerning the success or failure of your request. The `<response>` tag appears in every reply message issued by XML Services.

The structure of the `<response>` tag is consistent across all service objects and functions, all client environments, and all client products. It contains one required subtag and two optional subtags in a fixed sequence. It takes no attributes.

Syntactically, the `<response>` tag takes the following general form:

```
<response>
   <statusReturnCode>00</statusReturnCode>
   <statusReasonCode>0000</statusReasonCode>
   <statusMessage>Contents of message.</statusMessage>
</response>
```

You should always monitor the contents of the `<statusReturnCode>` tag to trap error conditions. The value returned will be ‘00’ if your request executed successfully. Successively higher numeric values correspond to increasingly severe error conditions. System error codes and ABENDs may append an alphanumeric prefix to the code. You should familiarize yourself with ChangeMan ZMF return codes and messages before taking action on the `<statusReturnCode>` subtag or other elements of the `<response>` tag.

Data structure details for the `<response>` tag appear in *Exhibit 2-8*.

Exhibit 2-8. Data Structure for Serena XML <response> Tag

Subtag	Use	Occurs	Data Type & Length	Description & Values
<statusReturnCode>	Required	1	String (4), variable	*Element.* ChangeMan ZMF return code indicating successful completion or class and severity of error. Typical values: 00 - Execution successful 04 - Warning message 08 - Processing error (e.g., package does not exist) 16 - Syntax error (e.g., unrecognized tag, possibly because of incorrect case) ***NOTE:*** Higher values indicate more severe errors. Abend or system error return codes may exceed 2 bytes & include alphanumerics. ***NOTE:*** Always check this tag to determine success of XML request.

Exhibit 2-8. Data Structure for Serena XML <response> Tag *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Description & Values
<statusReasonCode>	Optional	0 - 1	String (4), variable	*Element.* ChangeMan ZMF reason code indicating type or cause of error, if any. Generally the status codes in XML replies are the same as the internal message numbers. For example, a status code of 8203 corresponds to SERNET message SER8203x
<statusMessage>	Optional	0 - 1	String (255), variable	*Element.* ChangeMan ZMF message text associated with the return code and reason code, if any.

Filtering XML Services Messages

The ChangeMan ZMF XML Services API, like all text markup languages, is verbose. Occasionally, when large volumes of data are returned in response to a request, the verbosity of XML can overwhelm working storage capacity or severely degrade performance. To address this issue, Serena XML supports custom result filtering for XML services that accept `<request>` subtags in the request message and return `<result>` tags in the reply. This is accomplished by using the optional `<includeInResult>` tag in the `<request>` data structure.

<includeInResult> Tag

The `<includeInResult>` tag applies to all XML services with explicit `<request>` subtags in the request message and explicit `<result>` subtags in the reply.

The `<includeInResult>` tag explicitly identifies the subtags to include in the `<result>` tags returned in the XML reply message. The tag is repeatable to accommodate multiple `<result>` subtags. If used, *only* the subtags explicitly named in an instance of `<includeInResult>` will be returned. All other subtags normally returned in the `<result>` by the service are suppressed.

The `<includeInResult>` tag filters returned tags only. XML Services uses this tag to post-process reply messages and strip out extraneous tags as it builds each `<result>` data element. The `<includeInResult>` tag has no effect on the filtering applied by a service when identifying which records to process or include in a report.

An example of the `<includeInResult>` tag in a package general search follows. This example requests a search for all packages in frozen status. But the full set of `<result>` tags is not desired in the reply; instead, only the `<package>` tag and `<auditReturnCode>` will be returned.

Data structure details for the `<includeInResult>` tag appear in *Exhibit 2-9*.

XML Example — Filtering a General Package Search with <includeInResult>

```
<?xml version=”1.0” encoding=”UTF-8”?>
<service name=”PACKAGE”>
   <scope name=”GENERAL”>
      <message name=”SEARCH”>
         <request>
            <searchForFrozenStatus>Y</searchForFrozenStatus>
            <includeInResult>package</IncludeInResult>
            <includeInResult>auditReturnCode</IncludeInResult>
         </request>
      </message>
   </scope>
</service>
```

Exhibit 2-9. <includeInResult> Data Structure

Subtag	Use	Occurs	Data Type & Length	Description & Values
<includeInResult>	Optional in any `<request>` tag	0 - ∞	String (255), variable	Contains desired `<result>` subtag name without angle brackets. ***NOTE:*** Value is case-sensitive.

SERVICE, SCOPE, AND MESSAGE SUMMARY

Only certain combinations of service, scope, and message `name` attributes are valid in Serena XML. The combination chosen must make sense for the low-level service object invoked and for the task or information desired. Valid service/scope/message combinations are listed in the following tables:

- *Core XML Services Summary*
- *ERO XML Services Summary*

Core XML Services Summary

Valid combinations of service, scope, and message names for the core XML Services functions are listed in *Exhibit 2-10*. Names of the corresponding COBOL copybooks are also listed for each function.

Exhibit 2-10. Core XML Service, Scope, and Message Names with COBOL Copybooks

Service Name	Scope Name	Message Name	Description of Function	COBOL Copybook
approver	apl	• list	• List default approver list for application	• XMLCAAPR
	pkg	• list	• List package approvers	• XMLCPAPR

Exhibit 2-10. Core XML Service, Scope, and Message Names with COBOL Copybooks *(Continued)*

Service Name	Scope Name	Message Name	Description of Function	COBOL Copybook
baselib	service	• list	• List baseline library records	• XMLCBASL
calendar	service	• list	• List calendar records by site	• XMLCCLDR
cmponent	apl_cdsc	• find • list	• Find application-level component description • List application-level component description	• XMLCACGD • XLMCACGD
	apl_dprc	• check • find • list	• Check designated build procedure for component • Find components with designated build procedures • List designated build procedures for component	• XMLCADCP • XMLCADCP • XMLCADCP
	apl_secr	• check • find • list	• Check security authorization for component • Find security entity for component • List security entities for component	• XMLCACSC • XMLCACSC • XMLCACSC
	chg_desc	• list	• List active component change description	• XMLCPSVD
	gbl_cdsc	• list	• List global component description	• XMLCGCGD
	gbl_dprc	• list	• List global component build procedure	• XMLCGDCP
	gbl_secr	• list	• List global component member-level security setting	• XMLCGCSC
	history	• list • listbase • listconc • listcurr • listshrt	• List comprehensive component history • List baselined component history • List concurrent development history of component • List current component history • List active component history (short list)	• XMLCCHIS • XMLCCHIS • XMLCCHIS • XMLCCHIS • XMLCCHIS
	lod_subr	• list	• List component subroutines	• XMLCPINC
	pkg_comp	• list	• List source/copybook relationship (ISAL/ICPY) records for components in package	• XMLCPSCC
	pkg_lod	• list	• List load-to-source relationship (ILOD) records for components in package	• XMLCPILC
	pkg_util	• list	• List scratch/rename (IUTL) records for components in package	• XMLCPUTL
	pkg_wrkl	• list	• List users working on component (ICWK)	• XMLCPCUW
	prm_hist	• list	• List component promotion history	• XMLCPPCH
	service	• browse • build • checkin • checkout • compare • lock • recomp • relink • rename • scratch • unlock	• Browse (or download) component • Build component (with stage & compile options) • Check in component • Check out component • Compare component in package vs baseline • Lock component • Recompile component from baseline • Relink component from baseline • Rename a component/member • Scratch a component/member • Unlock component	• XMLCCBRW • XMLCBULD • XMLCCKIN • XMLCCKOT • XMLCCMPR • XMLCCLCK • XMLCRCMP • XMLCRLNK • XMLCSCRN • XMLCSCRN • XMLCCLCK
	src_incl	• list	• List source-to-included-copies relationship records for components in package	• XMLCPISC

Exhibit 2-10. Core XML Service, Scope, and Message Names with COBOL Copybooks
(Continued)

Service Name	Scope Name	Message Name	Description of Function	COBOL Copybook
	ssv_ver	• list • retrieve	• List component staging version change description o • Retrieve staging version of component	• XMLCPSSV • XMLCPSSV
db2admin	apl_actv	• list	• List active DB2 records for application	• XMLCAAD2
	apl_logl	• list	• List logical DB2 records for application	• XMLCALD2
	gbl_logl	• list	• List global DB2 logical records	• XMLCGLD2
	gbl_phys	• list	• List global DB2 physical records	• XMLCGPD2
dss	ispfile	• list	• List ISPF file	• XMLCDSIN
	service	• allocate • basestat • delete • expand • info • list • mbrdel • stclist	• Allocate dataset • List statistics for baseline library member • Delete dataset • Expand baseline member in SRD format • Get dataset allocation information • List dataset member, directory entries, & hash token • Delete dataset member • List datasets allocated to requested DDNAME by the ZMF started task	• XMLCDSAL • XMLCDSBS • XMLCDSDE • XMLCDSEX • XMLCDSIN • XMLCDSLI • XMLCDSMD • XMLCDSST
environ	service	• list	• List ChangeMan ZMF environment parameters	• XMLCENVR
file	dirs	• list	• List HFS directories	• XMLCFILL
	files	• list	• List HFS files in a directory	• XMLCFILL
	service	• access • change • copy • create • delete • download • export • import • link • list • mkdir • rename • rmdir • scan • upload	• List HFS (Hierarchical File System) file access • Change HFS file attributes • Copy HFS file • Create HFS file • Delete HFS file • Download HFS file • Export HFS file • Import HFS file • Link HFS file • List HFS file contents • Make an HFS file directory • Rename an HFS file or directory • Remove an HFS file directory • Scan HFS files for requested strings • Upload HFS files	• XMLCFILA • XMLCFILC • XMLCFILC • XMLCFILC • XMLCFILD • XMLCFILE • XMLCFILE • XMLCFILL • XMLCFILC • XMLCFILC • XMLCFILC • XMLCFILU • XMLCFILM • XMLCFILS • XMLCFILU
forms	gbl	• list	• List global online forms	• XMLCGOFM
	pkg	• approve • comment • detail • list • reject • submit	• Approve online form for package • Add comment or reject reason to form for package • List online form details for package • List online form for package • Reject online form for package • Submit online form for package	• XMLCPOFM • XMLCPOFM • XMLCPOFM • XMLCPOFM • XMLCPOFM • XMLCPOFM
impact	bun	• list	• List BUN library type information	• XMLCIABN
	cmponent	• list	• List impact analysis information for component	• XMLCIACM

Exhibit 2-10. Core XML Service, Scope, and Message Names with COBOL Copybooks
(Continued)

Service Name	Scope Name	Message Name	Description of Function	COBOL Copybook
	table	• list	• List impact analysis table	• XMLCIATB
imscrgn	apl	• list	• List IMS control region defaults for application	• XMLCAICR
	gbl	• list	• List global IMS control region defaults	• XMLCGICR
imsovrd	apl	• apl_dbd • apl_psb	• List IMS DBD overrides for application • List IMS PSB overrides for application	• XMLCAIOR • XMLCAIOR
	gbl	• gbl_dbd • gbl_psb	• List global IMS DBD overrides • List global IMS PSB overrides	• XMLCGIOR • XMLCGIOR
	pkg	• pkg_dbd • pkg_psb	• List IMS DBD overrides for package • List IMS PSB overrides for package	• XMLCPIOR • XMLCPIOR
language	apl	• list	• List default programming language for application	• XMLCALNG
	gbl	• list	• List global default programming language	• XMLCGLNG
libtype	apl	• list	• List library types defined for application	• XMLCALTP
	gbl	• list	• List globally defined library types	• XMLCGLTP
	pkg	• list	• List library types defined for package	• XMLCPLTP
log	service	• create • list	• Create activity log entry • List activity log entries	• XMLCALOG • XMLCALOG
notyfile	service	• download • upload	• Download the global notification file • Upload the global notification file	• XMLCNTFI • XMLCNTFI

Exhibit 2-10. Core XML Service, Scope, and Message Names with COBOL Copybooks
(Continued)

Service Name	Scope Name	Message Name	Description of Function	COBOL Copybook
package	aff_apls	• list	• List affected applications	• XMLCPAAP
	approve	• search	• Search for packages pending approval	• XMLCPSCH
	check	• promote	• Check promotion readiness of package	• XMLCPPRM
	cleanup	• demote	• Demote package & clean up promotion libraries	• XMLCPPRM
	cmponent	• integrty	• Check component integrity of package	• XMLCPINT
	cmp_desc	• list	• List component description records for package	• XMLCPCDS
	forms	• refreeze • unfreeze	• Refreeze online forms for package • Unfreeze online forms for package	• XMLCPFRZ • XMLCPFRZ
	gen_desc	• list	• List general description of package	• XMLCPDSC
	gen_prms	• list • refreeze • unfreeze	• List general parameters for package • Refreeze general parameters for package • Unfreeze general parameters for package	• XMLCPGPM • XMLCPFRZ • XMLCPFRZ
	general	• search	• General package search	• XMLCPSCH
	imp_inst	• list	• List implementation instructions	• XMLCPIMI
	ims_acb	• list	• List IMS ACB control blocks	• XMLCPIAS
	ims_crgn	• list	• List IMS control regions for package	• XMLCPICR
	limbo	• search	• Search for limbo packages	• XMLCPSCH
	non_src	• refreeze • unfreeze	• Refreeze non-source modules in package • Unfreeze non-source modules in package	• XMLCPFRZ • XMLCPFRZ
	pkg_link	• list • search	• List linked packages • Search for linked packages	• XMLCPLNK • XMLCPSCH
	prm_cmp	• list	• List component promotion history for package	• XMLCPPRC
	prm_hist	• list	• List promotion history for package	• XMLCPPRH
	prm_ovly	• list	• List overlaid components for package promotion	• XMLCPPRO
	promote	• lock	• Lock promotion site for a package	• XMLCPPLU
	prt_pkgs	• list	• List participating packages	• XMLCPPPK
	reasons	• list	• List reasons for backout or revert	• XMLCPRBR
	sch_recs	• list	• List installation schedule for package	• XMLCPSCD
	scr_ren	• refreeze • unfreeze	• Refreeze scratched/renamed member • Unfreeze scratched/renamed member	• XMLCPFRZ • XMLCPFRZ

Exhibit 2-10. Core XML Service, Scope, and Message Names with COBOL Copybooks
(Continued)

Service Name	Scope Name	Message Name	Description of Function	COBOL Copybook
	service	• approve • audit • backout • create • delete • demote • freeze • promote • revert • submit • summary	• Package approval action • Audit a frozen package • Back out installed package from production • Create change package • Memo-delete change package • Demote a promoted change package (no cleanup) • Freeze package • Promote package to next promotion library • Revert package to development status • Submit package for file tailoring and JCL build • List package summary statistics	• XMLCPAPV • XMLCPAUD • XMLCPBKO • XMLCPCRT • XMLCPMDL • XMLCPPRM • XMLCPFRZ • XMLCPPRM • XMLCPRVT • XMLCPFTC • XMLCPSUM
	sites	• refreeze • unfreeze	• Refreeze site records for package • Unfreeze site records for package	• XMLCPFRZ • XMLCPFRZ
	src_lod	• refreeze • unfreeze	• Refreeze source & load modules in package • Unfreeze source & load modules in package	• XMLCPFRZ • XMLCPFRZ
	usr_recs	• list	• List user records for package	• XMLCPURC
parms	apl	• list	• List general parameters for application	• XMLCAPRM
	gbl	• list	• List global default general parameters	• XMLCGPRM
procs	apl	• list	• List application procedures	• XMLCAPRC
	gbl	• list	• List global procedures	• XMLCGPRC
prodlib	service	• list	• List production libraries	• XMLCPRDL
promlib	library	• list	• List promotion library records	• XMLCPRLN
	site	• list	• List promotion site records	• XMLCPRSN
reasons	service	• list	• List global reason codes for unplanned changes	• XMLCGRSN
schedule	service	• hold • list • release	• Hold scheduled package installation • List installation schedule records • Release held package installation	• XMLCSCHD • XMLCSCHD • XMLCSCHD
site	apl	• list	• List site records for application	• XMLCASIT
	gbl	• list	• List global site records	• XMLCGSIT
	pkg	• list	• List site records for package	• XMLCPSIT
system	environ	• list	• List SERNET environment parameters	• SERVSYSO
	service	• list	• List system setup & install parameters	• SERVSYSO
user	service	• notify	• Sends notification message to user	• XMLCNTFY
util	line	• print	• SERNET print service	• XMLCUTIL

ERO XML Services Summary

Valid combinations of service, scope, and message names for the Enterprise Release Option (ERO) functions supported by XML Services are listed in the following table*Exhibit 2-11*. COBOL copybook names are also listed for each function. These services are shown here for completeness; they are documented in the *ChangeMan ZMF ERO XML Services User's Guide*.

Exhibit 2-11 . ERO XML Service, Scope, and Message Names with COBOL Copybooks

Service Name	Scope Name	Message Name	Description of Function	COBOL Copybook
package	service	• attach • detach	• Attaches a package to a release • Detaches a package from a release	• XMLCPGPM • XMLCPGPM
rlsmappl	promote	• list	• Lists release management promotion data	• XMLCRPRM
	service	• list • release	• Lists application release status • Lists release data for each release to which an application is joined	• XMLCRAPL • XMLCRARL
	syslib	• list	• Lists SYSLIB data for release applications	• XMLCRASY
rlsmappr	area	• list	• Lists release area approver data	• XMLCRAAP
	ascapprv	• list	• Lists the items that are associated with an approval entity	• XMLCRASC
	global	• list	• Lists global release approval entity data	• XMLCRGAP
	release	• list	• Lists data for install approval entities	• XMLCRAAP
rlsmarea	all_chk	• syslib	• Lists the COPYLIB, LOADLIB, and source concatenation lists for libraries that are allocated	• XMLCRSYL
	all_noc	• syslib	• Lists all of the COPYLIB, LOADLIB, and source concatenation lists, including libraries that are not yet allocated	• XMLCRSYL
	cim	• list	• Lists release area component in motion (CIM) information from the ERO DB2 CIM table	• XMLCRCIM
	cmp_lock	• list	• Lists the holder of a release component lock	• XMLCRCLK
	cpy	• syslib	• Lists the COPYLIB concatenation for a release application	• XMLCRSYL
	detail	• cmp_rlse • integrty • test	• Lists all components in a release concatenation and shows all locations where each component resides • Checks the integrity of the component-in-motion (CIM) table against physical members in area libraries. Checks all versions of all components in the release concatenation. • Tests the contents of a release area against all of the packages that may place a component in that area. Lists information for failing components and packages.	• XMLCRCML • XMLCRCHK • XMLCRTST
	hst	• list	• Lists history from the ERO component history table	• XMLCRHST
	imp	• list	• Lists impact data from the ERO DB2 impact table	• XMLCRIMP
	load	• syslib	• Lists the LOADLIB concatenation for a release application	• XMLCRSYL

Exhibit 2-11 . ERO XML Service, Scope, and Message Names with COBOL Copybooks

Service Name	Scope Name	Message Name	Description of Function	COBOL Copybook
	scan	• cmp_rlse	• Scans the latest version of components in a release concatenation to find those with content matching a search string	• XMLCRCML
	scanall	• cmp_rlse	• Scans all components in a release concatenation to find those with content matching a search string	• XMLCRCML
	service	• list • test	• Lists release area definitions • Tests the contents of a release against all of the packages that may place a component in that release. Displays a message describing the status of packages and components in the release.	• XMLCRARE • XMLCRTST
	source	• syslib	• Lists the source SYSLIB information for a library type	• XMLCRSYL
	start	• list	• Lists the release area definitions for a starting area	• XMLCRARE
	summary	• cmp_rlse • integrty	• Lists information for the latest version of each component in a release concatenation • Checks integrity of the component-in-motion (CIM) table against physical members in area libraries.	• XMLCRCML • XMLCRCHK
	syslib	• list	• Lists SYSLIB data for an application	• XMLCRASL
	ver_regr	• list	• Performs a version regression check on components. If a version regression situation exists between the current release and a prior release, lists information for the current and prior versions.	• XMLCRVER
rlsmltyp	bun	• list	• Lists information from the release BUN library-type table	• XMLCRBUN
	service	• list	• Lists library security and format information	• XMLCRLTP
rlsmrlse	cim	• list	• Lists release area component in motion (CIM) information from the ERO DB2 CIM table	• XMLCRLCM
	detail	• test	• Tests the contents of a release against all of the packages that may place a component in that release	• XMLCRTSC
	hst	• list	• Lists release component history from the ERO component history table	• XMLCRLHT
	imp	• list	• Lists impact data from the ERO DB2 impact table	• XMLCRLMP
	library	• list	• Lists release area libraries	• XMLCRLLT
	prior	• list	• Lists prior release information	• XMLCRLPR
	reasons	• list	• List Backout and Revert reasons for a release	• XMLCRRBR
	rls_link	• list	• Lists release management data across a TCP/IP link	• XMLCRLLK
	service	• list • search • test	• Lists scheduler dates, times, and status for a release • Searches for releases and lists information • Tests the contents of a release against all of the packages that may place a component in that release. Displays a status message.	• XMLCRLSM • XMLCRSRC • XMLCRTSC

Exhibit 2-11 . ERO XML Service, Scope, and Message Names with COBOL Copybooks

Service Name	Scope Name	Message Name	Description of Function	COBOL Copybook
	sites	• list	• Lists release/site dates and contacts	• XMLCRSTE

PACKAGE MANAGEMENT 3

Package management messages in Serena XML fall into four user task categories:

- ***Package Lifecycle Tasks*** — Tasks that comprise a major step in the lifecycle of a change package as a whole. These include package commands such as *package create*, *delete*, *freeze*, *promote*, and *approve*.
- ***Package-Level Component Change Management*** — Tasks related to the component lifecycle but which apply to one or more components of a package as a group. Package-level component groups include source and load modules, non-source modules, and scratch/rename records. Commands include *unfreeze*, *refreeze*, and *list*.
- ***Package Validation Tasks*** — Tasks that identify dependencies among package components, verify the integrity of package components, or check for versioning differences across components in different stages of development. These include package commands such as *list*, *check component integrity* and *audit.*
- ***Package Information Management Tasks***— Tasks that retrieve or manage descriptive metadata or control information about a package. Such information includes the package description, general package parameters, working component descriptions for the package, participating package records, affected application records, package-level site records, the package approver list, package promotion history, user-defined variables for a package, and similar records. Supported commands include *list*.

PACKAGE MESSAGE SYNTAX

Identifying Package Messages

Serena XML package messages contain syntax that tells ChangeMan ZMF to perform a task against a package rather than some other object. This occurs in one of two ways. Most commonly, the `name` attribute in the `<service>` takes the value `"PACKAGE"`, as follows:

```
<service name="PACKAGE">
```

However, some non-package services — such as the approver maintenance service and the site maintenance service — support a package-level scope of action. These identify a package-level task by the `name` attribute of the `<scope>` tag, which takes the value `"pkg"` or something similar (e.g., `"pkg_comp"`, `"pkg_lod"`, and so on). For example:

```
<service name="SITE">
<scope name="PKG">
```

Finally, some services are only implicitly allied to package management; there is no explicit syntax to make that relationship clear. For example, the package installation scheduler service works with install schedules one package at a time. It does not identify its scope as package-specific, though, because its present design gives the scheduler no other scope options.

Where explicit syntax exists, the same attributes appear in both request and reply messages. In requests, they tell ChangeMan ZMF to execute a package-level function. In replies, they tell your XML message processing software to parse the returned message for package data.

Package Naming Conventions

Package Name Tags

Two methods exist in Serena XML to identify a package to ChangeMan ZMF. The first uses the `<package>` tag to supply a complete package name. The second concatenates the `<applName>` tag, which identifies the application to which a package belongs, with the `<packageId>` tag, which contains the unique number of the package within its application. Together, the `<applName>` and `<packageId>` tags yield the same package identifier as that supplied in the `<package>` tag. Either method is acceptable to ChangeMan ZMF.

Embedded Blanks in the <package> Tag

The `<package>` tag appears as a subordinate data element in nearly all package management data structures. For ChangeMan ZMF, this tag takes a 10-byte fixed-format value, as follows:

`<package>`**`aaaannnnnn`**`</package>`, where:

`aaaa` = application name. If less than 4 characters, right-fill with blanks.
`nnnnnn` = package ID number. If less than 6 digits, left-fill with zeroes.

For example, a package name for ChangeMan ZMF that uses a 3-byte application name must include an embedded blank to fill out the application name portion of the `<package>` tag data, as follows:

```
<package>TST 123456</package>
```

Special Tag Syntax for Package Management

Serena XML supports up to 72 user-defined package variables that are established by users when customizing ChangeMan ZMF on the mainframe. These variables are stored in the package master.

The Serena XML tag names for these user-defined package variables use the following naming convention:

`<userVarLen`**xxyy**`>`

where:

- **`xx`** = length of variable data in bytes, formatted as 1-digit or 2-digit integer

- **`yy`** = unique 2-digit integer identifier for this particular variable of length **`xx`**

For example, `<userVarLen103>` represents the third user-defined variable with a length of one byte. Similarly, `<userVarLen4405>` is the fifth variable with a length of 44 bytes.

Serena XML provides 16 such tags for variables of 1 byte each in length, 11 tags of 2 bytes each, 10 tags of 3 bytes each, 10 tags of 4 bytes, 10 tags of 8 bytes, 5 tags of 16 bytes, 5 tags of 44 bytes, and 5 tags of 72 bytes.

Package Lifecycle Tasks

Serena XML supports the following package lifecycle tasks for general use:

- *Create a Package - PACKAGE SERVICE CREATE*
- *Delete a Package - PACKAGE SERVICE DELETE*
- *Freeze a Package - PACKAGE SERVICE FREEZE*
- *Submit a Package for JCL Build - PACKAGE SERVICE SUBMIT*
- *Check a Package for Promotion Readiness - PACKAGE CHECK PROMOTE*
- *Promote a Package - PACKAGE SERVICE PROMOTE*
- *Lock Promotion Site for Package - PACKAGE PROMOTE LOCK*
- *Demote a Package - PACKAGE SERVICE DEMOTE*
- *Demote a Package with Cleanup - PACKAGE CLEANUP DEMOTE*
- *Approve a Package - PACKAGE SERVICE APPROVE*
- *List Package Installation Schedule - SCHEDULE SERVICE LIST*
- *Hold Package Install Job - SCHEDULE SERVICE HOLD*
- *Release Package Install Job - SCHEDULE SERVICE RELEASE*
- *Back Out a Package - PACKAGE SERVICE BACKOUT*
- *Revert a Package - PACKAGE SERVICE REVERT*
-

Create a Package - PACKAGE SERVICE CREATE

The package create message in Serena XML creates an empty change package in the staging area. A parent application must already exist to provide default settings for the new package.

The Serena XML service/scope/message tags and attributes for a package creation message are:

```
<service name=”PACKAGE”>
<scope name=”SERVICE”>
<message name=”CREATE”>
```

These tags appear in both requests and replies.

PACKAGE SERVICE CREATE Requests

The Serena XML syntax for a package creation request varies with the creation method you select. Three creation methods exist:

- ***Short Method*** — Supplies only the minimum information required by the package master database. Complete information is supplied later via package updates using the ChangeMan ZMF ISPF interface. (Serena XML does not support updates to package master records for general use.)
- ***Copy Forward (or Clone) Method*** — Copies values from a preexisting model package into the new package master entry. Changes are made later via package updates using the ChangeMan ZMF ISPF interface. (Serena XML does not support updates to package master records for general use.)
- ***Long Method*** — Supplies all package master information in a single step. No subsequent updates are required. If you want to set the values of any user-defined variables for a package, you must use this method of package creation.

Choose a creation method using the `<createMethod>` subtag of the `<request>` message.

Example XML — PACKAGE SERVICE CREATE Request.

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="SERVICE">
  <message name="CREATE">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <applName>ACTP</applName>
    <createMethod>0</createMethod>
    <packageLevel>1</packageLevel>
    <packageType>1</packageType>
    <reasonCode>000</reasonCode>
    <requestorDept>IDD</requestorDept>
    <requestorName>USER24</requestorName>
    <requestorPhone>555 5555</requestorPhone>
    <packageTitle> TEST XML PACKAGE SERVICE CREATE</packageTitle>
    <packageDesc>TEST XML PACKAGE SERVICE CREATE</packageDesc>
    <packageImplInst>TEST XML PACKAGE SERVICE CREATE</packageImplInst>
    <siteInfo>
     <siteName>SERT8</siteName>
     <installDate>20091231</installDate>
     <fromInstallTime>0100</fromInstallTime>
     <toInstallTime>0200</toInstallTime>
     <contactName>USER24</contactName>
     <contactPhone>555 5555</contactPhone>
     <alternateContactName>USER24</alternateContactName>
     <alternateContactPhone>555 5555</alternateContactPhone>
    </siteInfo>
   </request>
  </message>
 </scope>
</service>
```

The foregoing example requests the creation of a simple, planned, permanent package using the “short” method. The package is part of the “ACTP” application. Installation is scheduled for one production sites.

As the example illustrates, the `<siteInfo>` tag represents a complex data element A complex data element consists of other XML tags, rather than simple data. Such markup syntax, which potentially nests tags within tags within tags to any depth, is how XML implements its hierarchical tree data structure in a text data stream.

In addition, `<siteInfo>` is a repeatable tag. A repeatable tag allows a variable number of consecutive repetitions to accommodate multiple instances of similarly structured information. For example, `<siteInfo>` can be repeated for each site where the newly created package will be installed. Repeatable tags enhance scalability in XML data structures.

Note that the XML data structures for package request and reply messages do not specify any particular order for the occurrence of tags. You must rely on tag name rather than tag ordinal position in a sequence to convey information to ChangeMan ZMF. Sequence within a data structure is not preserved.

For example, a package may be installed across multiple sites in any order. This is not necessarily the order you list your <siteInfo> data elements. Similarly, if you schedule multiple predecessor jobs to occur before package install, they may execute in any order so long as they precede package installation. You cannot assume that predecessor jobs will execute in the order you list them in your XML request.

Caution

Tag sequence is not preserved in package request and reply messages using Serena XML. Use tag names rather than tag ordinal position in a sequence to convey information to ChangeMan ZMF.

Data structure specifications for the package creation `<request>` tag appear in *Exhibit 3-1*.

Exhibit 3-1. PACKAGE SERVICE CREATE <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<affectedApplName>	Optional	0 - ∞	String (4), variable	Name of application affected by one or more participating packages in this complex/super package. Repeatable for multiple applications. ***NOTE:*** Valid only for complex or super packages. ***NOTE:*** If `<partPackageName>` used, at least one instance of this tag is required.
<applName>	Required	1	String (4), variable	Parent application name for new change package.
<complexSuperPackage>	Optional	0 - 1	String (10), variable	Name of complex/super package to which a participating package belongs. ***NOTE:*** Valid only when creating a participating package. ***NOTE:*** Required if `<packageLevel>` value is 4.
<complexSuperPackageAppl>	Optional	0 -1	String (4), variable	Application name of model package. Same as `<complexSuperPackage>` tag's first 4 bytes.
<complexSuperPackage-Number>	Optional	0 -1	Integer(6)	Package ID of model package. Same as `<complexSuperPackage>` tag's last 6 bytes.
<createMethod>	Required	1	Integer (1)	Package creation method code. Values: **0** = Short creation method **1** = Copy forward (clone) method **2** = Long creation method ***NOTE:*** If `<createMethod>` value is **0**, the following additional tags are required: `<packageTitle>`, `<packageLevel>`, `<packageType>`, `<schedulerType>`, `<requestorPhone>`,`<requestorName>`, `<problemActionCode>`, `<siteInfo>`. ***NOTE:*** If `<createMethod>` value is **1**, you must name the package to copy from in `<packageModel>`. ***NOTE:*** If `<createMethod>` value is **2**, you must supply all the tags needed when `<createMethod>` is 0, plus the following: `<packageDesc>`, `<packageImplInst>`, `<problemActionCode>`.

Exhibit 3-1. PACKAGE SERVICE CREATE <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<otherProblemAction>	Optional	0 -1	String (44), variable	Text of "Other" instructions if installation problem occurs. ***NOTE:*** Required when value of `<problemActionCode>` = 3.
<packageApplModel>	Optional	0 -1	String (4), variable	Application name of model package. Same as first 4 bytes of `<packageModel>`.
<packageDesc>	Optional	0 - 46	String (72), variable	Description of package contents. Multiple entries of 72 bytes each.
<packageImplInst>	Optional	0 - 46	String (72), variable	Package install & implementation instructions. Multiple tags of 72 bytes each. ***NOTE:*** Order of repeated tags is not preserved. Add sequence numbers to text if steps must be performed in order.
<packageLevel>	Optional	0 -1	Integer (1)	Code for package complexity or level in hierarchy. Values: **1** = Simple package **2** = Complex package **3** = Super package **4** = Participating package ***NOTE:*** If value = 2 or 3, the names of participating packages are required in the `<partPackageName>` tag. ***NOTE:*** If value = 4, you must supply name of complex/super package in tag `<complexSuperPackage>`.
<packageModel>	Optional	0 -1	String (10), variable	Name of source package from which entries are copied forward ("cloned") to new package. ***NOTE:*** This tag is required if value in `<createMethod>` = 1.
<packageNumberModel>	Optional	0 -1	Integer(6)	Package ID of model package. Same as last 6 bytes of `<packageModel>`.
<packageTitle>	Optional	0 -1	String (255), variable	Working title for package. Appears on most listings & reports.

Exhibit 3-1. PACKAGE SERVICE CREATE <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<packageType>	Optional	0 -1	String (1)	Package install type code. Values: **1** = Planned permanent **2** = Planned temporary **3** = Unplanned permanent **4** = Unplanned temporary ***NOTE:*** For code values = 2 or 4, the duration of change is required in `<tempChangeDuration>` tag. ***NOTE:*** For values = 3 or 4, a reason for the unplanned change is required in the `<reasonCode>` tag.
<partPackageName>	Optional	0 - ∞	String (10), variable	Name of a participating package pointed to by this complex/super package record. Repeatable for multiple participating packages. ***NOTE:*** Valid only when creating a complex or super package. ***NOTE:*** Required if `<packageLevel>` value is 2 or 3. ***NOTE:*** Tag `<affectedApplName>` is also required if this tag is used.
<problemActionCode>	Optional	1	Integer (1)	Code for action to take if problem occurs in package install. Values: **1** = Hold production & contact developer for instructions **2** = Back out change, then proceed with production **3** = Other instructions ***NOTE:*** If value = 3, you must supply instructions in `<otherProblemAction>`.
<reasonCode>	Optional	0 - 1	String (3), variable	Customer-defined reason code for unplanned package installation. ***NOTE:*** Required if `<packageType>` value is 3 or 4. ***NOTE:*** Reason codes defined separately by ZMF administrator.
<release>	Optional, for ERO	0 -1	String (8)	Name of ERO release with which package is associated.
<releaseArea>	Optional, for ERO	0 -1	String (8)	Name of starting release area for ERO release package check-in.
<requestorDept>	Optional	0 -1	String (4), variable	Workgroup or department code for package creator.
<requestorName>	Optional l	1	String (25), variable	Name of developer or contact person responsible for package.

Exhibit 3-1. PACKAGE SERVICE CREATE <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<requestorPhone>	Optional	1	String (15), variable	Phone of developer or contact person responsible for package.
<schedulerType>	Optional	1	Integer (1)	Code for type of installation scheduler used with package. Values: **1** = ChangeMan scheduler **2** = Manual install **3** = Other automated scheduler
<schedulingInfo>	Optional	0 - ∞	Complex	See `<schedulingInfo>` subtag.
<siteInfo>	Optional	0 - ∞	Complex	See `<siteInfo>` subtag.
<tempChangeDuration>	Optional	0 - 1	Integer (3)	Number of days for temporary package to stay installed before automatic backout. ***NOTE:*** Required if `<packageType>` value is 2 or 4.
<userVarLen101> . . . <userVarLen115>	Optional	0 -1 each	String (1)	User-defined variables in ZMF. Total of 15 individually named, 1-byte tags supported. ***NOTE:*** See topic “Package User Information” in the ChangeMan ZMF Customization Guide.
<userVarLen201> . . . <userVarLen211>	Optional	0 -1 each	String (2), variable	User-defined variables in ZMF. Total of 11 individually named, 2-byte tags supported.
<userVarLen301> . . . <userVarLen310>	Optional	0 -1 each	String (3), variable	User-defined variables in ZMF. Total of 10 individually named, 3-byte tags supported.
<userVarLen401> . . . <userVarLen410>	Optional	0 -1 each	String (4), variable	User-defined variables in ZMF. Total of 10 individually named, 4-byte tags supported.
<userVarLen801> . . . <userVarLen810>	Optional	0 -1 each	String (8), variable	User-defined variables in ZMF. Total of 10 individually named, 8-byte tags supported.
<userVarLen1601> . . . <userVarLen1605>	Optional	0 -1 each	String (16), variable	User-defined variables in ZMF. Total of 5 individually named, 16-byte tags supported.

Exhibit 3-1. PACKAGE SERVICE CREATE <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<userVarLen4401> . . . <userVarLen4405>	Optional	0 -1 each	String (44), variable	User-defined variables in ZMF. Total of 5 individually named, 44-byte tags supported.
<userVarLen7201> . . . <userVarLen7205>	Options	0 -1 each	String (72), variable	User-defined variables in ZMF. Total of 5 individually named, 72-byte tags supported.
<workChangeRequest>	Optional	0 -1	String (12), variable	Work order ID or change request number for package.

Tip

Tags: <userVarLen101> to <userVarLen7205>. See topic “Package User Information” in the ChangeMan ZMF Customization Guide.

The `<schedulingInfo>` and `<siteInfo>` tags both represent complex data elements — that is, they contain tags within tags. Their subordinate data structures are described below.

<schedulingInfo> Subtag

The `<schedulingInfo>` tag captures installation scheduling dependencies for a package. Each instance of the tag names a predecessor job and/or a successor job to run before and/or after the installation of the newly created package. The `<schedulingInfo>` tag may be repeated as many times as needed to ensure that all installation prerequisites and follow-up tasks occur. Data structure details for the `<schedulingInfo>` tag appear in the following exhibit.

Exhibit 3-2. <schedulingInfo> Subtag Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<successorJobName>	Optional	0 - 1	String (8), variable	Must be valid job name for system where install takes place.
<predecessorJobName>	Optional	0 - 1	String (8), variable	Must be valid job name for system where install takes place.

<siteInfo> Subtag

The `<siteInfo>` tag provides the site name, contact information, and scheduled package installation date for a remote production site. The tag may be repeated as many times as needed to cover all sites where the newly created package will be installed. At least one instance of the tag is required in a package creation request that uses either the “short” or "long" create method. Data structure details for the `<siteInfo>` tag appear in the following exhibit:

Exhibit 3-3. <siteInfo> Subtag Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<siteName>	Optional	0 - 1	String (8), variable	Name of site where package will be installed.
<installDate>	Optional	0 - 1	Date, yyyymmdd	Planned site install date for package. No punctuation.
<fromInstallTime>	Optional	0 - 1	Time, hhmmss	Start time for period during which site installation of package is planned. 24-hour format, no punctuation.
<toInstallTime>	Optional	0 - 1	Time, hhmmss	End time for period during which site installation of package is planned. 24-hour format, no punctuation.
<contactName>	Optional	0 - 1	String (25), variable	Name of contact person at remote site to assist with install.
<contactPhone>	Optional	0 - 1	String (15), variable	Phone of contact person at remote site to assist with install.
<alternateContactName>	Optional	0 - 1	String (25), variable	Name of alternate contact person at remote site to assist with install.
<alternateContactPhone>	Optional	0 - 1	String (15), variable	Phone of alternate contact person at remote site to assist with install.

PACKAGE SERVICE CREATE Replies

The Serena XML reply message returns, at most, one `<result>` data structure, which reports basic information about the newly created package. Most importantly, the `<result>` supplies a unique package name assigned to the package by ChangeMan ZMF.

Following the `<result>` data structure is the standard `<response>` data structure, which indicates the success or failure of the XML request and provides a status message. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

An example Serena XML package creation reply for a simple, planned, permanent package follows. Tags in bold always occur in a reply. Repeatable tags appear twice for illustration. Data structure details for the package creation `<result>` tag appear in *Exhibit 3-4*.

Example XML — PACKAGE SERVICE CREATE Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="SERVICE">
  <message name="CREATE">
   <result>
```

```
    <package>ACTP000012</package>
    <applName>ACTP</applName>
    <packageId>000012</packageId>
    <packageLevel>1</packageLevel>
    <packageType>1</packageType>
    <packageStatus>6</packageStatus>
    <installDate>20091231</installDate>
   </result>
   <response>
    <statusMessage>CMN2100I - ACTP000012 change package has been created.</
statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>2100</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-4. PACKAGE SERVICE CREATE <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<installDate>	Optional	0 - 1	Date, yyyymmdd	Planned install date for package, or start date of range.
<package>	Optional	0 - 1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), variable	New package ID number generated by ZMF. Same as last 6 bytes of package name.
<packageLevel>	Optional	0 -1	Integer (1)	Code for package complexity level. Values: **1** = Simple package **2** = Complex package **3** = Super package **4** = Participating package

Exhibit 3-4. PACKAGE SERVICE CREATE <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<packageStatus>	Optional	0 - 1	String (1)	Code for status of package in lifecycle. Values: **1** = Approved **2** = Backed out **3** = Baselined **4** = Complex/super pkg closed **5** = Deleted (memo delete) **6** = Development **7** = Distributed **8** = Frozen **9** = Installed **A** = Complex/super pkg open **B** = Rejected **C** = Temporary change cycle completed ***NOTE:*** Only values 6 or A should be returned for package create.
<packageType>	Optional	0 - 1	String (1)	Package install type code. Values: **1** = Planned permanent **2** = Planned temporary **3** = Unplanned permanent **4** = Unplanned temporary

Delete a Package - PACKAGE SERVICE DELETE

The package deletion function in Serena XML flags or unflags an entire package for deletion. Deletion (or undeletion) is logical rather than physical. Physical deletion of flagged packages occurs at a later time under ChangeMan ZMF control.

The Serena XML service/scope/message tags for a package deletion message are:

```
<service name="PACKAGE">
<scope name="SERVICE">
<message name="DELETE">
```

These tags appear in both requests and replies.

PACKAGE SERVICE DELETE Requests

Serena XML supports two kinds of delete requests against a package:

- ***Logical ("Memo") Delete*** — Flags a package for physical deletion at a future time. Package must be in development status prior to memo deletion. To choose this option, enter "1" in the `<processingOption>` tag.
- ***Logical Undelete*** — Removes deletion flag from a memo-deleted package. Assumes the package has not been aged past the scheduled, physical delete date and time. To choose this option, enter "2" in the `<processingOption>` tag.

The following example shows how you might code a logical delete request in Serena XML. Data structure details for the package deletion `<request>` tag appear in *Exhibit 3-5*.

Example XML — PACKAGE SERVICE DELETE Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="SERVICE">
  <message name="DELETE">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <processingOption>1</processingOption>
    <package>ACTP000015</package>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-5. PACKAGE SERVICE DELETE <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<processingOption>	Required	1	Integer (1), fixed	**1** = Logical delete **2** = Logical undelete

PACKAGE SERVICE DELETE Replies

No `<result>` data structure is returned in the package deletion reply message. However, the standard `<response>` data structure is returned to indicate the success or failure of the package deletion request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Freeze a Package - PACKAGE SERVICE FREEZE

On ChangeMan ZMF servers, a Serena XML package freeze request does two things:

- It freezes the package against changes.
- It builds the ".X node" staging library containing file-tailored JCL installation code.

For a freeze request to execute successfully, all of the following conditions must be met:

- The package is in development status.
- All components are active and are at the same promotion level.
- Any online forms in the package have been approved.

In addition, ChangeMan ZMF normally requires that a package pass the audit process before a freeze request can execute successfully.

The Serena XML service/scope/message tags for a package freeze message are:

```
<service name="PACKAGE">
<scope name="SERVICE">
<message name="FREEZE">
```

These tags appear in both requests and replies.

PACKAGE SERVICE FREEZE Requests

Serena XML allows you to freeze a package with or without prior validation of the staging library. Unless you are completely certain that all components in the package are ready to be frozen, you should validate the staging library as part of your package freeze request.

Tip

To validate the staging library as part of a package freeze request, enter "1" in the `<validationParm>` tag.

The example below shows how you might code a package freeze request in Serena XML. Data structure details for the package freeze `<request>` tag follow in *Exhibit 3-6*.

Example XML — PACKAGE SERVICE FREEZE Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="SERVICE">
  <message name="FREEZE">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000012</package>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-6. PACKAGE SERVICE FREEZE <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
userVariable01	Optional	0 - 8	String (8), variable	User variable 01
userVariable02	Optional	0 - 8	String (8), variable	User variable 02
userVariable03	Optional	0 - 8	String (8), variable	User variable 03
userVariable04	Optional	0 - 8	String (8), variable	User variable 04
userVariable05	Optional	0 - 8	String (8), variable	User variable 05
userVariable06	Optional	0 - 72	String (72), variable	User variable 06
userVariable07	Optional	0 - 72	String (72), variable	User variable 07
userVariable08	Optional	0 - 72	String (72), variable	User variable 08
userVariable09	Optional	0 - 72	String (72), variable	User variable 09

Exhibit 3-6. PACKAGE SERVICE FREEZE <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
userVariable10	Optional	0 - 72	String (72), variable	User variable 10
<validationParm>	Optional	0 -1	Integer (1)	**1** = Validate package readiness prior to freeze operation

Tip

Tags: <userVariable01> to <userVariable10>: See topic "Custom V01-V10 Variables" in the ChangeMan ZMF Customization Guide.

PACKAGE SERVICE FREEZE Replies

No `<result>` data structure is returned in the reply message for a package freeze request. However, the standard `<response>` data structure is returned to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Occasionally, a package may freeze successfully but the subsequent file tailoring and JCL build step may not complete. If this occurs, Serena XML provides a way of finishing the file tailoring step on its own.

Tip

Use Serena XML to submit a package for JCL build if the package freeze step is successful, but the subsequent file tailoring and JCL build step does not complete. (See *Submit a Package for JCL Build - PACKAGE SERVICE SUBMIT*.)

Submit a Package for JCL Build - PACKAGE SERVICE SUBMIT

The package service submit request submits a previously frozen package for stage file tailoring — that is, it builds (or rebuilds) the ".X node" staging library containing file-tailored JCL installation and backout code. It performs this task at the package level rather than the component level.

The Serena XML service/scope/message tags for a package submit message are:

```
<service name="PACKAGE">
<scope name="SERVICE">
<message name="SUBMIT">
```

These tags appear in both requests and replies.

When successful, this service submits a JOB with output similar to the following:

```
  SDSF OUTPUT DISPLAY CMN8ADSP S0786765  DSID     4 LINE 71      CO
  COMMAND INPUT ===>                                            SCR
 IEF285I   ZMFA.CMN8ADSP.S0786765.D0000106.?           SYSOUT
 IEF285I   ZMFA.CMN8ADSP.S0786765.D0000107.?           SYSOUT
 IEF373I STEP/        /START 2009065.0630
 IEF374I STEP/        /STOP  2009065.0630 CPU    0MIN 00.47SEC SRB
 IEF375I  JOB/CMN8ADSP/START 2009065.0630
 IEF376I  JOB/CMN8ADSP/STOP  2009065.0630 CPU    0MIN 00.47SEC SRB
     PROG=CMNASPFT,PARMS=PGMCMNVPIJB
 0032ACTP0000138USER35          Y
 READY
 END
 ChangeMan(R)      CMNVPIJB - 6.1.0 File Tailoring

 Function : Package install JCL build
 Subsystem: 8
 Userid   : USER24
 Package  : ACTP000013
 Schedule : Y
 Date/Time: 2009/03/06 06:30:10

 CMN8700I - ACTP000013 Installation JCL Build service completed
```

PACKAGE SERVICE SUBMIT Request

The following example shows how you might code a package service submit request using Serena XML. Data structure details for the packageservice submit `<request>` tags appear in *Exhibit 3-7*.

Example XML — PACKAGE SERVICE SUBMIT Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="SERVICE">
  <message name="SUBMIT">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000013</package>
    <cmnSubSystemId>8</cmnSubSystemId>
    <requestor>USER24</requestor>
    <addSchedulerOption>Y</addSchedulerOption>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-7. PACKAGE SERVICE SUBMIT <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<addSchedulerOption>	Optional	0 - 1	String (1)	Code to add installation scheduler record for automated scheduling system. Values: **Y** = Yes, add scheduler record **N** = No, don't add record **C** = Conditional, add scheduler record only if build succeeds.
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<cmnSubSystemId>	Optional	1	String (1)	ZMF subsystem ID where package resides (for batch execution).
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<requestor>	Optional	1	String (8), variable	Must be valid TSO user ID on mainframe LPAR where ZMF subsystem resides.

PACKAGE SERVICE SUBMIT Replies

No `<result>` data structure is returned in the reply message to a package submit request. However, the standard `<response>` data structure is returned to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Check a Package for Promotion Readiness - PACKAGE CHECK PROMOTE

The promotion check function determines whether a promote request is valid without performing the actual promotion. It ensures that the components to be promoted are active, the requested promotion library is a valid one for the requestor, and the package complies with administrator-defined promotion business rules.

The Serena XML service/scope/message tags for a promotion check message are:

```
<service name="PACKAGE">
<scope name="CHECK">
<message name="PROMOTE">
```

These tags appear in both requests and replies.

PACKAGE CHECK PROMOTE Requests

The syntax of a promotion check message is similar to that of the PACKAGE SERVICE PROMOTE request, with the following exceptions:

- the `name` attribute in the `<scope>` tag has a value of "`CHECK`"
- the `<applName>`, `<packageId>`, `<scheduledate>`, and `<scheduletime>` tags are not used

A code example appears in this chapter under *Promote a Package - PACKAGE SERVICE PROMOTE*. Data structure details for the promotion check `<request>` tag are discussed in *Exhibit 3-8*.

PACKAGE CHECK PROMOTE Replies

No `<result>` data structure is returned in the reply message to a promotion check request. However, the standard `<response>` data structure is returned to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Promote a Package - PACKAGE SERVICE PROMOTE

Package promotion applies the changes in a package to libraries used for testing and other purposes. All components to be promoted must be active, and business rules for promotion level transitions, promotion to remote sites, and package freeze must also be met.

The Serena XML service/scope/message tags for a package promotion message are:

```
<service name="PACKAGE">
<scope name="SERVICE">
<message name="PROMOTE">
```

These tags appear in both requests and replies.

The package promote function validates the promotion readiness of a package prior to executing the promote. It necessarily file-tailors the package for application to the target promotion library, as well — a step that can take some time.

Tip

To check the promotion readiness of a package in Serena XML without file tailoring for promotion or actually executing the promote, use package/check/ promote. (See *Check a Package for Promotion Readiness - PACKAGE CHECK PROMOTE*.)

PACKAGE SERVICE PROMOTE Request

Serena XML supports all three types of promotion: full promote, selective promote, and "first" promote. No special XML attribute or tag is required to choose a promotion type. ChangeMan ZMF determines the appropriate promotion type based on whether or not you supply an explicit component name (which indicates a selective promote), and on the business rules defined for promotion by your administrator (which may or may not allow a "first" promote).

The example below shows how you might code a selective promotion request in Serena XML. Data structure details for the packageservice promote `<request>` tag appear in *Exhibit 3-8*.

Example XML — PACKAGE SERVICE PROMOTE Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="SERVICE">
  <message name="PROMOTE">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000012</package>
    <promotionSiteName>SERT8</promotionSiteName>
    <promotionLevel>10</promotionLevel>
    <promotionName>C001AUT</promotionName>
    <jobCards01>//XMLX130  JOB (AMW,000),'DEFINE UCAT',MSGCLASS=Y,</jobCards01>
    <jobCards02>//             TIME=(,10),NOTIFY=USER24</jobCards02>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-8. PACKAGE SERVICE PROMOTE <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<component>	Optional	0 - 800	Complex	See `<component>` subtag, *Exhibit 3-9*. ***NOTE:*** Required for selective promote. If used, `<listCount>` tag is also required.
<jobCards01>	Required	1	String (72), fixed length	First of up to 4 JCL statements needed to execute the promote in batch mode.
<jobCards02>	Optional	0 - 1	String (72), fixed length	Second of up to 4 JCL statements needed to execute the promote in batch mode.
<jobCards03>	Optional	0 - 1	String (72), fixed length	Third of up to 4 JCL statements needed to execute the promote in batch mode.
<jobCards04>	Optional	0 - 1	String (72), fixed length	Fourth of up to 4 JCL statements needed to execute the promote in batch mode.
<listCount>	Optional	0 - 1	Integer (3), variable	Number of components to selectively promote. Must match number of `<component>` tags. ***Value range:*** 1 - 800 ***NOTE:*** Required for selective promote. If used, `<component>` tag is also required.
<overlayTargetComponents>	Optional	0 - 1	String (1)	Option to automatically overlay package components already in target library. Values: **Y** = Yes, overlay components **N** = No, don't overlay
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.

Exhibit 3-8. PACKAGE SERVICE PROMOTE <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<promotionLevel>	Required	1	Integer (2), variable	Sequence number of target promotion library in promotion hierarchy.
<promotionName>	Required	1	String (8), variable	Promotion/demotion nickname.
<promotionSiteName>	Required	1	String (8), variable	Name of site where target promotion library resides.
<scheduledate>	Optional	0 -1	String (8)	A date with no time (yyyyMMdd)
<scheduletime>	Optional	0 -1	String (4)	A time (HHmm)
<suppressNotify>	Optional	0 -1	String (1)	**Y** = Yes, suppress notify **N** = No, don't suppress
<userVariable01> . . . <userVariable05>	Optional	0 - 1 each	String (8), variable	Up to five user-defined variables of 8 bytes each, used to pass parameters to JCL interpreter.
<userVariable06> . . . <userVariable10>	Optional	0 - 1 each	String (72), variable	Up to five user-defined variables of 72 bytes each, used to pass parameters to JCL interpreter.

Tip

Tags: <userVariable01> to <userVariable10>: See topic "Custom V01-V10 Variables" in the ChangeMan ZMF Customization Guide.

<component> Subtag

In a selective or a "first" package promotion request, you explicitly name each component to promote. The `<component>` subtag serves this purpose. It delimits a complex data structure containing the name and library type of each component to be promoted, and is repeatable as many times as needed to accommodate the components selected for promotion.

This `<component>` tag does not stand alone. When used, it requires a `<listCount>` tag to precede the first instance of the `<component>` tag in the message. The `<listCount>` tag contains a count of components to be promoted. That number must match the actual number of `<component>` tags that immediately follow.

Data structure details for the complex `<component>` subtag appear in *Exhibit 3-9*.

Exhibit 3-9. <component> Subtag Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<componentName>	Required	1	String (256), variable	• If PDS member, the member name (max 8 bytes, no qualifiers). • If HFS file, the Unix-style long file name, optionally prefixed by path from installation root.
<componentType>	Required	1	String (3), fixed	Library type of component in `<componentName>`.

Package Service Promote Reply

No `<result>` data structure is returned in package promotion reply message. However, the standard `<response>` data structure is returned to indicate the success or failure of the promotion request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Lock Promotion Site for Package - PACKAGE PROMOTE LOCK

The Package Promote Lock service locks the promotion site for a requested package.

The Serena XML service/scope/message tags for a promotion site lock message are:

```
<service name="PACKAGE">
<scope name="PROMOTE">
<message name="LOCK">
```

These tags appear in both requests and replies.

PACKAGE PROMOTE LOCK Request

The example below shows how you might code a Package Promote Lock request in Serena XML. Data structure details for the `<request>` tag appear in *Exhibit 3-10*.

Example XML — PACKAGE PROMOTE LOCK Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="PROMOTE">
  <message name="LOCK">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
```

```
    <package>ACTP000012</package>
    <promotionSiteName>SERT8</promotionSiteName>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-10. PACKAGE PROMOTE LOCK <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<promotionSiteName>	Required	1	String (8), variable	Name of site where target promotion library resides.

Package Promote Lock Reply

No `<result>` data structure is returned in a Package Promote Lock reply message. However, the standard `<response>` data structure is returned to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Demote a Package - PACKAGE SERVICE DEMOTE

The standard package demotion function resets the desired components previously promoted to a specific promotion site and level to promotion level 00 in the staging library. In a full demote, it also resets the package master to development status. Copies of previously promoted components are deleted.

The Serena XML service/scope/message tags for a message to demote a package:

```
<service name="PACKAGE">
<scope name="SERVICE">
<message name="DEMOTE">
```

These tags appear in both requests and replies.

PACKAGE SERVICE DEMOTE Request

Serena XML supports both full demotion and selective demotion. No special XML attribute or tag is required to choose a demotion type. ChangeMan ZMF determines the appropriate demotion type based on whether or not you supply an explicit component name (which indicates a selective demote).

Except for the `name` attribute in the `<scope>` tag, the syntax of a request to demote a package is identical to that of a promotion request. A code example appears in this chapter under *Promote a Package - PACKAGE SERVICE PROMOTE*. Data structure details for the promotion check `<request>` tag are discussed in *Exhibit 3-8*, also in *Promote a Package - PACKAGE SERVICE PROMOTE*.

PACKAGE SERVICE DEMOTE Reply

Serena XML reply messages for a package demotion request do not return a `<result>` data structure. They do, however, return a standard `<response>` data structure to indicate the success or failure of the demotion request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Demote a Package with Cleanup - PACKAGE CLEANUP DEMOTE

The package cleanup demote function performs a full package demotion for all package components previously promoted to any promotion level at a named site. The promotion libraries that were last promoted to are cleaned up. It then resets the package master to development status.

The Serena XML service/scope/message tags for a message to demote a package with cleanup are:

```
<service name="PACKAGE">
<scope name="CLEANUP">
<message name="DEMOTE">
```

These tags appear in both requests and replies.

PACKAGE CLEANUP DEMOTE Requests

The example below shows how you might code a request for demotion with cleanup in Serena XML. Data structure details for the `<request>` tag appear in *Exhibit 3-11*.

Example XML — PACKAGE CLEANUP DEMOTE Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="CLEANUP">
  <message name="DEMOTE">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>TES5000004</package>
    <promotionSiteName>SERT8</promotionSiteName>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-11. PACKAGE CLEANUP DEMOTE <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<promotionSiteName>	Required	1	String (8), variable	Name of site where promotion library resides.
<suppressNotify>	Optional	0 - 1	String (1),	Suppress batch messages,Y or N.
<userVariable01> . . . <userVariable05>	Optional	0 - 1 each	String (8), variable	Up to five user-defined variables of 8 bytes each, used to pass parameters to JCL interpreter.

Exhibit 3-11. PACKAGE CLEANUP DEMOTE <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<userVariable06> . . . <userVariable10>	Optional	0 - 1 each	String (72), variable	Up to five user-defined variables of 72 bytes each, used to pass parameters to JCL interpreter.

Tip

Tags: <userVariable01> to <userVariable10>: See topic “Custom V01-V10 Variables” in the ChangeMan ZMF Customization Guide.

PACKAGE CLEANUP DEMOTE Replies

Serena XML reply messages for a package demotion with cleanup do not return a `<result>` data structure. They do, however, return a standard `<response>` data structure to indicate the success or failure of the demotion request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Example XML — PACKAGE CLEANUP DEMOTE Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="CLEANUP">
  <message name="DEMOTE">
   <response>
    <statusMessage>CMN3261I -  request submitted for demotion from
SERT8,C001AUT.</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>3261</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

A successful PACKAGE CLEANUP DEMOTE request will generate a JOB with output similar to the following:

```
   -STEPNAME PROCSTEP     RC    EXCP    CONN     TCB     SRB
 -DEL1CPY                 00      37      18     .00     .00
 -SUCCESS                 00     572     303     .00     .00
 -CHKCOND                 00      15       6     .00     .00
 -FAILURE              FLUSH       0       0     .00     .00
```

```
-PRINT                00     64     24    .00    .00
-CLNLCL               00     30     64    .00    .00
-SERT8    ENDED.  NAME-ACTP                TOTAL TCB
$HASP395 SERT8    ENDED

DELETE ACPCPY00
ACPCPY00 WAS DELETED FROM TARGET DATA SET
*********************************************************
* DDNAME: SUCCESS.SYSPRINT
*********************************************************

ChangeMan(R)      CMNBATCH - 6.1.0  2009/02/17  11:55:22
ATTEMPTING TO INITIATE DIALOG WITH CHANGE MAN SUBTASK
SESSION ESTABLISHED WITH CHANGE MAN SUBTASK
SYSIN: TES5000004 85 FUN=DEMOTE,NOD=SERT8
SYSIN: TES5000004 85 LVL=10,LNM=C001AUT,CID=USER24
SYSIN: TES5000004 85 SUP=YES,SSI=5C6A9D1F
SYSIN: TES5000004 85 TYP=CPY
SYSIN: TES5000004 85 CMP=ACPCPY00
Component History has been updated.
Component Promotion History has been updated
Demotion logged TES5000004
SYSIN: TES5000004 85 FUN=END
Package Promotion history has been updated
Package TES5000004       DEMOTE
Package General record has been updated.
END OF DATA ON SYSIN - TERMINATING
SESSION TERMINATED WITH CHANGE MAN STARTED TASK

<SIZE: RECS=25 BYTES=967>
```

Approve a Package - PACKAGE SERVICE APPROVE

The package approval function logs package approval actions such "approve" and "reject" and issues appropriate notifications. Approval entities may also override their previously defined notification addresses (e.g., to substitute a TCP/IP email address for a TSO "Send" message). Authorized approvers must be defined by approver list maintenance before they can approve a package.

Note

Approver list maintenance is a function of the approver maintenance service, not the package management service. This task is normally performed via ISPF.

The Serena XML service/scope/message tags for a package approval message are:

```
<service name="PACKAGE">
<scope name="SERVICE">
<message name="APPROVE">
```

These tags appear in both requests and replies.

PACKAGE SERVICE APPROVE Requests

The following example shows how you might code a package approval request using Serena XML. Data structure details for the package approval `<request>` tag appear in *Exhibit 3-12*.

Example XML — PACKAGE SERVICE APPROVE Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="SERVICE">
  <message name="APPROVE">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <approverAction>1</approverAction>
    <package>ACTP000009</package>
    <approverEntity>ACTPLEAD</approverEntity>
    <reasons>PACKAGE SERVICE APPROVE TEST</reasons>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-12. PACKAGE SERVICE APPROVE <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.

Exhibit 3-12. PACKAGE SERVICE APPROVE <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<approverAction>	Required	1	Integer (1), fixed	**1** = Approve package **2** = Checkoff **3** = Approval decision pending **4** = Reject package **5** = Under review **6** = Final approval for linked packages ***NOTE:*** If value is 2 or 4, `<reasons>` tag required.
<approverEntity>	Required	1	String (8), variable	Security system entity ID of authorized application approver.
<notifierAgentIpAddress>	Optional	0 - 1	String (32), variable	Network IP address for E-mail notifications. Overrides user record. ***NOTE:*** If used, also requires `<notifierAgentPortid>` tag.
<notifierAgentPortid>	Optional	0 - 1	Integer (5), variable	Network port ID of E-mail server for notifications. Overrides user record. ***NOTE:*** Required with tag `<notifierAgentIpAddress>`.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<reasons>	Optional	0 - 14	String (72), variable	Reject (or checkoff) reasons. May be repeated for multiple comments. ***NOTE:*** If `<approverAction>` value = 2 or 4, this tag is required.

PACKAGE SERVICE APPROVE Replies

Serena XML reply messages to a package approval request do not return a `<result>` data structure. They do, however, return a standard `<response>` data structure to indicate the success or failure of the approval action. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

List Package Installation Schedule - SCHEDULE SERVICE LIST

This function lists installation scheduler records defined for a named package. Information returned includes planned installation dates, install job status if held or released, install job participation in a multi-package release, temporary change duration, and package backout status. If no installation information has been defined, no results are returned.

The Serena XML service/scope/message tags and attributes for messages that *list* installation schedule information for a package are:

```
<service name="SCHEDULE">
<scope name="SERVICE">
<message name="LIST">
```

These tags appear in both requests and replies.

SCHEDULE SERVICE LIST — Requests

Request messages for this function require only a package name. A date range may also be supplied.

Example XML — SCHEDULE SERVICE LIST Request

```
<?xml version="1.0"?>
<service name="SCHEDULE">
 <scope name="SERVICE">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000009</package>
   </request>
  </message>
 </scope>
</service>
```

Data structure details for the `<request>` tag appear in *Exhibit 3-13*.

Exhibit 3-13. SCHEDULE SERVICE LIST<request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), variable	ZMF application name. Same as first 4 bytes of `<package>`. ***NOTE:*** Not recommended. Use `<package>` instead of separately specifying `<applName>` and `<packageId>`. ***NOTE:*** OK to omit trailing blanks.
<backoutJobSubmitted>	Optional	1	String (1)	**Y** = Yes, backout job submitted **N** = Backout job not submitted
<installDate>	Optional	0 - 1	Date, yyyymmdd	Planned install date for package, or start date of range.
<installJobHeld>	Optional	1	String (1)	**Y** = Yes, install job held **N** = No, install job not held
<installJobSubmitted>	Optional	1	String (1)	**Y** = Yes, install job submitted **N** = No, install job not submitted
<isReasonsInserted>	Optional	1	String (1)	**Y** = Yes, reason codes present **N** = No, reason codes absent
<package>	Required	1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID. Same as last 6 bytes of `<package>`. ***NOTE:*** Not recommended. Use `<package>` instead of separately specifying `<applName>` and `<packageId>`. ***NOTE:*** Leading zeroes required.
<releaseInstallation>	Optional	1	String (1)	**Y** = Yes, install with release **N** = No, not a release install
<toInstallDate>	Optional	0 - 1	Date, yyyymmdd	End date of planned installed date range.
<type>	Optional	0 - 1	1	Type of job scheduled, I = Install, P = Promote

SCHEDULE SERVICE LIST — Replies

The Serena XML reply message for this function returns one `<result>` tag, which contains installation scheduler information for a named package. It is followed by the standard `<response>` data element, which indicates the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Example XML — SCHEDULE SERVICE LIST Reply

```
<?xml version="1.0"?>
<service name="SCHEDULE">
 <scope name="SERVICE">
  <message name="LIST">
   <result>
    <package>ACTP000009</package>
    <applName>ACTP</applName>
    <packageId>000009</packageId>
    <type>I</type>
    <installDate>20091231</installDate>
    <installTime>0100</installTime>
    <installJobSubmitted>Y</installJobSubmitted>
    <installJobHeld>Y</installJobHeld>
    <isReasonsInserted>Y</isReasonsInserted>
    <backoutJobSubmitted>Y</backoutJobSubmitted>
    <releaseInstallation>Y</releaseInstallation>
    <tempChangeDuration>000</tempChangeDuration>
    <updateToken>5C7529CB</updateToken>
   </result>
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
 </service>
```

Data structure details for the `<result>` tag appear in *Exhibit 3-14*.

Exhibit 3-14. SCHEDULE SERVICE LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), variable	ZMF application name. Same as first 4 bytes of `<package>`.
<backoutJobSubmitted>	Optional	1	String (1)	**Y** = Yes, backout job submitted **N** = Backout job not submitted
<installDate>	Optional	0 - 1	Date, yyyymmdd	Planned installation date, or first date in range of install dates.
<installJobHeld>	Optional	1	String (1)	**Y** = Yes, install job held **N** = No, install job not held
<installJobSubmitted>	Optional	1	String (1)	**Y** = Yes, install job submitted **N** = No, install job not submitted
<installTime>	Optional	0 - 1	Time, hhmmss	Planned install time in 24-hour format.

Exhibit 3-14. SCHEDULE SERVICE LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<isReasonsInserted>	Optional	1	String (1)	**Y** = Yes, reason codes present **N** = No, reason codes absent
<package>	Optional	1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID. Same as last 6 bytes of `<package>`.
<reasonCode>	Optional	0 - 1	String (3), variable	Reject reason code if package rejected or backed out.
<releaseInstallation>	Optional	1	String (1)	**Y** = Yes, install with release **N** = No, not a release install
<tempChangeDuration>	Optional	0 - 1	String (3), variable	Life of temporary change package before automatic backout.
<type>	Optional	0 - 1	1	Type of job scheduled, I = Install, P = Promote
<updateToken>	Optional	0 - 1	String (8), variable	Binary hash token for updated package.

Hold Package Install Job - SCHEDULE SERVICE HOLD

This function holds a package installation job in the scheduling queue until it is explicitly released. The Serena XML service/scope/message tags and attributes for messages to *hold* a package install job are:

```
<service name=”SCHEDULE”>
<scope name=”SERVICE”>
<message name=”HOLD”>
```

These tags appear in both requests and replies.

SCHEDULE SERVICE HOLD — Requests

The request message for this function requires a package name. No filtering options are supported. Data structure details for the `<request>` tag appear in *Exhibit 3-15*.

Exhibit 3-15. SCHEDULE SERVICE HOLD <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), variable	ZMF application name. Same as first 4 bytes of `<package>`. ***NOTE:*** Not recommended. Use `<package>` instead of separately specifying `<applName>` and `<packageId>`. ***NOTE:*** OK to omit trailing blanks.
<package>	Required	1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID. Same as last 6 bytes of `<package>`. ***NOTE:*** Not recommended. Use `<package>` instead of separately specifying `<applName>` and `<packageId>`. ***NOTE:*** Leading zeroes required.
<type>	Optional	0 - 1	1	Type of job scheduled, I = Install, P = Promote

SCHEDULE SERVICE HOLD — Replies

No `<result>` tag is returned in the Serena XML reply message for a package install job *hold* request. However, the reply message does return a standard `<response>` data element to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Release Package Install Job - SCHEDULE SERVICE RELEASE

This function releases a previously held package installation job in the scheduling queue. The Serena XML service/scope/message tags and attributes for messages to *release* a package install job are:

```
<service name=”SCHEDULE”>
<scope name=”SERVICE”>
<message name=”RELEASE”>
```

These tags appear in both requests and replies.

SCHEDULE SERVICE RELEASE — Requests

The request message syntax to release a package install job is different from that to hold an install job only in the `name` attribute of the `<message>` tag, as shown above. Data structure details for the `<request>` tag are identical in both messages. They appeared previously in *Exhibit 3-15*.

SCHEDULE SERVICE RELEASE — Replies

No `<result>` tags are returned in the Serena XML reply message for a package install job *release* request. However, the reply message does return a standard `<response>` data element to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Back Out a Package - PACKAGE SERVICE BACKOUT

The package service backout function reverses a package baseline ripple. Serena XML does not back out changes to production libraries.

Note

If a package resides in remote production libraries as well as the baseline library, you must back out each installed instance of the package from the production libraries via the ISPF interface before you issue a Serena XML backout request.

The Serena XML service/scope/message tags for a package backout message are:

```
<service name="PACKAGE">
<scope name="SERVICE">
<message name="BACKOUT">
```

These tags appear in both requests and replies.

PACKAGE SERVICE BACKOUT Requests

Serena XML allows you to back out a package with or without validating the integrity of your baseline libraries afterward. This flexibility saves time when backing out minor or temporary changes. However, unless you are completely certain that the changes to be backed out are minor, you should validate baseline integrity as part of the backout process.

An example of how you might code a Serena XML request to back out a package from baseline appears below. Data structure details for the package backout `<request>` tag appear in *Exhibit 3-16*.

Example XML — PACKAGE SERVICE BACKOUT Request

```
<?xml version="1.0" encoding="UTF-8"?>
<service name="PACKAGE">
 <scope name="SERVICE">
  <message name="BACKOUT">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000012</package>
    <siteName>SERT8</siteName>
    <backoutReason01>TEST XML PACKAGE SERVICE BACKOUT</backoutReason01>
    <jobCards01>//XMLX127  JOB (AMW,000),'DEFINE UCAT',MSGCLASS=Y,</jobCards01>
    <jobCards02>//             TIME=(,10),NOTIFY=USER24</jobCards02>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-16. PACKAGE SERVICE BACKOUT <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
backoutReason01	Optional	0 - 1	String (72), variable	Backout reasons line - 1
backoutReason02	Optional	0 - 1	String (72), variable	Backout reasons line - 2
backoutReason03	Optional	0 - 1	String (72), variable	Backout reasons line - 3
backoutReason04	Optional	0 - 1	String (72), variable	Backout reasons line - 4
backoutReason05	Optional	0 - 1	String (72), variable	Backout reasons line - 5
backoutReason06	Optional	0 - 1	String (72), variable	Backout reasons line - 6
backoutReason07	Optional	0 - 1	String (72), variable	Backout reasons line - 7

Exhibit 3-16. PACKAGE SERVICE BACKOUT <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
backoutReason08	Optional	0 - 1	String (72), variable	Backout reasons line - 8
backoutReason09	Optional	0 - 1	String (72), variable	Backout reasons line - 9
<jobCards01>	Optional	0 - 1	String (72), fixed length	First of up to 4 JCL statements needed to execute the promote in batch mode.
<jobCards02>	Optional	0 - 1	String (72), fixed length	Second of up to 4 JCL statements needed to execute the promote in batch mode.
<jobCards03>	Optional	0 - 1	String (72), fixed length	Third of up to 4 JCL statements needed to execute the promote in batch mode.
<jobCards04>	Optional	0 - 1	String (72), fixed length	Fourth of up to 4 JCL statements needed to execute the promote in batch mode.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<siteName>	Optional	1	String (8), variable	Name of site where target demotion library resides.
<validateBackout>	Optional	0 - 1	String (1)	**Y** = Yes, validatebackout only. **N** = No, perform backout.

PACKAGE SERVICE BACKOUT Replies

The Serena XML reply messages to a package backout request do not return a `<result>` data structure. They do, however, return a standard `<response>` data structure to indicate the success or failure of the revert request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Successful requests will send messages like the following to the user who initiated the backout:

```
CMN406I - ACTP000012 BACKED OUT 2009/02/18 @ 08:36:08 AT SERT8, CN(INTERNAL)
CMN410I - ACTP000012 BASELINE REVERSE RIPPLED 2009/02/18 @ 08:36:08.
CN(INTERNAL)
```

Successful requests will submit a BACKOUT JOB with output similar to the following:

```
-STEPNAME PROCSTEP    RC   EXCP   CONN    TCB
-CMN00                00    554    301    .00
-RESTCPY              00    133    245    .00
-DSPTM                00    611    323    .00
-RRIPPIA           FLUSH      0      0    .00
-CMN00                00    552    299    .00
-CMN99                00     14      5    .00
-FAILURE           FLUSH      0      0    .00
-PRINT                00     33     16    .00
-ACTP5512 ENDED.  NAME-ACTP                T
$HASP395 ACTP5512 ENDED
 //*  IMS OPTION: JOB TO PERFORM REVERSE RIPPLE OF PACKAGE ACTP000012
 ChangeMan(R)      CMNBATCH - 6.1.0  2009/02/18  08:36:08
 ATTEMPTING TO INITIATE DIALOG WITH CHANGE MAN SUBTASK
 SESSION ESTABLISHED WITH CHANGE MAN SUBTASK
 SYSIN: ACTP000012 55 NOD=SERT8
 PACKAGE BACKED OUT AT DEV.                    ACTP000012
 BACKOUT AT DEV LOGGED.                        ACTP000012
 BASELINE REVERSE RIPPLE LOGGED                ACTP000012
 END OF DATA ON SYSIN - TERMINATING
 SESSION TERMINATED WITH CHANGE MAN STARTED TASK
```

Revert a Package - PACKAGE SERVICE REVERT

The package revert function reverts a package to development status after it has been backed out from baseline.

The Serena XML service/scope/message names for a package revert message are:

```
<service name="PACKAGE">
<scope name="SERVICE">
<message name="REVERT">
```

These tags appear in both requests and replies.

PACKAGE SERVICE REVERT Requests

You have the option to revert a package with or without concurrent validation of the staging library. However, validation is recommended.

Tip

To validate the staging library as part of your package revert request, enter “2” in the `<validationParm>` tag.

The following example shows how you might code a package revert request using Serena XML. Data structure details for the package revert `<request>` tag appear in *Exhibit 3-17*.

Example XML —PACKAGE SERVICE REVERT Request

```
<?xml version=”1.0”
<service name="PACKAGE">
 <scope name="SERVICE">
  <message name="REVERT">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000012</package>
    <siteName>SERT8</siteName>
   <revertReason01>TEST XML PACKAGE SERVICE REVERT</revertReason01>
    <jobCards01>//XMLX134  JOB (AMW,000),'DEFINE UCAT',MSGCLASS=Y,</
jobCards01>
   <jobCards02>//           TIME=(,10),NOTIFY=USER24</jobCards02>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-17. PACKAGE SERVICE REVERT <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<jobCards01>	Optional	0 - 1	String (72), fixed length	First of up to 4 JCL statements needed to execute the promote in batch mode.
<jobCards02>	Optional	0 - 1	String (72), fixed length	Second of up to 4 JCL statements needed to execute the promote in batch mode.
<jobCards03>	Optional	0 - 1	String (72), fixed length	Third of up to 4 JCL statements needed to execute the promote in batch mode.
<jobCards04>	Optional	0 - 1	String (72), fixed length	Fourth of up to 4 JCL statements needed to execute the promote in batch mode.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<revertReason01>	Optional	1	String (72), variable	Free format text of reason for reverting package to development.
<revertReason02>	Optional	1	String (72), variable	Free format text of reason for reverting package to development.
<revertReason03>	Optional	1	String (72), variable	Free format text of reason for reverting package to development.
<revertReason04>	Optional	1	String (72), variable	Free format text of reason for reverting package to development.
<revertReason05>	Optional	1	String (72), variable	Free format text of reason for reverting package to development.
<revertReason06>	Optional	1	String (72), variable	Free format text of reason for reverting package to development.
<revertReason07>	Optional	1	String (72), variable	Free format text of reason for reverting package to development.

Exhibit 3-17. PACKAGE SERVICE REVERT <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<revertReason08>	Optional	1	String (72), variable	Free format text of reason for reverting package to development.
<revertReason09>	Optional	1	String (72), variable	Free format text of reason for reverting package to development.
<siteName>	Required	1	String (8), variable	Name of site where target revert library resides.
<validationParm>	Optional	0 - 1	Integer (1)	**2** = Determine whether package is eligible for revert. Revert is not actually performed.

PACKAGE SERVICE REVERT Replies

Serena XML replies to a package revert request do not return a `<result>` data structure. They do, however, return a standard `<response>` data structure to indicate the success or failure of the revert request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Package-Level Component Change Management

Package-level component change tasks apply to one or more components within a particular change package. For example, you can work with the source and load components in a package, the non-source components (such as copybooks) in a package, or scratch/rename records in a package. Typical operations on components at the package level are *list*, *unfreeze* and *refreeze*.

Package-level component change management tasks include:

- *Component Change Description List- CMPONENT CHG_DESC LIST*
- *List Staged Components - CMPONENT PKG_COMP LIST*
- *Component Description List- PACKAGE CMP_DESC LIST*
- *List Components With Promotion Overlays - PACKAGE PRM_OVLY LIST*
- *Unfreeze Source/Load Components - PACKAGE SRC_LOD UNFREEZE*
- *Refreeze Source/Load Components - PACKAGE SRC_LOD REFREEZE*
- *Unfreeze Non-Source Components - PACKAGE NON_SRC UNFREEZE*
- *Refreeze Non-Source Components - PACKAGE NON_SRC REFREEZE*
- *List Scratch and Rename Utility Records - CMPONENT PKG_UTIL LIST*
- *Unfreeze Scratch/Rename Records - PACKAGE SCR_REN UNFREEZE*
- *Refreeze Scratch/Rename Records - PACKAGE SCR_REN REFREEZE*

Component Change Description List- CMPONENT CHG_DESC LIST

List all or any components in a package, together with their package-specific change descriptions, using the Serena XML component change description list function. All component types are included in the scope of this function, including source code members, load members, copybooks, skeletons, ISPF panels, and JCL procedures.

The Serena XML service/scope/message names for a component change description list at the package level are:

```
<service name="CMPONENT">
<scope name="CHG_DESC">
<message name="LIST">
```

These tags appear in both requests and replies.

CMPONENT CHG_DESC LIST — Request

Three common uses for component change description lists in Serena XML are:

- ***List All Components in Package*** — Name the desired package in the `<package>` tag. Enter a "match-all" (asterisk) wildcard character in both the `<component>` and `<componentType>` tags, or omit these tags altogether. All components in the package will be returned, together with their package-level change descriptions.
- ***List All Components of Given Library Type*** — Name the desired package in the `<package>` tag and the desired library type in the `<componentType>` tag. Enter a "match-all" (asterisk) wildcard character in the `<component>` tag or omit it altogether. All package components of the desired library type will be returned, together with their change descriptions, if a change description exists.
- ***Get Package-Level Change Description for Named Component*** — Name the desired package in the `<package>` tag and the desired component name in the `<component>` tag. Enter the library type of the component in the `<componentType>` tag if known; otherwise, enter a "match-all" (asterisk) wildcard character. The desired component and its change description are returned if the component exists in the package.

The following example shows how you might code a request to list all components for package ACTP000001 and any existing change descriptions using Serena XML. Data structure details follow the example in *Exhibit 3-18*.

Example XML — CMPONENT CHG_DESC LIST Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="CHG_DESC">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
```

```
    <package>ACTP000007</package>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-18. CMPONENT CHG_DESC LIST <request>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<component>	Optional	0 - 1	String (256), variable	ZMF name of desired component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root. Wildcards & patterns with question mark (?) & asterisk (*) are allowed.
<componentType>	Optional	0 - 1	String (3), variable	Library type for component. Wildcards & patterns with question mark (?) & asterisk (*) are allowed.
<package>	Required	1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.

CMPONENT CHG_DESC LIST — Reply

The XML reply to a component change description list request includes zero to many `<result>` tags. Each `<result>` tag contains the name, library type, and change description of a component in the named package if a change description exists.

A standard `<response>` tag follows the last `<result>` tag to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last tag returned in the reply message, the `<response>` tag also serves as an end-of-list marker.

An example XML reply to a component change description list request appears on the next page. Data structure details for the `<result>` tag follow the example in *Exhibit 3-19*.

Example XML — CMPONENT CHG_DESC LIST Reply

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="CHG_DESC">
  <message name="LIST">
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <component>ACPCPY00</component>
    <componentType>CPY</componentType>
    <changeDesc>SER5904E</changeDesc>
   </result>
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <component>ACPCPY1A</component>
    <componentType>CPY</componentType>
    <changeDesc>SER5904E</changeDesc>
   </result>
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <component>ACPCPY1B</component>
    <componentType>CPY</componentType>
    <changeDesc>SER5904E</changeDesc>
   </result>
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <component>ACPCPY1C</component>
    <componentType>CPY</componentType>
    <changeDesc>SER5904E</changeDesc>
   </result>
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <component>ACPCPY1X</component>
    <componentType>CPY</componentType>
    <changeDesc>SER5904E</changeDesc>
   </result>
.
.
.
.
```

```
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-19. CMPONENT CHG_DESC LIST <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), fixed	ZMF application name. Same as first 4 bytes of package name.
<changeDesc>	Optional	0 - 1	String (35), variable	Description of changes in progress with component in this package.
<component>	Optional	0 - 1	String (256), variable	ZMF name of component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentType>	Optional	0 - 1	String (3), variable	Library type for component.
<package>	Optional	0 - 1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.

List Staged Components - CMPONENT PKG_COMP LIST

List staged components for a package using the Serena XML function to list staged “source-and-load” (ISAL and ICPY) components. These include “like-source”, “like-load”, “like-copybook”, and “like-PDS” components staged from baseline or staged from development.

The Serena XML service/scope/message names for a staged component list at the package level are:

```
<service name=”CMPONENT”>
<scope name=”PKG_COMP”>
<message name=”LIST”>
```

These tags appear in both requests and replies.

CMPONENT PKG_COMP LIST — Request

The primary uses for a request to list staged components are:

- ***List All Staged Components in a Package*** — Name the desired package in the `<package>` tag. Submit a blank in `<recordType>` or omit this tag altogether. Component name, library type, and status are returned for each staged component in the named package.
- ***List Staged Source and Load Components*** — Name the desired package in the `<package>` tag. Enter an “A” in the `<recordType>` tag to request staged source-and-load (ISAL) records. For each staged “like-source” and “like-load” component in the package, this function returns the component name, library type, and status. If a staged like-source component has been compiled while staged, its record will also include a pointer to the primary “like-load” component generated by the compile. “Like-copybook” and “like-PDS” components are not listed.
- ***List Other Staged Components*** — Name the desired package in the `<package>` tag. Enter a “6” in the `<recordType>` tag to request staged copy-and-include (ICPY) records. The function lists component name, library type, and status information for all staged “like-copybook” and “like-PDS” components in the named package, including copybooks, skeletons, JCL procedures, and ISPF panels. Like-source and like-load components are not listed.
- ***Verify That a Particular Component Was Staged*** — Supply the desired component name in `<component>`, the component library type in `<componentType>`, and the package name in `<package>`. Submit a blank in `<recordType>` or omit this tag altogether. If the component was staged to the package named, a `<result>` data structure will return information about the desired component. If the component was not staged to that package, no results will be returned.

To further customize your query for a staged component list request, specify a library type, modification date range, updater ID, or component status of interest. Choose component status options using appropriate yes/no flag tags.

Note

Yes/no flags for component status filtering take default values as a group. The default changes based on whether or not you enter explicit values in these tags, as follows:

- If ***no*** status flag has an explicitly typed value, the default for all tags is “Y”.
- If ***any*** status flag has an explicitly typed value, the default for the remaining tags is “N”.

Build-Option Reply Tags

The following build-option reply tags are not automatically retrieved:

```
<compileOptions>
<linkOptions>
<useDb2PreCompileOption>
```

```
<userOption01>   thru <userOption20>
<userOption0101> thru <userOption0105>
<userOption0201> thru <userOption0203>
<userOption0301> thru <userOption0303>
<userOption0401> thru <userOption0403>
<userOption0801> thru <userOption0805>
<userOption1001> thru <userOption1002>
<userOption1601> thru <userOption1602>
<userOption3401> thru <userOption3402>
<userOption4401> thru <userOption4402>
<userOption6401> thru <userOption6405>
<userOption7201> thru <userOption7205>
```

Displaying these tags causes an increase in run time because the data must be retrieved from the component history records. Therefore, these tags are not retrieved unless you request them using the following tag:

```
<longFormat>Y</longFormat>
```

The default is “N” (do not retrieve the build-option tags).

The following example shows how you might code a request to list all source and load components staged to a package. Data structure details for the `<request>` tag appear in *Exhibit 3-20*.

Example XML — CMPONENT PKG_COMP LIST

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="PKG_COMP">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>CISQ000030</package>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-20. CMPONENT PKG_COMP LIST <request>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<component>	Optional	1	String (256), variable	ZMF name of staged component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root. Asterisk (*) wildcard is allowed.
<componentType>	Optional	0 -1	String (3), variable	Library type of staged component. ***NOTE:*** Takes asterisk (*) wildcard.
<filterActiveStatus>	Optional	0 - 1	String (1)	**Y** = Include active components **N** = Omit active components ***NOTE:*** Part of component status flag group. If *no* tag in group has explicit value, default is Y. If *any* tag in group has explicit value, default is N.
<filterCheckedOutStatus>	Optional	0 - 1	String (1)	**Y** = Include checked-out components **N** = Omit checked-out components ***NOTE:*** Part of component status flag group. If *no* tag in group has explicit value, default is Y. If *any* tag in group has explicit value, default is N.
<filterFrozenStatus>	Optional	0 - 1	String (1)	**Y** = Include frozen components **N** = Omit frozen components ***NOTE:*** Part of component status flag group. If *no* tag in group has explicit value, default is Y. If *any* tag in group has explicit value, default is N.

Exhibit 3-20. CMPONENT PKG_COMP LIST <request> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<filterHfsDirectory>	Optional	0 - 1	String (256), variable	Name of HFS directory containing components to be listed, prefixed by path from installation root (that is, path as stored in baseline library). If present, only files in this directory are listed. If absent, all HFS files meeting other criteria are listed. ***NOTE:*** Applies to z/OS Unix HFS components only. Irrelevant for native z/OS PDS library members.
<filterInactiveStatus>	Optional	0 - 1	String (1)	**Y** = Include inactive components **N** = Omit inactive components ***NOTE:*** Part of component status flag group. If *no* tag in group has explicit value, default is Y. If *any* tag in group has explicit value, default is N.
<filterIncompleteStatus>	Optional	0 - 1	String (1)	**Y** = Include incomplete components **N** = Omit incomplete components ***NOTE:*** Part of component status flag group. If *no* tag in group has explicit value, default is Y. If *any* tag in group has explicit value, default is N.
<filterUnfrozenStatus>	Optional	0 - 1	String (1)	**Y** = Include unfrozen components **N** = Omit unfrozen components ***NOTE:*** Part of component status flag group. If *no* tag in group has explicit value, default is Y. If *any* tag in group has explicit value, default is N.
<fromDateLastModified>	Optional	0 -1	Date, yyyymmdd	Start date in desired range of staged component modification dates.
<lockId>	Optional	0-1	String (7)	UserID component locked by (if locked)
<longFormat>	Optional	0 - 1	String (1)	Tag for requesting the build-option tags from component history data. The default is N. **Y** = Retrieve build-option tags **N** = Do not retrieve build-option tags
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.

Exhibit 3-20. CMPONENT PKG_COMP LIST <request> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<recordType>	Optional	0 - 1	String (1)	Type of staged component record to list. Values: **A** = ISAL (like-source & like-load) **6** = ICPY (like-copybook, like-PDS) **Blank** = Both record types ***NOTE:*** Omit tag or enter explicit blank to list both record types. Null tag returns no records. ***NOTE:*** Asterisk (*) wildcard not accepted in this tag.
<targetComponent>	Optional	0 - 1	String (256), variable	Name of a component, target member name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<toDateLastModified>	Optional	0 -1	Date, yyyymmdd	End date in desired range of staged component modification dates.
<updater>	Optional	0 -1	String (8), variable	TSO user ID of last person to update staged component.

CMPONENT PKG_COMP LIST Replies

The Serena XML reply to a staged component list request returns zero to many `<result>` data structures. Each `<result>` element lists one staged component, together with package name and component status information. If a staged, like-source component has been compiled after staging, the `<result>` also names its primary like-load target component.

In addition to any `<result>` data elements, the reply message returns a standard `<response>` data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because the `<response>` tag follows the last `<result>` tag, it also serves as an end-of-list marker.

The example below shows how a reply for this function might appear in Serena XML. Data structure details for the `<result>` tag appear in *Exhibit 3-21*.

Example XML — CMPONENT PKG_COMP LIST Reply

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="PKG_COMP">
  <message name="LIST">
   <result>
```

```
    <recordType>6</recordType>
    <package>CISQ000030</package>
    <applName>CISQ</applName>
    <packageId>000030</packageId>
    <component>CI2Q101</component>
    <targetComponent>CI2Q101</targetComponent>
    <componentType>LCT</componentType>
    <dateLastModified>20081126</dateLastModified>
    <timeLastModified>094237</timeLastModified>
    <updater>USER24</updater>
    <componentStatus>4</componentStatus>
    <sourceLibOrg>PDS</sourceLibOrg>
    <sourceLib>CMNTP.SERT8.BASE.CISQ.LCT</sourceLib>
    <chkOutLevel>00</chkOutLevel>
    <version>01</version>
    <modLevel>01</modLevel>
    <hashToken>C647B43A0000001B</hashToken>
    <baseDateLastModified>20080407</baseDateLastModified>
    <baseTimeLastModified>095500</baseTimeLastModified>
    <dataType>1</dataType>
    <chkOutToStageLib>N</chkOutToStageLib>
    <chkOutFromBaseLib>N</chkOutFromBaseLib>
    <chkOutToSernet>N</chkOutToSernet>
    <batchChkOut>N</batchChkOut>
    <chkOutComponentDesc>N</chkOutComponentDesc>
    <chkOutFromRelease>N</chkOutFromRelease>
    <lockComponent>Y</lockComponent>
    <checkedOutHashToken>0000000000000000</checkedOutHashToken>
    <lockId>USER015</lockId>
   </result>
.
.
.
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-21. CMPONENT PKG_COMP LIST <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<baseDateLastModified>	Optional	0 -1	Date, yyyymmdd	Date baseline version of staged component was last modified.

Exhibit 3-21. CMPONENT PKG_COMP LIST <result> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<baseSetssi>	Optional	0 - 1	String (8), fixed	Baseline component SETSSI date (seconds since 1/1/1960).
<baseTimeLastModified>	Optional	0 -1	Time, hhmmss	Time baseline version of staged component was last modified, 24 hr format.
<batchChkOut>	Optional	0 - 1	String (1)	**Y** = Batch checkout mode **N** = Not batch checkout mode
<buildProc>	Optional	0 - 1	String (8), variable	Name of required build procedure used with staged component. ***NOTE:*** Applies only to source code component in ISAL records.
<checkedOutHashToken>	Optional	1	String (16), fixed	Component hash at checkout.
<chkOutComponentDesc>	Optional	0 - 1	String (1)	**Y** = Description checked out **N** = Description not checked out
<chkOutFromBaselib>	Optional	0 - 1	String (1)	**Y** = Checked out from baseline **N** = Not checked out from baseline library
<chkOutFromRelease>	Optional	1	String(1)	Y = Checked out from release N = Not checked out from release
<chkOutLevel>	Optional	0 - 1	Integer (2)	Checkout level number for staged component.
<chkOutToSernet>	Optional	0 - 1	String (1)	**Y** = Checked out to SERNET **N** = Not checked out to SERNET
<chkOutToStageLib>	Optional	0 - 1	String (1)	**Y** = Checked out to staging lib **N** = Not checked out to staging
<compileOptions>	Optional	0 - 1	String (34)	Compile options for component not stored elsewhere. ***NOTE:*** Displayed only if <longFormat> request tag = “Y”.
<component>	Optional	1	String (256), variable	ZMF name of staged component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.

Exhibit 3-21. CMPONENT PKG_COMP LIST <result> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<componentStatus>	Optional	0 -1	String (1)	Code for staged component status. Values: **0** = Active **1** = Approved **2** = Checked Out **3** = Demoted **4** = Frozen **5** = Inactive **6** = Incomplete **7** = Promoted **8** = Refrozen **9** = Rejected **A** = Remote Promoted **B** = Submitted **C** = Unfrozen
<componentType>	Optional	0 - 1	String (3), fixed	Library type of staged component.
<dataType>	Optional	0 -1	String (1)	File type of staged component for data transfers. Values: **1** = Text **2** = Binary
<dateLastModified>	Optional	0 -1	Date, yyyymmdd	Date staged component was last modified.
<encryption>	Optional	0 -1	String (8)	Staged component encryption key.
<hashtoken>	Optional	0 - 1	String (16), fixed	Hash token or "fingerprint" of staged component.
<language>	Optional	0 - 1	String (8), variable	Language name of component.
<linkOptions>	Optional	0 - 1	String (34)	Link options for component not stored elsewhere. ***NOTE:*** Displayed only if <longFormat> request tag = "Y".
<lockComponent>	Optional	0 - 1	String (1)	**Y** = Component locked **N** = Component not locked
<lockId>	Optional	0-1	String (7)	UserID component locked by (if locked)
<modLevel>	Optional	0 -1	String (2), fixed	ISPF modification level of staged component.
<package>	Optional	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.

Exhibit 3-21. CMPONENT PKG_COMP LIST <result> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<recordType>	Optional	1	String (1)	Type of staged component record listed. Values: **A** = ISAL (like-source & like-load) **6** = ICPY (like-copybook, like-PDS) **Blank** = Both record types
<setssi>	Optional	0 - 1	String (8), fixed	Staged component SETSSI date (seconds since 1/1/1960).
<sourceLib>	Optional	0 -1	String (44), variable	Data set name of staged component library if PDS.
<sourceLibOrg>	Optional	0 -1	String (3), fixed	Data organization of staged component library. Values: **HFS** = Hierarchical File System **Lib** = Librarian **Pan** = Panvalet **PDS** = PDS or PDS/E **Seq** = Sequential **Oth** = Other
<targetComponent>	Optional	0 - 1	String (256), variable	ZMF name of primary like-load component generated from `<component>` while staged. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<targetLoadLibType>	Optional	0 - 1	String (3), fixed	Library type of component named in `<targetComponent> (relink).`
<timeLastModified>	Optional	0 -1	Time, hhmmss	Time staged component was last modified, 24 hr format.
<updater>	Optional	0 -1	String (8), variable	TSO user ID of last person to update staged component.
<useDb2PreCompileOption>	Optional	0 - 1	String (1)	**Y** = Yes, use DB2 precompile **N** = No, don't precompile for DB2 ***NOTE:*** Displayed only if <longFormat> request tag = "Y".
<userOption01> . . . <userOption20>	Optional	0 - 1	String (1)	Set of up to 20 one-byte, custom, administrator-defined variables. Values: **Y** = Yes **N** = No ***NOTE:*** Displayed only if <longFormat> request tag = "Y".

Exhibit 3-21. CMPONENT PKG_COMP LIST <result> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<userOption0101> . . . <userOption0105>	Optional	0 - 1 each	String (1), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0101 to 0105 on the ISPF user options panel for component build. ***NOTE:*** Displayed only if <longFormat> request tag = "Y".
<userOption0201> . . . <userOption0203>	Optional	0 - 1 each	String (2), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0201 to 0203 on the ISPF user options panel for component build. ***NOTE:*** Displayed only if <longFormat> request tag = "Y".
<userOption0301> . . . <userOption0303>	Optional	0 - 1 each	String (3), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0301 to 0303 on the ISPF user options panel for component build. ***NOTE:*** Displayed only if <longFormat> request tag = "Y".
<userOption0401> . . . <userOption0403>	Optional	0 - 1 each	String (4), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0401 to 0403 on the ISPF user options panel for component build. ***NOTE:*** Displayed only if <longFormat> request tag = "Y".
<userOption0801> . . . <userOption0805>	Optional	0 - 1 each	String (8), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0801 to 0805 on the ISPF user options panel for component build. ***NOTE:*** Displayed only if <longFormat> request tag = "Y".
<userOption1001> . . . <userOption1002>	Optional	0 - 1 each	String (10), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 1001 to 1002 on the ISPF user options panel for component build. ***NOTE:*** Displayed only if <longFormat> request tag = "Y".

Exhibit 3-21. CMPONENT PKG_COMP LIST <result> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<userOption1601> . . . <userOption1602>	Optional	0 - 1 each	String (16), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 1601 to 1603 on the ISPF user options panel for component build. ***NOTE:*** Displayed only if <longFormat> request tag = “Y”.
<userOption3401> . . . <userOption3402>	Optional	0 - 1 each	String (34), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 3401 to 3402 on the ISPF user options panel for component build. ***NOTE:*** Displayed only if <longFormat> request tag = “Y”.
<userOption4401> . . . <userOption4402>	Optional	0 - 1 each	String (44), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 4401 to 4402 on the ISPF user options panel for component build. ***NOTE:*** Displayed only if <longFormat> request tag = “Y”.
<userOption6401> . . . <userOption6405>	Optional	0 - 1 each	String (64), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 6401 to 6405 on the ISPF user options panel for component build. ***NOTE:*** Displayed only if <longFormat> request tag = “Y”.
<userOption7201> . . . <userOption7205>	Optional	0 - 1 each	String (72), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 7201 to 7205 on the ISPF user options panel for component build. ***NOTE:*** Displayed only if <longFormat> request tag = “Y”.
<utilType>	Optional	0 -1	String (1), fixed	Utility type - ‘C’ recompile, ‘L’ relink.
<version>	Optional	0 -1	String (2), fixed	ISPF version number of staged component.

Component Description List- PACKAGE CMP_DESC LIST

List the component descriptions for specified components and types within a package using the Serena XML "package component description list" function. All component types are included in the scope of this function, including source code members, load members, copybooks, skeletons, ISPF panels, and JCL procedures.

The Serena XML service/scope/message names for a component description list at the package level are:

```
<service name="PACKAGE">
<scope name="CMP_DESC">
<message name="LIST">
```

These tags appear in both requests and replies.

PACKAGE CMP_DESC LIST — Request

Three common uses for package component description lists in Serena XML are:

- ***List All Components in Package*** — Name the desired package in the `<package>` tag. Enter a "match-all" (asterisk) wildcard character in both the `<component>` and `<componentType>` tags. All components in the package that have a description will be returned.
- ***List All Components of Given Library Type*** — Name the desired package in the `<package>` tag and the desired library type in the `<componentType>` tag. Enter a "match-all" (asterisk) wildcard character in the `<component>` tag. All package components of the desired library type will be returned, together with their descriptions, if a description exists.
- ***Get Description for Named Component*** — Name the desired package in the `<package>` tag and the desired component name in the `<component>` tag. Enter the library type of the component in the `<componentType>` tag if known; otherwise, enter a "match-all" (asterisk) wildcard character. The desired component and its description are returned if the component exists in the package and it has a description.

The following example shows how you might code a request to list the description for a specific component in package ACTP000007. Data structure details follow the example in *Exhibit 3-22*.

Example XML — PACKAGE CMP_DESC LIST Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="CMP_DESC">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
```

```
    <package>ACTP000007</package>
    <component>ACPSRC1A</component>
    <componentType>SRC</componentType>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-22. PACKAGE CMP_DESC LIST <request>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<component>	Required	1	String (256), variable	ZMF name of desired component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root. Wildcards & patterns with question mark (?) & asterisk (*) are allowed.
<componentType>	Required	1	String (3), variable	Library type for component. Wildcards & patterns with question mark (?) & asterisk (*) are allowed.
<package>	Required	1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.

PACKAGE CMP_DESC LIST — Reply

The XML reply to a package component description list request includes zero to many `<result>` tags. Each `<result>` tag contains the name, library type, and description of a component in the named package if a description exists.

A standard `<response>` tag follows the last `<result>` tag to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last tag returned in the reply message, the `<response>` tag also serves as an end-of-list marker.

An example XML reply appears on the next page. Data structure details for the `<result>` tag follow the example in *Exhibit 3-23*.

Example XML — PACKAGE CMP_DESC LIST Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="CMP_DESC">
  <message name="LIST">
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <component>ACPCPY00</component>
    <componentDesc>ACCOUNT REC 00</componentDesc>
    <componentType>CPY</componentType>
   </result>
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <component>ACPCPY1A</component>
    <componentDesc>ACCOUNT REC 1A</componentDesc>
    <componentType>CPY</componentType>
   </result>
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <component>ACPCPY1B</component>
    <componentDesc>ACCOUNT REC 1B</componentDesc>
    <componentType>CPY</componentType>
   </result>
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <component>ACPCPY1C</component>
    <componentDesc>ACCOUNT REC 1C</componentDesc>
    <componentType>CPY</componentType>
   </result>
.
.
.
.
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-23. PACKAGE CMP_DESC LIST <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), fixed	ZMF application name. Same as first 4 bytes of package name.
<component>	Optional	0 - 1	String (256), variable	ZMF name of component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentDesc>	Optional	0 - 48	String (72), variable	Component description.
<componentType>	Optional	0 - 1	String (3), variable	Library type of component.
<package>	Optional	0 - 1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.

List Components With Promotion Overlays - PACKAGE PRM_OVLY LIST

If the promotion of a package would potentially cause some components to overwrite others of the same name — for example, as part of another package already in testing — you can know in advance using Serena XML. This function includes all component types and all promotion libraries for the package in its scope.

The Serena XML service/scope/message tags for a message to *list* package components with promotion overlays are:

```
<service name="PACKAGE">
<scope name="PRM_OVLY">
<message name="LIST">
```

These tags appear in both requests and replies.

PACKAGE PRM_OVLY LIST — Requests

Serena XML supports two kinds of component overlay lists:

- ***All Components with Promotion Overlays*** — Name the desired package in the `<package>` tag and specify the promotion level of interest using the `<promotionName>`, `<promotionLevel>`, and `<promotionSiteName>` tags. Omit the `<componentNameAndType>` tag. The function returns promotion overlay information for all staged package components with duplicate component names and library types in the chosen promotion environment.
- ***Promotion Overlays for Named Component(s)*** — Name the desired package in the `<package>` tag and specify the promotion level of interest using the `<promotionName>`, `<promotionLevel>`, and `<promotionSiteName>` tags. Itemize the components to check for promotion overlays using the `<componentNameAndType>` data element. A count of the itemized components is required in the `<listcount>` tag. The function returns overlay information only if an itemized component is duplicated in the target promotion environment.

The following example shows how you might code a request to check a particular package component for overlays in a named promotion library. Data structure details for the `<request>` tag appear in *Exhibit 3-24*.

Example XML — PACKAGE PRM_OVLY LIST Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="PRM_OVLY">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000002</package>
    <promotionSiteName>SERT8</promotionSiteName>
    <promotionLevel>10</promotionLevel>
    <promotionName>C001AUT</promotionName>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-24. PACKAGE PRM_OVLY LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<componentNameAndType>	Optional	0 - ∞	Complex	Complex element identifies component(s) to check selectively. See *Exhibit 3-25*. ***NOTE:*** If used, `<listCount>` required. ***NOTE:*** Omit tag to list all components in package with promotion overlays.
<listCount>	Optional	0 - 1	Integer (3), variable	Count of `<componentNameAndType>` tags included in request. ***NOTE:*** If `<componentNameAndType>` used, this tag is required.
<package>	Required	1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<promotionLevel>	Required	1	Integer (2), variable	Numeric code of promotion level to check for potential component overlays.
<promotionName>	Required	1	String (8), variable	ZMF name of promotion level to check for potential component overlays.
<promotionSiteName>	Required	1	String (8), variable	ZMF name of promotion site to check for potential component overlays.
<recallMigratedLib>	Optional	0 - 1	String (1)	**Y** = Yes, recall migrated shadow library **N** = No, don't recall shadow library

Note that <componentNameAndType> is a complex data element with subtags of its own. Its data structure appears in *Exhibit 3-25*.

Exhibit 3-25. <componentNameAndType> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<componentName>	Optional	0 - 1	String (256), variable	ZMF name of desired component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentType>	Optional	0 - 1	String (3), variable	Library type for component in <componentName>.

PACKAGE PRM_OVLY LIST — Replies

The Serena XML reply to a component overlay list request returns zero to many <result> data structures. Each <result> lists one component with potential component overlays in the named promotion library, together with package and component promotion status.

A package component has potential overlay issues in the target promotion library if:

- A component with the same name and library type already exists in the target.
- A component with the same name and library type exists in the promotion history records for the target.

If no duplicate components are found in either the target promotion library or its history records, no results are returned by this function.

In addition to any <result> tags, the reply message returns a standard <response> data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because the <response> tag follows the last <result> tag, it also serves as an end-of-list marker.

The example below shows how a reply for this function might appear in Serena XML. Data structure details for the <result> tag appear in *Exhibit 3-26*.

Example XML — PACKAGE PRM_OVLY LIST Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="PRM_OVLY">
  <message name="LIST">
   <result>
    <component>ACPSRS00</component>
    <componentType>SRS</componentType>
    <isComponentRestaged>N</isComponentRestaged>
       <overlayStatus>C</overlayStatus>
       <package>TES5000001</package>
       <applName>TES5</applName>
       <packageId>000001</packageId>
       <promotionSiteName>SERT8</promotionSiteName>
       <promotionLevel>10</promotionLevel>
       <promotionName>C001AUT</promotionName>
       <packageStatus>6</packageStatus>
       <promoter>USER24</promoter>
       <promotionDate>20090217</promotionDate>
       <promotionTime>105054</promotionTime>
      </result>
   .
   .
   .
      <response>
       <statusMessage>CMN8700I - Overlay service completed</statusMessage>
       <statusReturnCode>00</statusReturnCode>
       <statusReasonCode>8700</statusReasonCode>
      </response>
     </message>
    </scope>
   </service>
```

Exhibit 3-26. PACKAGE PRM_OVLY LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<component>	Optional	1	String (256), variable	ZMF name of staged component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentType>	Optional	0 - 1	String (3), fixed	Library type of staged component.
<isComponentRestaged>	Optional	0 - 1	String (1)	**Y** = Yes, component is restaged **N** = No, component not restaged

Exhibit 3-26. PACKAGE PRM_OVLY LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<overlayStatus>	Optional	0 - 1	String (1)	Code for overlay status of this component in this promotion library. Values: **N** = Exists in promotion library but has no history **H** = Exists in promotion history but not in promotion library **C** = Common to both promotion library and history
<package>	Optional	0 - 1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<packageStatus>	Optional	1	String (1)	Code for status of package in lifecycle. Values: **1** = Approved **2** = Backed out **3** = Baselined **4** = Complex/super pkg closed **5** = Deleted (memo delete) **6** = Development **7** = Distributed **8** = Frozen **9** = Installed **A** = Complex/super pkg open **B** = Rejected **C** = Temporary change cycle completed
<promoter>	Optional	0 - 1	String (8), variable	TSO user ID of latest package promoter.
<promotionDate>	Optional	0 - 1	Date, yyyymmdd	Latest promotion date for package.
<promotionLevel>	Optional	0 - 1	Integer (2), variable	Numeric code of promotion level to check for potential component overlays.
<promotionName>	Optional	0 - 1	String (8), variable	ZMF name of promotion level to check for potential component overlays.
<promotionSiteName>	Optional	0 - 1	String (8), variable	ZMF name of promotion site to check for potential component overlays.

Exhibit 3-26. PACKAGE PRM_OVLY LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<promotionTime>	Optional	0 - 1	Time, hhmmss	Latest promotion time for package, 24-hour format.
<release>	Optional (ERO only)	0 - 1	String (8), variable	Name of release to which package is attached.

Unfreeze Source/Load Components - PACKAGE SRC_LOD UNFREEZE

You can use Serena XML to unfreeze one or more “like-source” or “like-load” components in a package. “Like-copybook” or “like-PDS” components such as copybooks, skeletons, JCL procedures, or ISPF panels are not included in the scope of this function.

The Serena XML service/scope/message tags for a package-level *unfreeze* message for source and load components are:

```
<service name=”PACKAGE”>
<scope name=”SRC_LOD”>
<message name=”UNFREEZE”>
```

These tags appear in both requests and replies.

PACKAGE SRC_LOD UNFREEZE — Requests

Serena XML supports two types of unfreeze requests for source and load components:

- ***Full Unfreeze*** — Unfreezes all source and load component in the named package. This is the default.
- ***Selective Unfreeze*** — Unfreezes a subset of individually named source and/or load components in the named package. Desired components are itemized by name and library type in the `<component>` data element. A count of the itemized components to unfreeze is required in the `<listcount>` tag.

The following example shows how you might code a full unfreeze request for all components in a package. Data structure details for the `<request>` tag appear in *Exhibit 3-27*.

Example XML — PACKAGE SRC_LOD UNFREEZE Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="SRC_LOD">
  <message name="UNFREEZE">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000013</package>
    </request>
   </message>
  </scope>
</service>
```

Exhibit 3-27. PACKAGE SRC_LOD UNFREEZE <request>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<component>	Optional	0 - 800	Complex	Complex identifier for each component to selectively unfreeze or refreeze. See *Exhibit 3-28*. ***NOTE:*** If used, `<listCount>` tag also required.
<listCount>	Optional	0 - 1	Integer (3), variable	Number of components to selectively unfreeze or refreeze. Must match number of `<component>` tags. ***Value range:*** 1 - 800 ***NOTE:*** If `<component>` tag used, this tag is required.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.

The <component> subtag represents a complex data structure that is frequently reused among the package-level requests in Serena XML. Data structure details for this tag appear in *Exhibit 3-28* below.

Exhibit 3-28. <component> Subtag Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<componentName>	Required	0 - 1	String (256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentType>	Required	0 - 1	String (3), fixed	Library type of component in <componentName>.

PACKAGE SRC_LOD UNFREEZE — Replies

The Serena XML reply to a source and load component unfreeze request does not return a <result> data structure. It does, however, return a standard <response> data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Example XML — PACKAGE SRC_LOD UNFREEZE Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="SRC_LOD">
  <message name="UNFREEZE">
   <response>
    <statusMessage>CMN8700I - UNFREEZE:SRC_LOD  service completed</
statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Refreeze Source/Load Components - PACKAGE SRC_LOD REFREEZE

The inverse of the Serena XML unfreeze function for source and load components is the refreeze function for these components. Like its inverse, the refreeze function applies to one or more “like-source” or “like-load” components. Other components — copybooks, skeletons, JCL procedures, ISPF panels, and the like — are not included in the scope of this function.

The Serena XML service/scope/message tags for a package-level *refreeze* message for source and load components are:

```
<service name=”PACKAGE”>
<scope name=”SRC_LOD”>
<message name=”REFREEZE”>
```

These tags appear in both requests and replies.

Refreeze Source and Load Components — Requests

As with unfreeze requests, Serena XML supports two types of package-level refreeze requests for source and load components:

- ***Full Refreeze*** — Refreezes all source and load components in the named package. This is the default.
- ***Selective Refreeze*** — Refreezes a subset of individually named source and load components in the named package. Desired components are itemized by name and library type in the `<component>` data element. A count of the itemized components to refreeze is required in the `<listcount>` tag.

The `<request>` tag syntax for a source-and-load component refreeze request is identical to that for an source-and-load component unfreeze request. (See *Exhibit 3-27*.) Only the `name` parameter in the high-level `<message>` tag differs in this request, as shown above.

Example XML — PACKAGE SRC_LOD REFREEZE Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="SRC_LOD">
  <message name="REFREEZE">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000013</package>
   </request>
  </message>
 </scope>
</service>
```

PACKAGE SRC_LOD REFREEZE — Replies

The Serena XML reply to a source-and-load component refreeze request does not return a `<result>` data structure. It does, however, return a standard `<response>` data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Example XML — PACKAGE SRC_LOD REFREEZE Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="SRC_LOD">
  <message name="REFREEZE">
   <response>
    <statusMessage>CMN8700I - REFREEZE:SRC_LOD  service completed</
statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Unfreeze Non-Source Components - PACKAGE NON_SRC UNFREEZE

You can use Serena XML to unfreeze one or more “non-source” components in a package. This unfreeze request includes in its scope all “like-load”, “like-copybook”, and “like-PDS” component library types. “Like-source” components are excluded.

The Serena XML service/scope/message tags for a non-source component *unfreeze* message are:

```
<service name=”PACKAGE”>
<scope name=”NON_SRC”>
<message name=”UNFREEZE”>
```

These tags appear in both requests and replies.

PACKAGE NON_SRC UNFREEZE — Requests

Serena XML supports two types of unfreeze requests for non-source components:

- ***Full Unfreeze*** — Unfreezes all non-source components in the named package. This is the default.
- ***Selective Unfreeze*** — Unfreezes a subset of individually named non-source components in the named package. Desired components are itemized by name and library type in the `<component>` data element. A count of the itemized components to unfreeze is required in the `<listcount>` tag.

The `<request>` tag syntax for a non-source component unfreeze request is identical to that for a source-and-load component unfreeze request. (See *Exhibit 3-27*.) Only the `name` parameter in the high-level `<scope>` tag differs in this request, as shown above.

PACKAGE NON_SRC UNFREEZE — Replies

The Serena XML replies to a unfreeze request for non-source components do not return a `<result>` data structure. They do, however, return a standard `<response>` data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Refreeze Non-Source Components - PACKAGE NON_SRC REFREEZE

The inverse of the Serena XML unfreeze function for non-source components is the refreeze function for these members. Like its inverse, the refreeze function applies to one or more "non-source" components in a package, such as executable load modules, JCL procedures, and copybooks. "Like-source" components are excluded. Scratch and rename utility records are also outside the scope of this function.

The Serena XML service/scope/message tags for a non-source component *refreeze* message are:

```
<service name=”PACKAGE”>
<scope name=”NON_SRC”>
<message name=”REFREEZE”>
```

These tags appear in both requests and replies.

PACKAGE NON_SRC REFREEZE — Requests

Serena XML supports two types of refreeze requests for non-source components:

- ***Full Refreeze*** — Refreezes all non-source components in the named package. This is the default.
- ***Selective Refreeze*** — Refreezes a subset of individually named non-source components in the named package. Desired components are itemized by name and library type in the `<component>` data element. A count of the itemized components to refreeze is required in the `<listcount>` tag.

The `<request>` tag syntax for a non-source component refreeze request is identical to that for a non-source component unfreeze request. (See *Exhibit 3-27*.) Only the `name` parameter in the high-level `<message>` tag differs in this request, as shown above.

Refreeze Non-Source Components — Replies

The Serena XML reply to a package-level refreeze request for non-source components does not return a `<result>` data structure. It does, however, return a standard `<response>` data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

List Scratch and Rename Utility Records - CMPONENT PKG_UTIL LIST

Serena XML can list the scratch and rename utility requests for all components in a package. The service/scope/message tags for this *list* message are:

```
<service name="CMPONENT">
<scope name="PKG_UTIL">
<message name="LIST">
```

These tags appear in both requests and replies.

The service name is `"cmponent"`, not `"package"`, because XML Services calls the low-level component service in ChangeMan ZMF to perform most tasks associated with this function. The scope name, `"pkg_util"`, identifies this message as a package-level component service.

Note

The spelling of "cmponent" in the service name attribute is condensed to eight bytes for legacy compatibility on the mainframe.

CMPONENT PKG_UTIL LIST — Requests

Serena XML supports several uses for the scratch and rename request list. For example, using appropriate selection criteria in your request, you can:

- ***Find Old Component Name From New Component Name*** — Name the desired package in the `<package>` tag. Enter "8" in the `<utilityType>` tag to select rename records. Enter the known, new component name (after rename) in the `<newComponent>` tag. The function returns any rename records that match that new component name, together with the old component name prior to the rename action.
- ***Find New Component Name from Old Component Name*** — Name the desired package in the `<package>` tag. Enter "8" in the `<utilityType>` tag to select rename records. Enter the known, old component name (before rename) in the `<component>` tag. The function returns any rename records that match that old component name, together with the new component name after the rename action.
- ***List All Scratched and Renamed Components*** — Name the desired package in the `<package>` tag. Enter a blank in the `<utilityType>` tag or omit it entirely to request both scratch and rename record types. A list of all components with scratch and rename requests, including old and new component names, will be returned.
- ***List All Scratched Components*** — Name the desired package in the `<package>` tag. Enter "9" in the `<utilityType>` tag to request scratch records. The functions lists all components in the package with outstanding scratch requests.

To further customize your list request, specify a library type, modification date range, updater ID, or component status of interest. Choose component status options using appropriate yes/no flag tags.

Note

Yes/no flags for component status filtering take default values as a group. The default changes based on whether or not you enter explicit values in these tags, as follows:

- If ***no*** status flag has an explicitly typed value, the default for all tags is "Y".
- If ***any*** status flag has an explicitly typed value, the default for the remaining tags is "N".

The following example shows how you might code a request to list all renamed components in a package using Serena XML. The example request includes only components that were renamed while in unfrozen status; active, inactive, or frozen components are omitted.

Data structure details for the `<request>` tag appear in *Exhibit 3-29*.

Example XML — CMPONENT PKG_UTIL LIST Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="PKG_UTIL">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>TES5000001</package>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-29. CMPONENT PKG_UTIL LIST <request>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.

Exhibit 3-29. CMPONENT PKG_UTIL LIST <request> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<component>	Optional	0 - 1	String (256), variable	Original component name before scratch or rename operation. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root. ***NOTE:*** Takes asterisk (*) wildcard.
<componentType>	Optional	0 - 1	String (3), fixed	Library type of component in `<componentName>`.
<filterActiveStatus>	Optional	0 - 1	String (1)	**Y** = Include active components **N** = Omit active components ***NOTE:*** Part of component status flag group. If *no* tag in group has explicit value, default is Y. If *any* tag in group has explicit value, default is N.
<filterHfsDirectory>	Optional	0 - 1	String (256), variable	Name of HFS directory containing components to be listed, prefixed by path from installation root (that is, path as stored in baseline library). If present, only files in this directory are listed. If absent, all HFS files meeting other criteria are listed. ***NOTE:*** Applies to z/OS Unix HFS components only. Irrelevant for native z/OS PDS library members.
<filterInactiveStatus>	Optional	0 - 1	String (1)	**Y** = Include inactive components **N** = Omit inactive components ***NOTE:*** Part of component status flag group. If *no* tag in group has explicit value, default is Y. If *any* tag in group has explicit value, default is N.
<filterUnfrozenStatus>	Optional	0 - 1	String (1)	**Y** = Include unfrozen components **N** = Omit unfrozen components ***NOTE:*** Part of component status flag group. If *no* tag in group has explicit value, default is Y. If *any* tag in group has explicit value, default is N.
<fromDateLastModified>	Optional	0 -1	Date, yyyymmdd	Start date in desired range of component modification dates.

Exhibit 3-29. CMPONENT PKG_UTIL LIST <request> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<newComponent>	Optional	0 -1	String (256), variable	New component name after rename operation. Blank for scratch operation. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root. ***NOTE:*** Takes asterisk (*) wildcard.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<toDateLastModified>	Optional	0 -1	Date, yyyymmdd	End date in desired range of component modification dates.
<updater>	Optional	0 -1	String (8), variable	TSO user ID of last person to update component.
<utilityType>	Optional	0 - 1	String (1)	Selects type of utility record to list. Values: **8** = Rename record **9** = Scratch record **Blank** = Both record types ***NOTE:*** Omit tag or enter explicit blank to list both record types. Null tag returns no records. ***NOTE:*** Asterisk (*) wildcard is not accepted in this tag.

CMPONENT PKG_UTIL LIST — Replies

The Serena XML reply to this request returns zero to many `<result>` data structures, each of which lists one component scratch or rename utility record for a package. Scratch records report the names of components awaiting deletion, along with status information. Rename records report old and new component names, along with status information for the original component at the time it was renamed.

The reply message returns a standard `<response>` data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because the `<response>` tag follows the last `<result>` tag, it also serves as an end-of-list marker.

The example below shows how a reply for this function might appear in Serena XML. Data structure details for the `<result>` tag appear in *Exhibit 3-30*.

Example XML — CMPONENT PKG_UTIL LIST Reply

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="PKG_UTIL">
  <message name="LIST">
   <result>
    <utilityType>9</utilityType>
    <package>TES5000001</package>
    <applName>TES5</applName>
    <packageId>000001</packageId>
    <componentType>CPY</componentType>
    <updater>USER24</updater>
    <dateLastModified>20090205</dateLastModified>
    <timeLastModified>112910</timeLastModified>
    <component>ACPCPY00</component>
    <componentStatus>0</componentStatus>
    <encryption>00000000</encryption>
   </result>
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-30. CMPONENT PKG_UTIL LIST <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<component>	Optional	1	String (256), variable	Original ZMF name of scratched or renamed component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.

Exhibit 3-30. CMPONENT PKG_UTIL LIST <result> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<componentStatus>	Optional	0 -1	String (1)	Code for original component status. Values: **0** = Active **1** = Approved **2** = Checked Out **3** = Demoted **4** = Frozen **5** = Inactive **6** = Incomplete **7** = Promoted **8** = Refrozen **9** = Rejected **A** = Remote Promoted **B** = Submitted **C** = Unfrozen
<componentType>	Optional	1	String (3), variable	Library type of scratched or renamed component.
<dateLastModified>	Optional	0 -1	Date, yyyymmdd	Date original component was last modified.
<encryption>	Optional	0 - 1	String (8), variable	Component encryption number
<newComponent>	Optional	0 -1	String (256), variable	New ZMF name of renamed component.
<package>	Optional	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.
<timeLastModified>	Optional	0 -1	Time, hhmmss	Time original component was last modified, 24-hr format.
<updater>	Optional	0 -1	String (8), variable	TSO user ID of last person to update original component.
<utilityType>	Optional	1	String (1)	Code for type of component utility record listed. Values: **8** = Rename record **9** = Scratch record

Unfreeze Scratch/Rename Records - PACKAGE SCR_REN UNFREEZE

Unfreeze requests for scratch and rename utility records (the so-called IUTL records) selectively unlock these records so you can scratch and rename package components without otherwise modifying component contents. An audit trail of such actions is maintained in the IUTL records for later listing or impact analysis.

The Serena XML service/scope/message tags for a scratch and rename utility records *unfreeze* message are:

```
<service name="PACKAGE">
<scope name="SCR_REN">
<message name="UNFREEZE">
```

These tags appear in both requests and replies.

PACKAGE SCR_REN UNFREEZE — Requests

Serena XML supports two types of unfreeze requests for scratch and rename utility records:

- ***Full Unfreeze*** — Unfreezes scratch and rename utility records for all components in the named package. This is the default.
- ***Selective Unfreeze*** — Unfreezes a subset of scratch and rename records for individually named components in the named package. Desired components are itemized by name and library type in the `<component>` data element. A count of the components to be unfrozen for scratch or rename purposes is required in the `<listcount>` tag.

The `<request>` tag syntax for a scratch and rename unfreeze request is identical to that for other component unfreeze requests. (See *Exhibit 3-27*.) Only the `name` parameter in the high-level `<scope>` and `<message>` tags differ, as shown above.

PACKAGE SCR_REN UNFREEZE — Replies

The Serena XML reply to a scratch and rename unfreeze request does not return a `<result>` data structure. It does, however, return a standard `<response>` data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Refreeze Scratch/Rename Records - PACKAGE SCR_REN REFREEZE

The inverse of the Serena XML unfreeze function for scratch and rename records is the refreeze function for these records. The refreeze function returns the scratch and rename utility functions to their previously locked-down condition for frozen package components in ChangeMan ZMF.

The Serena XML service/scope/message tags for a scratch and rename record *refreeze* message are:

```
<service name="PACKAGE">
<scope name="SCR_REN">
<message name="REFREEZE">
```

These tags appear in both requests and replies.

PACKAGE SCR_REN REFREEZE — Requests

Serena XML supports two types of refreeze requests for scratch and rename utility records:

- ***Full Refreeze*** — Refreezes scratch and rename records for all components in the named package. This is the default.
- ***Selective Unfreeze/Refreeze*** — Refreezes scratch and rename records for a subset of individually named components in the named package. Desired components are itemized by name and library type in the `<component>` data element. A count of the components to be refrozen is required in the `<listcount>` tag.

The `<request>` tag syntax for a scratch and rename refreeze request is identical to that for other component refreeze requests. (See *Exhibit 3-27*.) Only the `name` parameter in the high-level `<scope>` and `<message>` tags differ, as shown above.

PACKAGE SCR_REN REFREEZE — Replies

The Serena XML reply to a scratch and rename utility records refreeze request does not return a `<result>` data structure. It does, however, return a standard `<response>` data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

PACKAGE VALIDATION TASKS

Package validation tasks identify dependencies among package components, verify the integrity of package components, or check for versioning differences and out-of-synch errors across components in different stages of development. Typical commands include *list*, *check*, and *audit*.

Serena XML supports the following package validation tasks:

- *List Source-to-Load Dependencies - CMPONENT PKG_LOD LIST*
- *Check Component Integrity - PACKAGE CMPONENT INTEGRTY*
- *Audit a Package - PACKAGE SERVICE AUDIT*

List Source-to-Load Dependencies - CMPONENT PKG_LOD LIST

Source-to-load relationship (ILOD) records track dependencies between “like-source” components and any “like-load” or “like-other” components generated from a source component by compilation, DB2 precompile, or similar transformation process. These are the records returned by the Serena XML function listing source-to-load dependencies.

Not included in these source-to-load dependency records are “like-load” components staged to development, but not generated from a “like-source” component while in staging. Also omitted from the scope of this function are source-to-include relationships involving copybooks, macros, subroutines, skeletons, ISPF panels, or JCL procedures.

The Serena XML service/scope/message tags for a source-to-load dependency list are:

```
<service name="CMPONENT">
<scope name="PKG_LOD">
<message name="LIST">
```

These tags appear in both request and reply messages.

The service name is `"cmponent"`, not `"package"`, because XML Services calls the low-level component service in ChangeMan ZMF to perform most tasks associated with this function. The scope name, `"pkg_lod"`, identifies this message as a package-level component service.

Note

The spelling of "cmponent" in the service name attribute is condensed to eight bytes for legacy compatibility on the mainframe.

CMPONENT PKG_LOD LIST — Requests

Serena XML supports the following options for source-to-load dependency lists:

- ***All Dependencies in Package*** — Name the desired package in the `<package>` package tag. The `<component>` and `<targetComponent>` tags should be omitted from the request. The function returns all source-to-load dependencies involving "like-load" components generated during the life of the named package.
- ***Load Dependencies for a Source Component*** — Name the desired package in the `<package>` tag and the desired "like-source" component in the `<component>` tag. Include the library type for the like-source component in `<componentType>`. Omit the `<targetComponent>` tag from the request. The function lists all "like-load" components affiliated with the named like-source component.
- ***Source Dependencies for a Load Component*** — Name the desired package in the `<package>` tag and the desired "like-load" component in the `<targetComponent>` tag. Include the library type for the like-load component in `<targetComponentType>`. Omit the `<component>` tag from the request. The function lists all "like-source" components affiliated with the named like-load component.

To further customize your request, specify a library type, modification date range, updater ID, or component status of interest. Choose component status options using appropriate yes/no flag tags.

Note

Yes/no flags for component status filtering take default values as a group. The default changes based on whether or not you enter explicit values in these tags, as follows:

- If ***no*** status flag has an explicitly typed value, the default for all tags is "Y".
- If ***any*** status flag has an explicitly typed value, the default for the remaining tags is "N".

The following example shows how you might code a request to list all “like-load” dependencies for a given “like-source” component in a package.

Example XML — CMPONENT PKG_LOD LIST Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="PKG_LOD">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>CISQ000030</package>
   </request>
  </message>
 </scope>
</service>
```

Data structure details for the `<request>` tag appear in *Exhibit 3-31*.

Exhibit 3-31. CMPONENT PKG_LOD LIST <request>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended to replace `<package>`. Use `<package>` instead of `<applName>` & `<packageId>`.
<component>	Optional	0 - 1	String (256), variable	Name of “like-source” component for which “like-load” dependencies are wanted. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentType>	Optional	0 - 1	String (3), variable	Library type of “like-source” component for which “like-load” dependencies are wanted. ***NOTE:*** Omit to include all.

Exhibit 3-31. CMPONENT PKG_LOD LIST <request> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<filterActiveStatus>	Optional	0 - 1	String (1)	**Y** = Include active components **N** = Omit active components ***NOTE:*** Part of component status flag group. If *no* tag in group has explicit value, default is Y. If *any* tag in group has explicit value, default is N.
<filterFrozenStatus>	Optional	0 - 1	String (1)	**Y** = Include frozen components **N** = Omit frozen components ***NOTE:*** Part of component status flag group. If *no* tag in group has explicit value, default is Y. If *any* tag in group has explicit value, default is N.
<filterHfsDirectory>	Optional	0 - 1	String (256), variable	Name of HFS directory containing components to be listed, prefixed by path from installation root (that is, path as stored in baseline library). If present, only files in this directory are listed. If absent, all HFS files meeting other criteria are listed. ***NOTE:*** Applies to z/OS Unix HFS components only. Irrelevant for native z/OS PDS library members.
<filterUnfrozenStatus>	Optional	0 - 1	String (1)	**Y** = Include unfrozen components **N** = Omit unfrozen components ***NOTE:*** Part of component status flag group. If *no* tag in group has explicit value, default is Y. If *any* tag in group has explicit value, default is N.
<fromdateLastModified>	Optional	0 -1	Date, yyyymmdd	Start date in desired range of component modification dates.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.

Exhibit 3-31. CMPONENT PKG_LOD LIST <request> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<targetComponent>	Optional	0 - 1	String (256), variable	Name of “like-load” component for which “like-source” dependencies are wanted. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root. ***NOTE:*** Omit to include all.
<targetComponentType>	Optional	0 - 1	String (3), variable	Library type of “like-load” component.
<toDateLastModified>	Optional	0 -1	Date, yyyymmdd	End date in desired range of component modification dates.
<updater>	Optional	0 -1	String (8), variable	TSO user ID of last person to update component.

CMPONENT PKG_LOD LIST — Replies

The Serena XML source-to-load dependency list returns zero to many `<result>` data structures, each of which lists one source-to-load relationship for a package. Status information for the “like-source” component in the pair is also returned. If no compilations, submissions for JCL install job build, or other transformation processes have occurred for components in the package, no `<result>` tags are returned.

The reply message returns a standard `<response>` data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because the `<response>` tag follows the last `<result>` tag, it also serves as an end-of-list marker.

The example below shows how a reply for this function might appear in Serena XML. Data structure details for the `<result>` tag appear in *Exhibit 3-32*.

Example XML — CMPONENT PKG_LOD LIST Reply

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="PKG_LOD">
  <message name="LIST">
   <result>
    <package>CISQ000030</package>
    <applName>CISQ</applName>
    <packageId>000030</packageId>
    <component>APPSTATS</component>
    <componentType>SRS</componentType>
    <targetComponent>APPSTATS</targetComponent>
    <targetComponentType>DBR</targetComponentType>
    <updater>USER24</updater>
    <dateLastModified>20081126</dateLastModified>
    <timeLastModified>094711</timeLastModified>
    <setssi>5BFD1269</setssi>
    <componentStatus>4</componentStatus>
    <rebuildFromBaseline>N</rebuildFromBaseline>
   </result>
   <result>
    <package>CISQ000030</package>
    <applName>CISQ</applName>
    <packageId>000030</packageId>
    <component>APPSTATS</component>
    <componentType>SRS</componentType>
    <targetComponent>APPSTATS</targetComponent>
    <targetComponentType>LOS</targetComponentType>
    <updater>USER24</updater>
    <dateLastModified>20081126</dateLastModified>
    <timeLastModified>094711</timeLastModified>
    <setssi>5BFD1269</setssi>
    <componentStatus>4</componentStatus>
    <rebuildFromBaseline>N</rebuildFromBaseline>
   </result>
.
.
.
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-32. CMPONENT PKG_LOD LIST <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), fixed	ZMF application name. Same as first 4 bytes of package name.
<component>	Optional	0 - 1	String (256), variable	Name of "like-source" component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.

Exhibit 3-32. CMPONENT PKG_LOD LIST <result> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<componentStatus>	Optional	0 -1	String (1)	Code for original component status. Values: **0** = Active **1** = Approved **2** = Checked Out **3** = Demoted **4** = Frozen **5** = Inactive **6** = Incomplete **7** = Promoted **8** = Refrozen **9** = Rejected **A** = Remote Promoted **B** = Submitted **C** = Unfrozen
<componentType>	Optional	0 - 1	String (3), variable	Library type of “like-source” component.
<dateLastModified>	Optional	0 -1	Date, yyyymmdd	Date source component was last modified.
<hashToken>	Optional	0 -1	String (16), variable	Hash token for HFS.
<package>	Optional	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.
<rebuildFromBaseline>	Optional	0 -1	String (1)	**Y** = Recompile/relink from baseline, not package component **N** = Recompile/relink from package component, not baseline
<setssi>	Optional	0 -1	String (8), fixed	Staged component SETSSI date (seconds since 1/1/1960).
<targetComponent>	Optional	0 - 1	String (256), variable	Name of “like-load” component generated from “like-source” component.
<targetComponentType>	Optional	0 - 1	String (3), variable	Library type of “like-load” component.
<timeLastModified>	Optional	0 -1	Time, hhmmss	Time source component was last modified, 24-hr format.
<updater>	Optional	0 -1	String (8), variable	TSO user ID of last person to update source component.

Check Component Integrity - PACKAGE CMPONENT INTEGRTY

The package-level component integrity check verifies that all package component records in the package master database have corresponding physical components in the staging libraries, and vice versa. A component integrity check can be made only against simple or participating packages.

Tip

To verify that all independently queued batch jobs have completed in ChangeMan ZMF before a dependent job step executes, use the Serena XML component integrity check.

The Serena XML service/scope/message tags for a package-level *component integrity check* message are:

```
<service name="PACKAGE">
<scope name="CMPONENT">
<message name="INTEGRTY">
```

These tags appear in both request and reply messages.

PACKAGE CMPONENT INTEGRTY Requests

The following example shows how you might code a Serena XML request for a package-level component integrity check.

Example XML — PACKAGE CMPONENT INTEGRTY Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="CMPONENT">
  <message name="INTEGRTY">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>IMSQ000012</package>
   </request>
  </message>
 </scope>
</service>
```

Data structure details for the package cmponent integrity check `<request>` tag appear in *Exhibit 3-33*.

Exhibit 3-33. PACKAGE CMPONENT INTEGRTY <request>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended to replace `<package>`. Use `<package>` instead of `<applName>` & `<packageId>`.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<processLstStagingLibs>	Optional	0 -1	String (1)	Option to include listings (library type LST) in check. Values: **Y** = Yes, include listing libraries **N** = No, omit listing libraries

PACKAGE CMPONENT INTEGRTY Replies

The Serena XML reply message to a package-level request to check component integrity returns zero to many `<result>` tags. Each `<result>` tag contains information about a package component that failed the component integrity check. If all components pass the check, no `<result>` tag is returned.

A standard `<response>` data structure always follows the `<result>` tags, if any, to indicate the overall success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

An example of a Serena XML reply to a component integrity check reply message follows. In this example, multiple package components fail the integrity check. No other errors occur. Data structure details for the component integrity check `<result>` appear in *Exhibit 3-34*.

Note that the `<component>` subtag of the component integrity check `<result>` tag represents a simple data element containing the component name only. It differs from the `<component>` tag used in the selective package promote and demote messages, in selective unfreeze and refreeze messages, and others where `<component>` is a complex data element containing subtags of its own. Serena XML distinguishes the two by service/scope/message context.

Example XML — PACKAGE CMPONENT INTEGRTY Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="CMPONENT">
  <message name="INTEGRTY">
   <response>
    <statusMessage>CMN8700I - Package Integrity service completed</
statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-34. PACKAGE CMPONENT INTEGRTY <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<componentName>	Optional	0 - 1	String (256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally pre-fixed by path from installation root.
<componentType>	Optional	0 - 1	String (3), variable	Must be valid ZMF library type.
<message>	Optional	0 - 1	String (255), variable	Component integrity error message for named component.

Audit a Package - PACKAGE SERVICE AUDIT

The package audit function checks for out-of-synch conditions in package components. The package(s) to be audited must be in development status, with all components at promotion level 00, prior to running the audit request. The outcome of the audit is an audit report, which the package audit function spools to your system's output utility (e.g., Spool Display and Search Facility, or SDSF) for later interactive access and printing.

The Serena XML service/scope/message tags for a package audit message are:

```
<service name=”PACKAGE”>
<scope name=”SERVICE”>
<message name=”AUDIT”>
```

These tags appear in both requests and replies.

PACKAGE SERVICE AUDIT Requests

Serena XML supports package audit options of narrow to broad scope. A set of yes/no audit flag tags lets you tailor the scope of your audit to the following:

- ***Staging Libraries Only, No Cross-Package Dependencies*** — Performs an audit against staging libraries but not baseline libraries for a simple package. To select this option, set the `<auditLite>` tag flag to “Y”.
- ***Staging and Baseline Libraries, No Cross-Package Dependencies*** — Performs an audit against both staging and baseline libraries for a simple package. To select this option, set the `<auditLite>` tag flag to “N”.
- ***Staging Libraries Only, Check Cross-Package Dependencies*** — Performs an audit against staging libraries for non-baselined participating packages in a complex package. To select this option for a participating package, set the `<auditPartAsPrimary>` tag flag to “Y”. To select this option for a complex package, set the `<auditLite>` tag flag to “Y”.
- ***Staging and Baseline Libraries, Check Cross-Package Dependencies*** — Performs an audit against staging libraries and baseline libraries for all participating packages in a complex package. To select this option, enter the name of the complex package in the `<package>` tag and set the `<auditLite>` tag flag to “N”.
- ***Cross-Application Audit*** — Performs an audit against both staging and baseline libraries for all applications identified in the request message. Cross-application dependencies are audited across applications defined in the package master for complex packages and participating packages not otherwise excluded from this option. You can also request that cross-application dependencies be checked for the applications you name. To do this, list the desired applications using the repeating `<scopeAppl>` tag and provide the number of those applications in `<listCount>`.

The following example shows how you might code a package audit request using Serena XML. Data structure details for the package audit `<request>` tag appear in *Exhibit 3-35*.

Example XML — PACKAGE SERVICE AUDIT Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="SERVICE">
  <message name="AUDIT">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000013</package>
    <jobCards01>//XMLX126  JOB (AMW,000),'DEFINE UCAT',MSGCLASS=Y,</jobCards01>
    <jobCards02>//             TIME=(,10),NOTIFY=USER24</jobCards02>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-35. PACKAGE SERVICE AUDIT <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<auditAutoParms>	Optional	0 - 1	String (44), variable	The name of a data set containing the auto resolve parameters. See the ChangeMan ZMF User's Guide, Using Auto Resolve.
<auditAutoResolve>	Optional	0 - 1	String (1)	Option to automatically resolve out-of-synch conditions. Values: **Y** = Yes, turn on auto-resolve **N** = No, don't auto-resolve errors
<auditFormatReport>	Optional	0 - 1	String (1)	Option to include printer control characters in output file. Values: **Y** = Yes, include control chars **N** = No, don't use control chars
<auditIncludeHistory>	Optional	0 - 1	String (1)	Option to include component history in audit report. Values: **Y** = Yes, include history **N** = No, omit history
<auditLite>	Optional	0 - 1	String (1)	Option to omit baseline libraries from audit. Values: **Y** = Yes, omit baseline libraries **N** = No, don't omit baseline libs
<auditPartAsPrimary>	Optional	0 - 1	String (1)	Option to include cross-package dependencies for all but previously installed packages from audit of participating package. Values: **Y** = Yes, audit as primary pkg **N** = No, don't change audit scope
<auditPartAsSimple>	Optional	0 - 1	String (1)	Option to omit cross-package dependencies from audit of a participating package & follow rules for simple packages concerning baseline libraries. Values: **Y** = Yes, audit as simple package **N** = No, don't change audit scope

Exhibit 3-35. PACKAGE SERVICE AUDIT <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<auditPartByDept>	Optional	0 - 1	String (1)	Option to limit cross-package dependency check to department of target package. Values: **Y** = Yes, audit within department **N** = No, don't change audit scope
<auditRCUpdateRestrictToTarget>	Optional	0 - 1	String (1)	Option for cross-package & cross-application audits to update audit return code only in the package named in `<package>` tag, rather than all packages audited. Values: **Y** = Yes, restrict audit return code **N** = No, don't restrict return code
<auditTraceOption>	Optional	0 - 1	String (1)	Option to turn on trace option for audit function. Values: **Y** = Yes, turn on trace option **N** = No, don't trace audit job
<includeXAPheaders>	Optional	0 - 1	String (1)	Include cross application common baseline headers: **Y** = Yes **N** = No
<jobCard01>	Optional	0 - 1	String (72), fixed length	First of up to 4 JCL statements needed to execute the audit in batch mode.
<jobCard02>	Optional	0 - 1	String (72), fixed length	Second of up to 4 JCL statements needed to execute the audit in batch mode.
<jobCard03>	Optional	0 - 1	String (72), fixed length	Third of up to 4 JCL statements needed to execute the audit in batch mode.
<jobCard04>	Optional	0 - 1	String (72), fixed length	Fourth of up to 4 JCL statements needed to execute the audit in batch mode.
<listCount>	Optional	0 - 1	Integer (3), variable	Count of `<scopeAppl>` tags to include in cross-application audit. ***NOTE:*** Required if value is Y in `<auditCrossApplication>` tag.
<lockPackage>	Optional	0 - 1	String (1)	Lock package during audit.: **Y** = Yes **N** = No
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name. ***NOTE:*** May be simple, complex, or participating.

Exhibit 3-35. PACKAGE SERVICE AUDIT <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<scopeAppl>	Optional	0 - ∞	Complex	Name of application to include in scope of cross-application audit. Repeatable to accommodate multiple applications. ***NOTE:*** Required if value is Y in `<auditCrossApplication>` tag. ***NOTE:*** If used, `<listCount>` tag also required.
<suppressNotify>	Optional	0 - 1	String (1)	Suppress Batch Messages: **Y** = Yes **N** = No
<userVariable01> . . . <userVariable05>	Optional	0 - 1 each	String (8), variable	Up to five user-defined variables of 8 bytes each, used to pass parameters to JCL interpreter.
<userVariable06> . . . <userVariable10>	Optional	0 - 1 each	String (72), variable	Up to five user-defined variables of 72 bytes each, used to pass parameters to JCL interpreter.

Tip

Tags: <userVariable01> to <userVariable10>: See topic “Custom V01-V10 Variables” in the ChangeMan ZMF Customization Guide.

PACKAGE SERVICE AUDIT Replies

Because the ZMF audit report is spooled to SDSF, Serena XML replies to a package audit request do not return a `<result>` data structure. They do, however, return a standard `<response>` data structure to indicate the success or failure of the audit request.

Example XML — PACKAGE SERVICE AUDIT Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="SERVICE">
  <message name="AUDIT">
   <response>
    <statusMessage>CMN2600I - The job to audit this package has been
submitted.</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>2600</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

A job with output similar to the following is submitted:

```
//*  JOB TO AUDIT PACKAGE ACTP000013
                                                Legend and Summary Report
  The local level of audit chosen at this point;      0
      0 - Audit is recommended but entirely optional
  Out-of-synch messages (hint - search for "!" marks)
      DUPLIC! (Staging duplicates baseline)      ===> 9
      SYNCH0! (Not in scope of audit or unknown) ===> 1
      SYNCH1! (ISPF statistics not available)    ===> 0
      SYNCH2! (Compile/designated proc differ)   ===> 0
      SYNCH3! (Unparsable load module)           ===> 0
      SYNCH4! (CPY problem in staging)           ===> 0
      SYNCH5! (CPY high-date problem in baseline)===> 17
      SYNCH6! (Activity file not checked out)    ===> 0
      SYNCH7! (Called subroutine in staging)     ===> 0
      SYNCH8! (Called subroutine in baseline)    ===> 0
      SYNCH9! (Source and load discrepancy)      ===> 0
      SYNCH10! (Version regression problem)      ===> 0
      SYNCH11! (Component hash discrepancy)      ===> 0
      SYNCH12! (Orphan module in staging)        ===> 0
      SYNCH13! (Baseline/staging discrepancy)    ===> 0
      SYNCH14! (Components not in active status) ===> 0
      SYNCH15! (Source to relationship problem)  ===> 0
      SYNCH16! (CPY low-date problem in baseline)===> 0
      SYNCH17! (CPY deleted problem in staging)  ===> 0
      SYNCH18! (LOD deleted problem in staging)  ===> 0

  Highest return code encountered                ===> 8
  CMN2678I - AUDIT RETURN CODE NOT UPDATED.
```

Serena XML audit return codes are the same as those for audits requested through the ISPF interface. Successful audits — i.e. those that find no out-of-synch conditions — have a return code of 00. Unsuccessful audit requests have a return code of 04 or higher.

PACKAGE INFORMATION MANAGEMENT TASKS

Package information management tasks retrieve or manage metadata and control information for a package. Such information includes title and descriptions, general parameters, user-defined package variables, install information, promotion history for the package, promotion history for package components, and the like. Typical commands include *list*, *unfreeze*, and *refreeze*.

Serena XML supports the follow information management tasks for packages:

- *List Package Description - PACKAGE GEN_DESC LIST*
- *List General Package Parameters - PACKAGE GEN_PRMS LIST*
- *Unfreeze Package Parameters - PACKAGE GEN_PRMS UNFREEZE*
- *Refreeze Package Parameters - PACKAGE GEN_PRMS REFREEZE*
- *List User-Defined Package Variables - PACKAGE USR_RECS LIST*
- *List Package Install Sites - SITE PKG LIST*
- *Unfreeze Package Install Sites - PACKAGE SITES UNFREEZE*
- *Refreeze Package Install Sites - PACKAGE SITES REFREEZE*
- *List Package Installation Dependencies - PACKAGE SCH_RECS LIST*
- *List Package Implementation Instructions - PACKAGE IMP_INST LIST*
- *List Package Approvers - APPROVER PKG LIST*
- *List Affected Applications - PACKAGE AFF_APLS LIST*
- *List Participating Packages - PACKAGE PRT_PKGS LIST*
- *List Linked Packages - PACKAGE PKG_LINK LIST*
- *List Package Library Types - LIBTYPE PKG LIST*
- *List Package Promotion History - PACKAGE PRM_HIST LIST*
- *Package Promoted Component List - PACKAGE PRM_CMP LIST*
- *List Reasons for Backout or Revert - PACKAGE REASONS LIST*

List Package Description - PACKAGE GEN_DESC LIST

Serena XML lists the package description for one package. Multiple package descriptions require multiple requests. Descriptions for baselined packages are accessible as long as the package master record remains in the package database.

The Serena XML service/scope/message names for a package description *list* message are:

```
<service name=”PACKAGE”>
<scope name=”GEN_DESC”>
<message name=”LIST”>
```

These tags appear in both request and reply messages.

PACKAGE GEN_DESC LIST — Requests

The package description list requires a package name as input. Wildcard characters are not accepted in the `<package>` tag.

The following example shows how you might code a package description list request in Serena XML. Data structure details for the `<request>` tag appear in the following exhibit. This data structure is common to many package requests in Serena XML.

Example XML — PACKAGE GEN_DESC LIST Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="GEN_DESC">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000007</package>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-36.PACKAGE GEN_DESC LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), fixed	ZMF application name. Same as first 4 bytes of package name.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.

PACKAGE GEN_DESC LIST — Replies

Replies to a package description list request return no more than one `<result>` tag. The `<result>` contains zero to many package description entries for a single package.

The result is followed by a standard `<response>` tag that indicates the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

An example XML reply message for the package description list follows. Data structure details for `<result>` tag appear after the example, in *Exhibit 3-37*.

Example XML — PACKAGE GEN_DESC LIST Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="GEN_DESC">
  <message name="LIST">
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <packageDesc>SER5904E</packageDesc>
   </result>
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-37. PACKAGE GEN_DESC LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), fixed	ZMF application name. Same as first 4 bytes of package name.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageDesc>	Optional	0 - 46	String (72), variable	General description of package contents. Up to 46 lines of 72 bytes each are allowed by ZMF.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.

List General Package Parameters - PACKAGE GEN_PRMS LIST

General parameters for a package include descriptive items entered during package creation and update, as well as programmatically maintained status, audit trail, and release information. The Serena XML function to list general package parameters retrieves this information for any package master record, even after a package has been baselined.

User-defined package variables, package install information, and package approver lists require other Serena XML functions for access.

The Serena XML service/scope/message names for a request to list general package parameters are:

```
<service name="PACKAGE">
<scope name="GEN_PRMS">
<message name="LIST">
```

These tags appear in both request and reply messages.

PACKAGE GEN_PRMS LIST — Request

The syntax for a request to list general package parameters is identical to that for many package information management functions, including the package description list.

PACKAGE GEN_PRMS LIST — Replies

The Serena XML reply to a general package parameters list request contains one `<result>` tag for the package named in the request.

A standard `<response>` tag follows the `<result>` to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

An example reply message listing general package parameters appears below. Data structure details for the `<result>` tag appear after the example in *Exhibit 3-38*.

Example XML — PACKAGE GEN_PRMS LIST Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="GEN_PRMS">
  <message name="LIST">
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <packageLevel>1</packageLevel>
    <packageType>1</packageType>
    <packageStatus>3</packageStatus>
    <dateCreated>20090127</dateCreated>
    <timeCreated>082516</timeCreated>
    <dateFrozen>20090127</dateFrozen>
    <timeFrozen>082824</timeFrozen>
    <dateApproved>20090127</dateApproved>
    <timeApproved>083154</timeApproved>
    <dateSent>20090127</dateSent>
    <timeSent>083210</timeSent>
    <dateReceived>20090127</dateReceived>
    <timeReceived>083238</timeReceived>
    <dateInstalled>20090127</dateInstalled>
    <timeInstalled>083537</timeInstalled>
    <dateBaselined>20090127</dateBaselined>
```

```
     <timeBaselined>083656</timeBaselined>
     <requestorDept>IDD</requestorDept>
     <requestorName>USER24</requestorName>
     <requestorPhone>5555555</requestorPhone>
     <workChangeRequest>USER24</workChangeRequest>
     <packageTitle>SER5906E</packageTitle>
     <creator>USER24</creator>
     <lastPromotionAction>0</lastPromotionAction>
     <schedulerType>2</schedulerType>
     <isPostApprovalPending>N</isPostApprovalPending>
     <isPostApproversAdded>N</isPostApproversAdded>
     <isPostRejected>N</isPostRejected>
     <isShortApproverListUsed>N</isShortApproverListUsed>
     <tempChangeDuration>000</tempChangeDuration>
     <isStageLibsDeleted>N</isStageLibsDeleted>
     <isLinkedPackage>N</isLinkedPackage>
     <isCmnSchedulerUsed>N</isCmnSchedulerUsed>
     <isManualSchedulerUsed>Y</isManualSchedulerUsed>
     <isOtherSchedulerUsed>N</isOtherSchedulerUsed>
     <isAuditPending>N</isAuditPending>
     <isFreezePending>N</isFreezePending>
     <isApprovalPending>N</isApprovalPending>
     <isXNodeBuildRequired>N</isXNodeBuildRequired>
     <isInstallPending>Y</isInstallPending>
     <isRevertPending>N</isRevertPending>
     <isReverseRippleSubmitted>N</isReverseRippleSubmitted>
     <isBackedOut>N</isBackedOut>
     <isXNodeBuildPending>N</isXNodeBuildPending>
     <generalComponentStatus>4</generalComponentStatus>
     <nonSourceComponentStatus>4</nonSourceComponentStatus>
     <sourceLoadComponentStatus>4</sourceLoadComponentStatus>
     <utilityInfoStatus>4</utilityInfoStatus>
     <siteInfoStatus>4</siteInfoStatus>
     <customComponentStatus>4</customComponentStatus>
     <nearestInstallDate>20090127</nearestInstallDate>
     <problemActionCode>2</problemActionCode>
     <stageDevLibModel>CMNTP.SERT8.DEV.ACTP.#000007</stageDevLibModel>
     <stageProdLibModel>CMNTP.SERT8.PRD.ACTP.#000007</stageProdLibModel>
     <stageLibStatus>2</stageLibStatus>
     <installTimeExpiration>0200</installTimeExpiration>
     <packageCheckedIntoRelease>N</packageCheckedIntoRelease>
    </result>
    <response>
     <statusMessage>CMN8700I - LIST     Package service completed</statusMessage>
     <statusReturnCode>00</statusReturnCode>
     <statusReasonCode>8700</statusReasonCode>
    </response>
   </message>
  </scope>
 </service>
```

Exhibit 3-38. PACKAGE GEN_PRMS LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), fixed	ZMF application name. Same as first 4 bytes of package name.
<auditLockUserid>	Optional	0 - 1	String (8), variable	Userid who locked package for audit.
<auditReturnCode>	Optional	0 - 1	String (2), variable	Return code issued by ZMF package audit. Values: **00** = No major errors found **04** = Errors found **08** = Major errors found **12** = Possibly fatal errors found
<complexSuperPackage>	Optional	0 -1	String (10), fixed	Name of complex/super package to which this participating package belongs. ***NOTE:*** Returned only if value of `<packageLevel>` = 4.
<complexSuperPackageAppl>	Optional	0 -1	String (4), variable	Package application. ***NOTE:*** Returned only if value of `<packageLevel>` = 4.
<complexSuperPackageNumber>	Optional	0 -1	String (6), fixed	Package number. ***NOTE:*** Returned only if value of `<packageLevel>` = 4.
<creator>	Optional	0 -1	String (8), variable	TSO user ID of package creator.
<customComponentStatus>	Optional	0 - 1	String (1)	Status code for custom components of package as a group. Values: **0** = Active **1** = Approved **2** = Checked out **3** = Demoted **4** = Frozen **5** = Inactive **6** = Incomplete **7** = Promoted **8** = Refrozen **9** = Rejected **A** = Remote promoted **B** = Submitted **C** = Unfrozen
<dateApproved>	Optional	0 -1	Date (8), yyyymmdd	Date package approved.
<dateBaselined>	Optional	0 -1	Date (8), yyyymmdd	Date package baselined.

Exhibit 3-38. PACKAGE GEN_PRMS LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<dateBackedOut>	Optional	0 -1	Date (8), yyyymmdd	Date package backed out.
<dateCreated>	Optional	0 -1	Date (8), yyyymmdd	Date package created.
<dateFrozen>	Optional	0 -1	Date (8), yyyymmdd	Date package frozen.
<dateInstalled>	Optional	0 -1	Date (8), yyyymmdd	Date package installed.
<dateMemoDeleted>	Optional	0 -1	Date (8), yyyymmdd	Date package logically deleted.
<dateReceived>	Optional	0 -1	Date (8), yyyymmdd	Date package received.
<dateRejected>	Optional	0 -1	Date (8), yyyymmdd	Date package rejected.
<dateReverted>	Optional	0 -1	Date (8), yyyymmdd	Date package reverted.
<dateSent>	Optional	0 -1	Date (8), yyyymmdd	Date package sent.
<dateTempChangeCycled>	Optional	0 - 1	Date (10), yyyymmdd	Date when temporary change package expired.
<generalComponentStatus>	Optional	0 - 1	String (1)	General status code for all package components as a group. Values: **0** = Active **1** = Approved **2** = Checked out **3** = Demoted **4** = Frozen **5** = Inactive **6** = Incomplete **7** = Promoted **8** = Refrozen **9** = Rejected **A** = Remote promoted **B** = Submitted **C** = Unfrozen
<installTimeExpiration>	Optional	0 -1	Time (8), hhmm	Ending time for installation window on next planned install, 24-hour. ***NOTE:*** No punctuation included in time returned by ZMF.
<isApprovalPending>	Optional	0 - 1	String (1)	**Y** = Yes, approval pending **N** = No, approval not pending
<isAuditPending>	Optional	0 - 1	String (1)	**Y** = Yes, audit pending **N** = No, audit not pending

Exhibit 3-38. PACKAGE GEN_PRMS LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<isBackedOut>	Optional	0 - 1	String (1)	**Y** = Yes, package backed out **N** = No, package not backed out
<isCmnSchedulerUsed>	Optional	0 - 1	String (1)	**Y** = Yes, package uses ZMF installation scheduler **N** = No, ZMF scheduler not used ***NOTE:*** Value should be "Y" if `<schedulerType>` = 1. ***NOTE:*** Value should be "N" if `<schedulerType>` = 2 or 3.
<isFreezePending>	Optional	0 - 1	String (1)	**Y** = Yes, package freeze pending **N** = No, freeze not pending
<isInstallPending>	Optional	0 - 1	String (1)	**Y** = Yes, package install pending **N** = No, install not pending
<isLinkedPackage>	Optional	0 - 1	String (1)	**Y** = Yes, this package is linked **N** = No, not a linked package
<isManualSchedulerUsed>	Optional	0 - 1	String (1)	**Y** = Yes, manual installation **N** = No, install is automated ***NOTE:*** Value should be "Y" if `<schedulerType>` = 2. ***NOTE:*** Value should be "N" if `<schedulerType>` = 1 or 3.
<isOtherSchedulerUsed>	Optional	0 - 1	String (1)	**Y** = Yes, package uses 3rd-party installation scheduler **N** = No 3rd-party scheduler used ***NOTE:*** Value should be "Y" if `<schedulerType>` = 3. ***NOTE:*** Value should be "N" if `<schedulerType>` = 1 or 2.
<isPostApprovalPending>	Optional	0 - 1	String (1)	**Y** = Yes, post-approval pending **N** = No post-approval pending
<isPostApproversAdded>	Optional	0 - 1	String (1)	**Y** = Yes, post-approver list added **N** = No, list not added
<isPostRejected>	Optional	0 - 1	String (1)	**Y** = Yes, package post-rejected **N** = No, not post-rejected
<isReverseRippleSubmitted>	Optional	0 - 1	String (1)	**Y** = Yes, baseline reverse ripple job submitted. **N** = No, baseline reverse ripple job not submitted.
<isRevertPending>	Optional	0 - 1	String (1)	**Y** = Yes, package revert pending **N** = No, revert not pending

Exhibit 3-38. PACKAGE GEN_PRMS LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<isShortApproverListUsed>	Optional	0 - 1	String (1)	**Y** = Yes, post-approver list has emergency approvers only **N** = No, not using emergency list of package approvers
<isStageLibsDeleted>	Optional	0 - 1	String (1)	**Y** = Yes, staging libraries deleted **N** = No, staging libs not deleted
<isXNodeBuildPending>	Optional	0 - 1	String (1)	**Y** = Yes, JCL install build pending **N** = No, JCL build not pending
<isXNodeBuildRequired>	Optional	0 - 1	String (1)	**Y** = Yes, JCL install build required **N** = No, JCL build not required
<lastBackoutUserid>	Optional	0 - 1	String (8), variable	TSOID of the last package backout.
<lastPromoter>	Optional	0 - 1	String (8), variable	TSO user ID of most recent promoter/demoter.
<lastPromotionAction>	Optional	0 - 1	String (1)	Code for most recent promotion action. Values: **0** = No promotion **1** = Full demotion **2** = Full promotion **3** = Reverted **4** = Selective demotion **5** = Selective promotion **6** = First promotion
<lastPromotionDate>	Optional	0 - 1	Date (10), yyyymmdd	Date of most recent promotion action.
<lastPromotionLevel>	Optional	0 - 1	String (2), variable	Most recent promotion level in user-defined promotion hierarchy.
<lastPromotionName>	Optional	0 - 1	String (8), variable	Name of most recent promotion action.
<lastPromotionSite>	Optional	0 - 1	String (8), variable	Site name where most recent promotion action took place.
<lastPromotionTime>	Optional	0 - 1	Time (8), hhmmss	Time of most recent promotion action, 24-hour format.
<lastRevertUserid>	Optional	0 - 1	String (8), variable	TSOID of the last package revert.
<nearestInstallDate>	Optional	0 -1	Date (8), yyyymmdd	Next planned installation date among prescheduled site installs.

Exhibit 3-38. PACKAGE GEN_PRMS LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<nonSourceComponentStatus>	Optional	0 - 1	String (1)	Status code for non-source package components as a group. Values: **0** = Active **1** = Approved **2** = Checked out **3** = Demoted **4** = Frozen **5** = Inactive **6** = Incomplete **7** = Promoted **8** = Refrozen **9** = Rejected **A** = Remote promoted **B** = Submitted **C** = Unfrozen
<otherProblemAction>	Optional	0 -1	String (44), variable	Text of “Other” instructions if installation problem occurs. ***NOTE:*** Returned when value of `<problemActionCode>` = 3.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageCheckedIntoRelease>	Optional	0 - 1	String (1)	**Y** = Yes, checked into release **N** = No, not checked into release
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.
<packageLevel>	Optional	1	Integer (1)	Code for package complexity level. Values: **1** = Simple package **2** = Complex package **3** = Super package **4** = Participating package ***NOTE:*** If value = 4, name of complex/super package is returned in <complexSuperPackage>.

Exhibit 3-38. PACKAGE GEN_PRMS LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<packageStatus>	Required	1	String (1)	Code for status of package in lifecycle. Values: **1** = Approved **2** = Backed out **3** = Baselined **4** = Complex/super pkg closed **5** = Deleted (memo delete) **6** = Development **7** = Distributed **8** = Frozen **9** = Installed **A** = Complex/super pkg open **B** = Rejected **C** = Temporary change cycle completed ***NOTE:*** Only values 6 or A should be returned for package create.
<packageTitle>	Optional	0 -1	String (72), variable	Working title for package. Appears on most listings & reports.
<packageType>	Optional	0 - 1	String (1)	Package install type code. Values: **1** = Planned permanent **2** = Planned temporary **3** = Unplanned permanent **4** = Unplanned temporary ***NOTE:*** For code values = 2 or 4, `<tempChangeDuration>` also required. ***NOTE:*** For code values = 3 or 4, `<reasonCode>` also required.
<problemActionCode>	Optional	0 -1	Integer (1)	Code for action to take if problem occurs in package install. Values: **1** = Hold production & contact developer for instructions **2** = Back out change, then proceed with production **3** = Other instructions ***NOTE:*** If value = 3, instructions in `<otherProblemAction>` tag also returned.
<reasonCode>	Optional	0 -1	String (3), variable	Customer-defined reason code for unplanned package installation. ***NOTE:*** Returned if value of `<packageType>` = 3 or 4.
<release>	Optional, ZMF with ERO only	0 -1	String (8)	Name of ERO release with which package is associated.

Exhibit 3-38. PACKAGE GEN_PRMS LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<releaseArea>	Optional, ZMF with ERO only	0 -1	String (8)	Name of starting release area for release package checkin.
<releaseJoinedDate>	Optional	0 -1	Date (8), yyyymmdd	Date package joined release.
<releaseJoinedTime>	Optional	0 -1	Time (6), HHMMSS	Time package joined release.
<requestorDept>	Optional	0 -1	String (4), variable	Workgroup or department code for package requestor.
<requestorName>	Optional	0 -1	String (25), variable	Name of developer or contact person requesting package create.
<requestorPhone>	Optional	0 -1	String (15), variable	Phone of developer or contact person requesting package create.
<schedulerType>	Optional	0 -1	Integer (1)	Code for type of installation scheduler. Values: **1** = ChangeMan scheduler **2** = Manual install **3** = Other automated scheduler
<siteInfoStatus>	Optional	0 - 1	String (1)	Status code for site components of package as a group. Values: **0** = Active **1** = Approved **2** = Checked out **3** = Demoted **4** = Frozen **5** = Inactive **6** = Incomplete **7** = Promoted **8** = Refrozen **9** = Rejected **A** = Remote promoted **B** = Submitted **C** = Unfrozen

Exhibit 3-38. PACKAGE GEN_PRMS LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<sourceLoadComponentStatus>	Optional	0 - 1	String (1)	Status code for source & load components of package as a group. Values: **0** = Active **1** = Approved **2** = Checked out **3** = Demoted **4** = Frozen **5** = Inactive **6** = Incomplete **7** = Promoted **8** = Refrozen **9** = Rejected **A** = Remote promoted **B** = Submitted **C** = Unfrozen
<stageDevLibModel>	Optional	0 - 1	String (36), variable	Qualified name of model library to use when staging from development environments outside ZMF control.
<stageLibStatus>	Optional	0 - 1	String (1)	Status code for package staging library. Values: **1** = Absent **2** = Present **3** = Archived
<stageProdLibModel>	Optional	0 - 1	String (36), variable	Qualified name of model library to use when staging from ZMF baseline or production libraries.
<tempChangeDuration>	Optional	0 -1	Integer (3)	Number of days for temporary package to remain in production. ***NOTE:*** Returned if value of `<packageType>` = 2 or 4.
<timeApproved>	Optional	0 -1	Time (6), hhmmss	Time package approved, 24-hour.
<timeBackedOut>	Optional	0 -1	Time (6), hhmmss	Time package backed out, 24-hour.
<timeBaselined>	Optional	0 -1	Time (6), hhmmss	Time package baselined, 24-hour.
<timeCreated>	Optional	0 -1	Time (6), hhmmss	Time package created, 24-hour.
<timeFrozen>	Optional	0 -1	Time (6), hhmmss	Time package frozen, 24-hour.
<timeInstalled>	Optional	0 -1	Time (6), hhmmss	Time package installed, 24-hour.

Exhibit 3-38. PACKAGE GEN_PRMS LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<timeMemoDeleted>	Optional	0 -1	Time (6), hhmmss	Time package logically deleted, 24-hour format.
<timeReceived>	Optional	0 -1	Time (6), hhmmss	Time package received, 24-hour.
<timeRejected>	Optional	0 -1	Time (6), hhmmss	Time package rejected, 24-hour.
<timeReverted>	Optional	0 -1	Time (6), hhmmss	Time package reverted, 24-hour.
<timeSent>	Optional	0 -1	Time (6), hhmmss	Time package sent, 24-hour.
<timeTempChangeCycled>	Optional	0 - 1	Time (6), hhmmss	Time when temporary change package expired, 24-hour format.
<utilityInfoStatus>	Optional	0 - 1	String (1)	Status code for scratch/rename components of package as a group. Values: **0** = Active **1** = Approved **2** = Checked out **3** = Demoted **4** = Frozen **5** = Inactive **6** = Incomplete **7** = Promoted **8** = Refrozen **9** = Rejected **A** = Remote promoted **B** = Submitted **C** = Unfrozen
<workChangeRequest>	Optional	0 -1	String (12), variable	Work order ID or change request number for package.

Unfreeze Package Parameters - PACKAGE GEN_PRMS UNFREEZE

The Serena XML function to unfreeze general package parameters unlocks those parameters for change. You can then change the scheduled installation date or implementation instructions, make a temporary change permanent, or update the package description to better fit the delivered code. The package and its components remain frozen overall.

The Serena XML service/scope/message tags for a general package parameters *unfreeze* message are:

```
<service name="PACKAGE">
<scope name="GEN_PRMS">
<message name="UNFREEZE">
```

These tags appear in both requests and replies.

PACKAGE GEN_PRMS UNFREEZE — Requests

The `<request>` tag syntax for a general package parameters unfreeze request is identical to that for many package information management functions, including the package description list and package general description list. Only the `name` parameters in the high-level `<scope>` and `<message>` tags differ, as shown above.

PACKAGE GEN_PRMS UNFREEZE — Replies

The Serena XML reply message to an unfreeze request for general package parameters does not return a `<result>` data structure. It does, however, return a standard `<response>` data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Refreeze Package Parameters - PACKAGE GEN_PRMS REFREEZE

The Serena XML refreeze function for general package parameters resets these previously unfrozen parameters to frozen status, locking them down against change.

The Serena XML service/scope/message tags for a general package parameters *refreeze* message are:

```
<service name=”PACKAGE”>
<scope name=”GEN_PRMS”>
<message name=”REFREEZE”>
```

These tags appear in both requests and replies.

PACKAGE GEN_PRMS REFREEZE — Requests

The `<request>` tag syntax for a general package parameters *refreeze* request is identical to that for an *unfreeze* request. Only the `name` parameter in the high-level `<message>` tag differs, as shown above.

PACKAGE GEN_PRMS REFREEZE — Replies

The Serena XML reply message to a refreeze request for general package parameters does not return a `<result>` data structure. It does, however, return a standard `<response>` data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

List User-Defined Package Variables - PACKAGE USR_RECS LIST

Serena XML supports up to 72 user-defined package variables established by users when they customize ChangeMan ZMF. These are stored in the package master record with the general parameters for a package, but are listed separately.

The Serena XML service/scope/message names for a message to *list* the user-defined variables for a package are:

```
<service name="PACKAGE">
<scope name="USR_RECS">
<message name="LIST">
```

These tags appear in both requests and replies.

Naming Conventions for User-Defined Variables in Serena XML

Serena XML tag names for user-defined package variables take the general form:

`<userVarLen`**`xxyy`**`>`

where:

- **`xx`** = length of variable data in bytes, formatted as 1-digit or 2-digit integer
- **`yy`** = unique 2-digit integer identifier for this particular variable of length **`xx`**

For example, `<userVarLen103>` represents the third user-defined variable with a length of one byte. Similarly, `<userVarLen4405>` is the fifth variable with a length of 44 bytes.

ChangeMan ZMF stores these values for user reference at customized exit points, but otherwise ignores them; internally, they are meaningless. Similarly, Serena XML retrieves these values without respect to any meaning they may hold for the user. It is the user's responsibility to know the meaning of these variables and to manage them accordingly.

PACKAGE USR_RECS LIST — Requests

The following example shows how you might code a request to list user-defined variables for a package in Serena XML. Notice the similarity of this syntax with that of many other package requests.

Example XML — PACKAGE USR_RECS LIST Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="USR_RECS">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>IMSQ000012</package>
   </request>
  </message>
 </scope>
</service>
```

PACKAGE USR_RECS LIST — Replies

User-defined variable lists for a package return nor more than one `<result>` tag. This tag is followed by a standard `<response>` tag that indicates the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

An example XML reply to a user-defined variable list request appears below. Because the reply may contain values for up to 72 user-defined variables, many optional tags for these variables are omitted for clarity. Data structure details for the `<result>` tag follow the example in *Exhibit 3-39*.

Example XML — PACKAGE USR_RECS LIST Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="USR_RECS">
  <message name="LIST">
   <result>
    <package>IMSQ000012</package>
    <applName>IMSQ</applName>
    <packageId>000012</packageId>
    <userVarLen199>Y</userVarLen199>
    <userVarLen301>NO</userVarLen301>
    <userVarLen302>NO</userVarLen302>
    <userVarLen303>NO</userVarLen303>
    <userVarLen304>NO</userVarLen304>
    <userVarLen305>NO</userVarLen305>
    <userVarLen306>NO</userVarLen306>
    <userVarLen401>NO</userVarLen401>
    <userVarLen402>NO</userVarLen402>
    <userVarLen403>NO</userVarLen403>
    <userVarLen404>NO</userVarLen404>
   </result>
   <response>
    <statusMessage>CMN8700I - LIST     User record service completed</
statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-39. PACKAGE USR_RECS LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name.
<package>	Required	1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.
<userVarLen101> . . . <userVarLen115> <userVarLen199>	Optional	0 -1 each	String (1)	User-defined in ZMF. Total of 16 individually named 1-byte tags supported. ***NOTE:*** See topic “Package User Information” in the ChangeMan ZMF Customization Guide.
<userVarLen201> . . . <userVarLen211>	Optional	0 -1 each	String (2), variable	User-defined in ZMF. Total of 11 individually named 2-byte tags supported.
<userVarLen301> . . . <userVarLen310>	Optional	0 -1 each	String (3), variable	User-defined in ZMF. Total of 10 individually named 3-byte tags supported.
<userVarLen401> . . . <userVarLen410>	Optional	0 -1 each	String (4), variable	User-defined in ZMF. Total of 10 individually named 4-byte tags supported.
<userVarLen801> . . . <userVarLen810>	Optional	0 -1 each	String (8), variable	User-defined in ZMF. Total of 10 individually named 8-byte tags supported.
<userVarLen1601> . . . <userVarLen1605>	Optional	0 -1 each	String (16), variable	User-defined in ZMF. Total of 5 individually named 16-byte tags supported.
<userVarLen4401> . . . <userVarLen4405>	Optional	0 -1 each	String (44), variable	User-defined in ZMF. Total of 5 individually named 44-byte tags supported.
<userVarLen7201> . . . <userVarLen7205>	Optional	0 -1 each	String (72), variable	User-defined in ZMF. Total of 5 individually named 72-byte tags supported.

Tip

Tags: <userVarLen101> to <userVarLen7205>. See topic “Package User Information” in the ChangeMan ZMF Customization Guide.

List Package Install Sites - SITE PKG LIST

The planned install sites for a package can be listed using Serena XML. This function assumes that the sites themselves already exist, thanks to site maintenance performed elsewhere by the ChangeMan ZMF administrator. The function also assumes that package create or update operations have already assigned install sites to the package. If neither condition is met, no sites will be returned by the package install site list function.

The Serena XML service/scope/message tags for a package install site *list* message are:

```
<service name=”SITE”>
<scope name=”PKG”>
<message name=”LIST”>
```

These tags appear in both requests and replies.

The service name is `“site”`, not `“package”`, because XML Services calls the low-level site maintenance service in ChangeMan ZMF to perform most tasks associated with this function. The scope name, `“pkg”`, identifies this function as a package-level site service.

SITE PKG LIST — Requests

Serena XML supports two kinds of package install site lists:

- ***All Install Sites for Package*** — Name the desired package in the `<package>` tag. Omit the `<siteName>` tag. All install sites for the named package are returned, together with site descriptions and installation status information.
- ***Package Install Status for Site*** — Name the desired package in the `<package>` tag and the desired install site in the `<siteName>` tag. Installation status information is returned for the named package and site.

The following example shows how you might code a Serena XML request to list install status for one remote install site associated with a package.

Example XML — SITE PKG LIST Request

```
<?xml version="1.0"?>
<service name="SITE">
 <scope name="PKG">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>IMSQ000012</package>
   </request>
  </message>
 </scope>
</service>
```

SITE PKG LIST — Replies

The Serena XML reply to a package install site list request contains zero to many `<result>` tags. Each `<result>` tag contains site description and install status information about one remote site associated with the named package.

A standard `<response>` tag follows the `<result>`, where it can serve as an end-of-list marker. It reports the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

The example below shows what a package install list reply message might look like. Data structure details for the `<result>` tag appear after the example in *Exhibit 3-40*.

Example XML — SITE PKG LIST Reply

```
<?xml version="1.0"?>
<service name="SITE">
 <scope name="PKG">
  <message name="LIST">
   <result>
    <package>IMSQ000012</package>
    <applName>IMSQ</applName>
    <packageId>000012</packageId>
    <siteName>SERT8</siteName>
    <installDate>20081231</installDate>
    <fromInstallTime>010000</fromInstallTime>
    <toInstallTime>020000</toInstallTime>
    <contactName>DDDDDDDD</contactName>
    <contactPhone>1234567</contactPhone>
    <alternateContactName>DDDDDDDD</alternateContactName>
    <alternateContactPhone>1234567</alternateContactPhone>
    <siteLibModel>CMNTP.SERT8.DEV.IMSQ.#000012</siteLibModel>
    <dateSent>20081028</dateSent>
    <timeSent>101800</timeSent>
    <dateReceived>20081028</dateReceived>
    <timeReceived>101800</timeReceived>
    <dateInstalled>20081028</dateInstalled>
    <timeInstalled>101900</timeInstalled>
    <dateBackedOut>20081212</dateBackedOut>
    <timeBackedOut>070200</timeBackedOut>
    <dateReverted>20081212</dateReverted>
    <timeReverted>070500</timeReverted>
    <backoutReasons>TEST</backoutReasons>
    <backoutReason01>TEST</backoutReason01>
    <db2InstallBindJobCount>00</db2InstallBindJobCount>
    <db2BackoutBindJobCount>00</db2BackoutBindJobCount>
    <db2RippleBindJobCount>00</db2RippleBindJobCount>
    <db2ReverseRippleBindJobCount>00</db2ReverseRippleBindJobCount>
    <revertUserid>USER24</revertUserid>
    <backoutUserid>USER24</backoutUserid>
    <siteStatus>DEV</siteStatus>
   </result>
   <response>
    <statusMessage>CMN8700I - Site Name service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-40. SITE PKG LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<alternateContactName>	Optional	0 - 1	String (25), variable	Name of alternate analyst or point of contact for install problem.
<alternateContactPhone>	Optional	0 - 1	String (15), variable	Phone number of contact in `<alternateContactName>`.
<applName>	Optional	0 - 1	String (4), fixed	ZMF application name. Same as first 4 bytes of package name.
<backoutReason01> . . . <backoutReason09>	Optional	0 - 1 each	String (72), variable	Up to nine sequential notations for reason package backed out at site. ***NOTE:*** If `<dateBackedOut>` is non-zero, `<backoutReason01>` is required.
<backoutReasons>	Optional	0 - 1	String (72), variable	Reason package backed out at site.
<backoutUserid>	Optional	0 - 1	String (8), variable	USERID that performed backout.

Exhibit 3-40. SITE PKG LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<contactName>	Optional	0 - 1	String (25), variable	Name of analyst originating package, or point of contact for install problem.
<contactPhone>	Optional	0 - 1	String (15), variable	Phone number of contact in `<contactName>`.
<dateBackedOut>	Optional	0 - 1	Date (8), yyyymmdd	Date package backed out at site.
<dateInstalled>	Optional	0 - 1	Date (8), yyyymmdd	Date package installed at site.
<dateReceived>	Optional	0 - 1	Date (8), yyyymmdd	Date package received at site.
<dateReverted>	Optional	0 - 1	Date (8), yyyymmdd	Date package reverted.
<dateTempChangeCycled>	Optional	0 - 1	Date (8), yyyymmdd	Date temporary change expired.
<dateSent>	Optional	0 - 1	Date (8), yyyymmdd	Date package sent to site.
<db2BackoutBindJobCount>	Optional	0 - 1	Integer (2), fixed	Number of DB2 backout bind jobs executed at site. ***NOTE:*** Requires ZMF DB2 Option.
<db2InstallBindJobCount>	Optional	0 - 1	Integer (2), fixed	Number of DB2 install bind jobs executed at site. ***NOTE:*** Requires ZMF DB2 Option.
<db2Reverse RippleBindJobCount>	Optional	0 -1	Integer (2), fixed	Number of DB2 baseline reverse ripple bind jobs executed at site. ***NOTE:*** Requires ZMF DB2 Option.
<db2RippleBindJobCount>	Optional	0 -1	Integer (2), fixed	Number of DB2 baseline ripple bind jobs executed at site. ***NOTE:*** Requires ZMF DB2 Option.
<fromInstallTime>	Optional	0 - 1	Time, hhmmss	Start time for period during which installation of package is planned at named site, 24-hour format.
<installDate>	Optional	0 - 1	Date, yyyymmdd	Planned site install date for package.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.
<revertUserid>	Optional	0 - 1	String (8), variable	TSOID of reverter.

Exhibit 3-40. SITE PKG LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<siteLibModel>	Optional	0 - 1	String (32), variable	Fully qualified z/OS data set name of library used as model for library setup when package is installed.
<siteName>	Optional	0 - 1	String (8), variable	ZMF name of install site.
<siteStatus>	Optional	0 - 1	String (3), variable	Determined site status.
<timeBackedOut>	Optional	0 - 1	Time (8), hhmmss	Time package backed out, 24-hour.
<timeInstalled>	Optional	0 - 1	Time (8), hhmmss	Time package installed, 24-hour.
<timeReceived>	Optional	0 - 1	Time (8), hhmmss	Time package received, 24-hour.
<timeReverted>	Optional	0 - 1	Time (8), hhmmss	Time package reverted, 24-hour.
<timeSent>	Optional	0 - 1	Time (8), hhmmss	Time package sent, 24-hour.
<timeTempChangeCycled>	Optional	0 - 1	Time (8), hhmmss	Time temporary change expired, 24-hour format.
<toInstallTime>	Optional	0 - 1	Time, hhmmss	End time for period during which installation of package is planned at named site, 24-hour format.

Unfreeze Package Install Sites - PACKAGE SITES UNFREEZE

The Serena XML function to unfreeze package install sites unlocks these site assignments for change. The XML service/scope/message tags for a package-level site *unfreeze* message are:

```
<service name="PACKAGE">
<scope name="SITES">
<message name="UNFREEZE">
```

These tags appear in both requests and replies.

PACKAGE SITES UNFREEZE — Requests

The `<request>` tag syntax for a package install site unfreeze request is identical to that for for many package information management functions, including the package description list and package general description list. Only the `name` parameters in the high-level `<scope>` and `<message>` tags differ, as shown above.

PACKAGE SITES UNFREEZE — Replies

The Serena XML reply message to an unfreeze request for package install sites does not return a `<result>` data structure. It does, however, return a standard `<response>` data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Refreeze Package Install Sites - PACKAGE SITES REFREEZE

The Serena XML refreeze function for package install sites resets these previously unfrozen assignments to frozen status, locking them down against change.

The XML service/scope/message tags for a package-level site *refreeze* message are:

```
<service name=”PACKAGE”>
<scope name=”SITES”>
<message name=”REFREEZE”>
```

These tags appear in both requests and replies.

PACKAGE SITES REFREEZE — Requests

The `<request>` tag syntax for a general package parameters *refreeze* request is identical to that for an *unfreeze* request. Only the `name` parameter in the high-level `<message>` tag differs, as shown above.

PACKAGE SITES REFREEZE — Replies

The Serena XML reply message to a refreeze request for package install sites does not return a `<result>` data structure. It does, however, return a standard `<response>` data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

List Package Installation Dependencies - PACKAGE SCH_RECS LIST

ChangeMan ZMF captures package installation dependencies in the package installation schedule records. Serena XML can list all such dependency records for a package, or can selectively determine whether a dependency exists between a package and a particular job.

The Serena XML service/scope/message names for message to *list* package installation dependency records are:

```
<service name=”PACKAGE”>
<scope name=”SCH_RECS”>
<message name=”LIST”>
```

These tags appear in both request and reply messages.

PACKAGE SCH_RECS LIST — Requests

Serena XML supports two types of package installation dependency requests:

- ***List All Installation Dependencies*** — Name the desired package in the `<package>` tag. Omit the `<predecessorJob>` and `<successorJob>` tags, or enter a "match-all" (asterisk) wild card in each. The function returns a list of all predecessor and successor jobs that must execute before or after package installation to complete a successful install.
- ***Selective Installation Dependency Check*** — Name the desired package in the `<package>` tag. Enter the job name to check for an installation dependency in the `<predecessorJob>` tag, the `<successorJob>` tag, or both as appropriate. Optionally, enter a wildcard pattern to check for several similar job names. The function returns installation scheduling dependency information for the named job(s) if such dependencies exist. Otherwise, no `<result>` is returned in the reply message.

The following example shows how you might code a request to list all package installation dependencies in Serena XML. Notice the use of match-all wildcard characters in the `<predecessorJob>` and `<successorJob>` tags. Data structure details for the `<request>` tag appear in *Exhibit 3-41*.

Example XML — PACKAGE SCH_RECS LIST Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="SCH_RECS">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>TES5000001</package>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-41. PACKAGE SCH_RECS LIST <request>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended to replace `<package>`. Use `<package>` instead of `<applName>` & `<packageId>`.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<predecessorJob>	Optional	0 - 1	String (8), variable	Name of job(s) that must run before package is installed. ***NOTE:*** Accepts standard wildcards & patterns using question mark (?) & asterisk (*). ***NOTE:*** Omit tag or use asterisk (*) wildcard to list all predecessor jobs.
<successorJob>	Optional	0 - 1	String (8), variable	Name of job(s) that must run after package is installed. ***NOTE:*** Accepts standard wildcards & patterns using question mark (?) & asterisk (*). ***NOTE:*** Omit tag or use asterisk (*) wildcard to list all successor jobs.

PACKAGE SCH_RECS LIST — Replies

Serena XML returns zero to many `<result>` tags in package installation dependency list reply. Each `<result>` tag contains the name of a predecessor job, a successor job, or both that the package requires for successful installation. The `<result>` tag recurs as needed to accommodate all scheduled predecessor and successor jobs.

A standard `<response>` tag follows the last `<result>` tag to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last tag returned in the reply message, the `<response>` tag may serve as an end-of-list marker.

An example XML reply that lists package installation scheduling records appears on the next page. Data structure details for the `<result>` tag follow in *Exhibit 3-42*.

Example XML — PACKAGE SCH_RECS LIST Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="SCH_RECS">
  <message name="LIST">
   <result>
    <package>TES5000001</package>
    <applName>TES5</applName>
    <packageId>000001</packageId>
    <successorJob>SCHJOB01</successorJob>
    <predecessorJob>SCHJOB02</predecessorJob>
   </result>
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-42. PACKAGE SCH_RECS LIST <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<package>	Optional	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.
<predecessorJob>	Optional	0 - 1	String (8), variable	Name of a job that must run before package is installed.
<successorJob>	Optional	0 - 1	String (8), variable	Name of a job that must run after package is installed.

List Package Implementation Instructions - PACKAGE IMP_INST LIST

You can retrieve package implementation instructions independently of other package parameters using Serena XML. Results are returned for one package.

The Serena XML service/scope/message names for message to *list* implementation instructions for a package are:

```
<service name="PACKAGE">
<scope name="IMP_INST">
<message name="LIST">
```

These tags appear in both request and reply messages.

PACKAGE IMP_INST LIST — Requests

The `<request>` tag syntax for request to list package implementation instructions is identical to that for many package information management functions, including the package description list and package general description list. Only the `name` parameter in the high-level `<scope>` tag differs, as shown above.

PACKAGE IMP_INST LIST — Replies

The reply message for a package implementation instruction list includes one `<result>` tag with package name and implementation instructions, if the package is found. This tag is followed by a standard `<response>` tag that indicates the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

An example XML reply to a package description list request appears below. Data structure details for the `<result>` tag follow the example in *Exhibit 3-43*.

Example XML — PACKAGE IMP_INST LIST Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="IMP_INST">
  <message name="LIST">
   <result>
    <package>TES5000001</package>
    <applName>TES5</applName>
    <packageId>000001</packageId>
    <packageImplInst>CR153620</packageImplInst>
   </result>
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-43. PACKAGE IMP_INST LIST <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), fixed	ZMF application name. Same as first 4 bytes of package name.
<package>	Optional	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.
<packageImplInst>	Optional	0 - 46	String (72), variable	Implementation instructions in free format text. Repeatable to accommodate multiple lines of text.

List Package Approvers - APPROVER PKG LIST

The Serena XML function to list package approvers retrieves an instantiated list of actual approvers for a named package. Actual approvers are those relevant to the named package after applying the approver business rules established by your administrator. They comprise a subset of relevant application approvers, emergency approvers for unplanned or temporary changes, and approvers assigned to review this package by a customized ChangeMan ZMF exit.

Note

The list of *authorized* approvers for a package, *before* the application of business rules, is associated with the parent application of the package rather than the package itself. Application approvers are retrieved with a different Serena XML function.

The Serena XML service/scope/message tags for a message to *list* package approvers are:

```
<service name="APPROVER">
<scope name="PKG">
<message name="LIST">
```

These tags appear in both request and reply messages.

The service name is "approver", not "package", because XML Services calls the low-level approver maintenance service in ChangeMan ZMF to perform most tasks associated with this function. The scope name, "pkg", identifies this function as a package-level service.

APPROVER PKG LIST — Requests

Serena XML supports two types of package approver lists:

- ***All Approvers for Named Package*** — Name the desired package in the `<package>` tag. Enter a "match-all" (asterisk) wildcard character in the `<approverEntity>` tag or omit it altogether. Returns all package approvers and reports their approval actions (approved, rejected, reviewing, no response).
- ***Approval Activity for Named Approver(s)*** — Name the desired package in the `<package>` tag. Enter the desired approver entity ID, as defined to RACF or other security system, in the `<approverEntity>` tag. Returns approver description for all TSO user IDs associated with the named approver entity and reports their approval actions (approved, rejected, reviewing, no response).

The following example shows how you might code a request to list all approvers for a package. Data structure details for the <request> tag appear in *Exhibit 3-44*.

Example XML — APPROVER PKG LIST Request

```
<?xml version="1.0"?>
<service name="APPROVER">
 <scope name="PKG">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000007</package>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-44. APPROVER PKG LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended to replace `<package>`. Use `<package>` instead of `<applName>` & `<packageId>`.

Exhibit 3-44. APPROVER PKG LIST <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<approverEntity>	Optional	0 - 1	String (8), variable	TSO user ID or security system entity ID of desired package approver. ***NOTE:*** Accepts standard wildcards & patterns using question mark (?) & asterisk (*). ***NOTE:*** Omit tag or use asterisk (*) wildcard to list all approver entities.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.

APPROVER PACKAGE LIST — Replies

The Serena XML reply to a package approver list request returns zero to many `<result>` tags. Each `<result>` tag contains information about one package approver, including TSO user ID, the associated approver entity defined in RACF (or other security system), an approver entity description, and approver status in the approval process. If multiple TSO user IDs are associated with the RACF approval entity, each generates a separate `<result>` tag.

A standard `<response>` data element follows the last `<result>` tag, if any, to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last tag returned in the reply message, the `<response>` tag serves as an end-of-list marker.

An example XML reply that lists package approvers appears below. Data structure details for the `<result>` tag follow in *Exhibit 3-45*.

Example XML — APPROVER PKG LIST Reply

```
<?xml version="1.0"?>
<service name="APPROVER">
 <scope name="PKG">
  <message name="LIST">
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <approverEntity>ACTPLEAD</approverEntity>
    <approverDesc>Lead Programmer - ACTP Application</approverDesc>
    <approverAction>1</approverAction>
    <approvedDate>20090127</approvedDate>
    <approvedTime>083100</approvedTime>
    <approver>USER24</approver>
    <approvalOrder>10</approvalOrder>
    <userListCount>02</userListCount>
    <isApproverNotified>Y</isApproverNotified>
    <postApprovalNoticeEnabled>N</postApprovalNoticeEnabled>
    <notification>
     <notifierType>4</notifierType>
     <userList>USER24@SERENA.COM;USER24</userList>
    </notification>
    <notification>
     <notifierType>1</notifierType>
     <userList>USER24</userList>
    </notification>
   </result>
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <approverEntity>ACCTPAY</approverEntity>
    <approverDesc>Accounts Payable Manager</approverDesc>
    <approverAction>1</approverAction>
    <approvedDate>20090127</approvedDate>
    <approvedTime>083100</approvedTime>
    <approver>USER24</approver>
    <approvalOrder>20</approvalOrder>
    <userListCount>01</userListCount>
    <isApproverNotified>Y</isApproverNotified>
    <postApprovalNoticeEnabled>N</postApprovalNoticeEnabled>
    <notification>
     <notifierType>4</notifierType>
     <userList>USER24@SERENA.COM;USER24</userList>
    </notification>
   </result>
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-45. APPROVER PKG LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), fixed	ZMF application name. Same as first 4 bytes of package name.
<approvalOrder>	Optional	0 - 1	Integer (2), variable	Approval level or sequence assigned to this approver entity for hierarchical approvals.
<approvedDate>	Optional	0 - 1	Date, yyyymmdd	Date package approval action taken by this approver. No punctuation.
<approvedTime>	Optional	0 - 1	Time, hhmmss	Time package approval action taken by this approver. No punctuation.
<approver>	Optional	0 - 1	String (8), variable	TSO user ID of individual approver. Mapped to `<approverEntity>` by RACF or other security system.
<approverAction>	Optional	0 - 1	Integer (1)	Code for most recent approval action of approver entity. Values: **1** = Approved **2** = Checkoff **3** = Rejected **4** = Review pending **5** = No response to notification
<approverDesc>	Optional	0 - 1	String (44), variable	Text description of approver level or function (e.g., project leader, QA manager) for `<approverEntity>`.
<approverEntity>	Optional	1	String (8), variable	Security system entity ID of package approver. Mapped to TSO user ID in `<approver>` by RACF or other security system.
<checkoffList>	Optional	0 - 14	String (72), variable	Checkoff list.
<checkoffList01> . . . <checkoffList14>	Optional	0 - 1 each	String (72), variable	Text descriptions for up to 14 possible approval actions taken by approver from custom-defined checkoff list. ***NOTE:*** At least one tag required if value in `<approverAction>` = 2.
<isApproverNotified>	Optional	0 - 1	String (1), variable	Has approver entity been notified that approval action is requested? **Y** = Yes **N** = No
<isLinkedApprover>	Optional	0 - 1	String (1), variable	Is this a linked package approver? **Y** = Yes **N** = No

Exhibit 3-45. APPROVER PKG LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<notification>	Optional	0 - 35	Complex	Describes notifications sent when this approver takes an approval action. See *Exhibit 3-46*.
<package>	Optional	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.
<postApprovalNoticeEnabled>	Optional	0 - 1	String (1), variable	Is this approver to be notified of emergency/temporary changes for post-installation approval review? **Y** = Yes **N** = No
<rejectReasons>	Optional	0 - 10	String (72), variable	Reject reasons.
<rejectReasons01> . . . <rejectReasons10>	Optional	0 - 1 each	String (72), variable	Up to ten sequentially numbered, free-format text entries containing reason(s) for package rejection by approver. ***NOTE:*** At least one tag required if value in `<approverAction>` = 3.
<userListCount>	Optional	0 - 1	Integer(2)	The number of users to notify, the number of returned <notification> complex tags.

<notification> Subtag

The `<notification>` tag describes the notifications to be issued when this approver entity takes an approval action. The tag represents a complex data structure with subtags of its own. It is repeatable to accommodate multiple approvers and multiple notification methods. Data structure details for this tag appear in *Exhibit 3-46*.

Exhibit 3-46. <notification> Subtag Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<notifierType>	Optional	0 - 1	Integer (1)	ZMF code for notification method to use with notifications sent to users in `<userList>`. Values: **1** = MVS Send message **4** = E-mail **5** = SERNET email msg **6** = Batch messaging job

Exhibit 3-46. <notification> Subtag Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<userList>	Optional	0 - 1	String (44), variable	List of individual approvers to notify when the named approver entity takes an approval action. List consists of user TSO IDs or E-mail addresses separated by commas. ***NOTE:*** TSO IDs required if `<notifierType>` = 1. ***NOTE:*** E-mail addresses required if `<notifierType>` = 4 or 5.

List Affected Applications - PACKAGE AFF_APLS LIST

List the applications affected by a complex/super package using the Serena XML function to list affected applications for a package. This function includes only complex and super packages in its scope.

The Serena XML service/scope/message tags for a message to *list* applications affected by a complex/super package are:

```
<service name="PACKAGE">
<scope name="AFF_APLS">
<message name="LIST">
```

These tags appear in both request and reply messages.

PACKAGE AFF_APLS LIST — Requests

The `<request>` tag syntax for a request to list affected applications is identical to that for a request to list general package parameters. Only the `name` parameter in the high-level `<scope>` tag differs, as shown above. However, additional data constraints apply. (See *Exhibit 3-47*.

Exhibit 3-47. PACKAGE AFF_APLS LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as `<package>` substitute. Use `<package>` instead of `<applName>` & `<packageId>`.

Exhibit 3-47. PACKAGE AFF_APLS LIST <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<package>	Required	0 - 1	String (10), fixed	Fixed-format ZMF package name for the complex/super package.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.

PACKAGE AFF_APLS LIST — Replies

The Serena XML reply message for this function returns zero to many `<result>` tags. Each `<result>` contains names one application affected by the named complex/super package. If no participating packages are attached to the complex/super package, no `<result>` tags are returned.

A standard `<response>` tag follows the last `<result>` tag, if any, to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last tag returned in the reply message, the `<response>` tag may serve as an end-of-list marker.

An example XML reply that lists all affected applications for a package appears below. Data structure details for the `<result>` tag follow in *Exhibit 3-48*.

Example XML — PACKAGE AFF_APLS LIST Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="AFF_APLS">
  <message name="LIST">
   <result>
    <package>TES5000003</package>
    <applName>TES5</applName>
    <packageId>000003</packageId>
    <affectedAppl>ACTP</affectedAppl>
   </result>
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-48. PACKAGE AFF_APLS LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<affectedAppl>	Optional	0 - 1	String (8), variable	ZMF application name for a participating package.
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<package>	Optional	0 - 1	String (10), fixed	Fixed-format ZMF package name for the complex/super package.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.

List Participating Packages - PACKAGE PRT_PKGS LIST

List the participating applications in a complex/super package using the Serena XML function to list participating packages. This function includes only complex/super packages in its scope.

The Serena XML service/scope/message tags for a message to *list* participating packages for a complex/super package are:

```
<service name="PACKAGE">
<scope name="PRT_PKGS">
<message name="LIST">
```

These tags appear in both request and reply messages.

PACKAGE PRT_PKGS LIST — Requests

The `<request>` tag syntax for a request to list participating packages is identical to that for a request to list affected applications. (See *Exhibit 3-47*.) Only the `name` parameter in the high-level `<scope>` tag differs, as shown above.

PACKAGE PRT_PKGS LIST — Replies

The Serena XML reply message for this function returns zero to many `<result>` tags. Each `<result>` names one participating package in the named complex/super package. If no participating packages are attached to the complex/super package, no `<result>` tags are returned.

A standard `<response>` tag follows the last `<result>` tag, if any, to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last tag returned in the reply message, the `<response>` tag may serve as an end-of-list marker.

An example XML reply that lists participating packages follows. Data structure details for the `<result>` tag appear in *Exhibit 3-49*.

Example XML — PACKAGE PRT_PKGS LIST Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="PRT_PKGS">
  <message name="LIST">
   <result>
    <package>TES5000002</package>
    <applName>TES5</applName>
    <packageId>000002</packageId>
    <partPackage>TES5000003</partPackage>
   </result>
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-49. PACKAGE PRT_PKGS LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<package>	Optional	0 - 1	String (10), fixed	Fixed-format ZMF package name of complex/super package.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.
<partPackage>	Optional	0 - 1	String (10), variable	Fixed-format ZMF name of participating package in the complex/super package named in `<package>`.

List Linked Packages - PACKAGE PKG_LINK LIST

Serena XML provides a means for ChangeMan ZMF customers to list any packages on remote LPARs or non-mainframe servers that are linked (via external software) to an explicitly named ChangeMan ZMF package on the local mainframe LPAR. Only simple packages on the local LPAR are included in the scope of this function.

The XML service/scope/message names for a message to *list* linked packages are:

```
<service name=”PACKAGE”>
<scope name=”PKG_LINK”>
<message name=”LIST”>
```

These tags appear in both request and reply messages.

PACKAGE PKG_LINK LIST — Request

Serena XML supports the following linked package list options:

- ***All Remote Packages Linked to a Local Package*** — Name the desired local package in the `<package>` tag. Omit all other subtags of the `<request>` element. The function returns a list of all remote packages linked to the local package.
- ***All Local Packages Linked to a Remote Package*** — Name the desired remote package in the `<linkPackage>` tag. The package naming conventions of the remote system are accepted in `<linkPackage>`. Omit all other subtags of the `<request>` data element. The function returns a list of all local packages linked to the named remote package.
- ***All Local Packages Linked to Packages on a Remote Server*** — In tag `<sourceLinkIpAddress>`, enter the name or IP address of the desired remote server. Use the same naming or addressing conventions used by the remote link management software when it passes these values to ChangeMan ZMF. Omit all other subtags of the `<request>` data element. The function returns a list of all local packages linked to remote packages that reside on the named remote server.
- ***All Local Packages Linked Elsewhere by a User*** — Enter the name or TSO user ID of the desired link requestor in the `<linkRequestor>` tag. Omit all other subtags of the `<request>` data element. The function returns a list of all local packages linked to remote packages when those links were requested by the named user.

The following example shows how you might code a request to list all remote, linked packages for a named local package using Serena XML. Data structure details for the linked package *list* `<request>` tag appear in *Exhibit 3-50*.

Example XML — PACKAGE PKG_LINK LIST Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="PKG_LINK">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>TES5000003</package>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-50. PACKAGE PKG_LINK LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<linkPackage>	Optional	0 - 1	String (255), variable	Name(s) of one or more linked package(s) on remote server, delimited by semicolons. Package naming conventions are those of remote system.
<linkRequestor>	Optional	0 - 1	String (20), variable	Name or TSO user ID of package link requestor.
<package>	Required	0 - 1	String (10), fixed	Fixed-format ZMF package name for target package on local LPAR.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.
<sourceLinkIpAddress>	Optional	0 - 1	String (255), variable	Network IP address for remote server where linked package resides. ***NOTE:*** ZMF stores address as provided by external link management software. May contain server name known to that software instead of an IP address.
<sourceLinkPortid>	Optional	0 - 1	String (8), variable	Network port ID for remote server where linked package resides.

PACKAGE PKG_LINK LIST — Replies

The linked package list reply contains zero to many `<result>` tags. Each `<result>` contains information about a package on the local LPAR that is linked to at least one remote package with the requested characteristics. Remote package name(s), application, server, and link requestor are included as they are stored in the package master records for the local package. Information such as package level, type, and status pertain to the local package, not the linked remote package(s).

A standard `<response>` data structure follows the `<result>` tags, if any, to indicate the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last data element returned in a Serena XML reply message, the `<response>` tag serves as an end-of-list marker.

An example reply to a linked package list request follows. Data structure details for the linked package list `<result>` tag appear in *Exhibit 3-51*.

Example XML — PACKAGE PKG_LINK LIST Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
```

```
 <scope name="PKG_LINK">
  <message name="LIST">
   <result>
    <package>TES5000003</package>
    <applName>TES5</applName>
    <packageId>000003</packageId>
    <packageLevel>4</packageLevel>
    <packageType>1</packageType>
    <packageStatus>6</packageStatus>
    <installDate>20091231</installDate>
    <linkPackage></linkPackage>
    <sourceLinkIpAddress></sourceLinkIpAddress>
   </result>
   <response>
    <statusMessage>CMN8764I - Package is not Linked.</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8764</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-51. PACKAGE PKG_LINK LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of `<package>`.
<installDate>	Optional	0 - 1	Date, yyyymmdd	Planned install date for local package.
<linkDate>	Optional	0 - 1	Date, yyyymmdd	Link date.
<linkPackage>	Optional	0 -1	String (255), variable	Name(s) of one or more linked package(s) on remote server, delimited by semicolons. Naming conventions are those of remote system.
<linkRequestor>	Optional	0 - 1	String (20), variable	Name or TSO user ID of package link requestor.
<linkTime/>	Optional	0 - 1	Time, hhmmss	Link time
<package>	Optional	0 - 1	String (10), fixed	ZMF fixed-format package name for local package.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of `<package>`.

Exhibit 3-51. PACKAGE PKG_LINK LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<packageLevel>	Optional	0 - 1	String (1)	Code for package complexity or level in hierarchy of local package. Values: **1** = Simple package **2** = Complex package **3** = Super package **4** = Participating package
<packageStatus>	Optional	0 - 1	String (1)	Code for status of local package in lifecycle. Values: **1** = Approved **2** = Backed out **3** = Baselined **4** = Complex/super pkg closed **5** = Deleted (memo delete) **6** = Development **7** = Distributed **8** = Frozen **9** = Installed **A** = Complex/super pkg open **B** = Rejected **C** = Temporary change cycle completed
<packageType>	Optional	0 - 1	String (1)	Code for package install type of local package. Values: **1** = Planned permanent **2** = Planned temporary **3** = Unplanned permanent **4** = Unplanned temporary
<sourceLinkIpAddress>	Optional	0 - 1	String (255), variable	Network IP address for remote server where linked package resides. ***NOTE:*** ZMF stores address as provided by external link management software. May contain server name known to that software instead of an IP address.
<sourceLinkPortid>	Optional	0 - 1	String (8), variable	Network port ID for remote server where linked package resides.

List Package Library Types - LIBTYPE PKG LIST

You can retrieve library type specifications for a package using the Serena XML package library type list function. These specifications are defined separately by your ChangeMan ZMF administrator.

The Serena XML service/scope/message tags and attributes for messages to *list* package library type records are:

```
<service name="LIBTYPE">
<scope name="PKG">
<message name="LIST">
```

These tags appear in both requests and replies.

The service name is "`libtype`", not "`package`", because XML Services calls the low-level library type management service in ChangeMan ZMF to perform most tasks associated with this function. The scope name, "`pkg`", identifies this message as a package-level service.

LIBTYPE PKG LIST — Requests

You can request specifications for one or more library types defined for a named package. To retrieve all library type specifications for the package, no library type name is required. Specifications for an explicitly named library can also be requested. Filtering by DB2 library type is an additional option.

The following example shows how you might code a request to list all library types for a package that are not DB2 libraries. Data structure details for the `<request>` data element appear in *Exhibit 3-52*.

Example XML —LIBTYPE PKG LIST Request

```
<?xml version="1.0"?>
<service name="LIBTYPE">
 <scope name="PKG">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>IMSQ000012</package>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-52. LIBTYPE PKG LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), variable	ZMF application name. Same as first 4 bytes of `<package>`. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as `<package>` substitute. Use `<package>` instead of `<applName>` & `<packageId>`.
<isDb2LibType>	Optional	0 - 1	String (1)	**Y** = Include only DB2 libraries. **N** = Omit all DB2 libraries. ***NOTE:*** Omit tag or use asterisk (*) wildcard to request both DB2 and non-DB2 library types.
<libType>	Optional	0 - 1	String (3), variable	Name of specific library type to list. ***NOTE:*** Omit tag or use asterisk (*) wildcard to request all library types.
<package>	Optional	0 - 1	String (10), fixed	Fixed-format ZMF name of package for which library type info is requested.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of `<package>`. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.

LIBTYPE PKG LIST — Replies

The reply message listing package library types returns zero to many `<result>` data elements. Each `<result>` tag contains specifications for one library type defined for the named package.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

The following example shows what a Serena XML package library list reply message might look like. Data structure details for the `<result>` tag appear in *Exhibit 3-53*.

Example XML — LIBTYPE PKG LIST Reply

```
<?xml version="1.0"?>
<service name="LIBTYPE">
 <scope name="PKG">
  <message name="LIST">
   <result>
    <package>IMSQ000012</package>
    <applName>IMSQ</applName>
    <packageId>000012</packageId>
    <libType>CPC</libType>
    <likeType>1</likeType>
    <isPdseLibType>N</isPdseLibType>
    <chkOutComponentGenDesc>N</chkOutComponentGenDesc>
    <chkOutActivityFile>N</chkOutActivityFile>
    <deferStageLibCreation>Y</deferStageLibCreation>
    <includeUtilityInfo>N</includeUtilityInfo>
    <libTypeDesc>Copybooks common</libTypeDesc>
    <isImsLibType>N</isImsLibType>
    <isDb2LibType>N</isDb2LibType>
    <ddlSqlSubType>N</ddlSqlSubType>
    <storedProcSubType>N</storedProcSubType>
    <triggerSubType>N</triggerSubType>
    <bindControlSubType>N</bindControlSubType>
    <packageBindControlSubType>N</packageBindControlSubType>
    <sqlStoredProcDefinition>N</sqlStoredProcDefinition>
   </result>
.
.
.
   <response>
    <statusMessage>CMN8600I - The package library type list is complete.</
statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8600</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-53. LIBTYPE PKG LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), variable	ZMF application name. Same as first 4 bytes of `<package>`.

Exhibit 3-53. LIBTYPE PKG LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<apsDevLib>	Optional, APS only	0 - 1	String (44), variable	Name of APS development library associated with this ZMF library type. ***NOTE:*** If `<isApsLibType>` is Y, this tag is required.
<apsEntity>	Optional, APS only	0 - 1	String (8), variable	Name of APS entity type associated with this ZMF library type. ***NOTE:*** If `<isApsLibType>` is Y, this tag is required.
<bindControlSubType>	Optional, DB2 only	0 - 1	String (1)	**Y** = Yes, bind control library subtype **N** = No, not bind control library subtype
<chkOutActivityFile>	Optional	0 - 1	String (1)	**Y** = Yes, copy component to activity file at checkout **N** = Don't make activity file copy ***NOTE:*** Tag `<chkOutActivityFile>` also required if value is Y.
<chkOutComponentGenDesc>	Optional	0 - 1	String (1)	**Y** = Yes, copy component general description to staging change description at checkout **N** = No, leave component change description blank in staging at checkout
<db2SqlTerminationChar>	Optional, DB2 only	0 - 1	String (1)	DB2 SQL sentence termination character.
<dbrmSubType>	Optional	0 - 1	String (1)	**Y** = Yes, DBRM subtype **N** = Not DBRM subtype
<ddlSqlSubType>	Optional, DB2 only	0 - 1	String (1)	**Y** = Yes, SQL DDL library subtype **N** = No, not SQL DDL library subtype
<deferStageLibCreation>	Optional	0 - 1	String (1)	**Y** = Yes, defer allocation of library type in staging library to first component checkout **N** = No, don't defer library allocation
<imsEntity>	Optional, IMS only	0 - 1	String (1)	Code for IMS entity type associated with this ZMF library type. Values: **1** = PSB source **2** = DBD source **3** = MFS source **4** = PSB target **5** = DBD target **6** = FMT target **7** = REF target ***NOTE:*** If `<isImsLibType>` is Y, this tag is required.
<includeUtilityInfo>	Optional	0 - 1	String (1)	**Y** = Yes, include scratch/rename utility info with library type. **N** = No, omit scratch/rename info.

Exhibit 3-53. LIBTYPE PKG LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<isApsLibType>	Optional	0 - 1	String (1)	**Y** = Yes, this is APS library type **N** = No, not APS library type
<isDb2LibType>	Optional	0 - 1	String (1)	**Y** = Yes, this is DB2 library type **N** = No, not DB2 library type
<isHfsLibType>	Optional	0 - 1	String (1)	**Y** = Yes, this is HFS library type **N** = No, not HFS library type
<isImsLibType>	Optional	0 - 1	String (1)	**Y** = Yes, this is IMS library type **N** = No, not IMS library type ***NOTE:*** Tag `<imsEntity>` also required if value is Y.
<isPdseLibType>	Optional	0 - 1	String (1)	**Y** = Yes, this is PDSE library type **N** = No, not PDSE library type
<libType>	Required	1	String (3), variable	Name of library type for which specifications are reported.
<libTypeDesc>	Optional	0 - 1	String (44), variable	Text description of library type.
<librarySequenceNo>	Optional	0 - 1	Integer	Library sequence number
<likeType>	Optional	0 - 1	String (1)	Code for “like-library” type assigned to library type name. Values: **1** = Like Copy Library **2** = Like Load Library **3** = Like Other Library **4** = Like PDS Library **5** = Like Source Library 6 = Like Ncal Library 7 = Like Object Library ***NOTE:*** Tag `<targetLoadLibType>` also required if value is 5.
<package>	Required	1	String (10), fixed	ZMF fixed-format package name.
<packageBindControlSubType>	Optional, DB2 only	0 - 1	String (1)	**Y** = Yes, package bind control subtype **N** = No, not package bind ctrl subtype
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of `<package>`.
<sqlStoredProcDefinition>	Optional, DB2 only	0 - 1	String (1)	**Y** = Yes, SQL stored proc definition **N** = No, not SQL stored proc definition
<storedProcSubType>	Optional, DB2 only	0 - 1	String (1)	**Y** = Yes, DB2 stored procedure subtype **N** = No, not stored procedure subtype
<targetActivityFile>	Optional	0 - 1	String (3), variable	Name of target activity file library type associated with this library. ***NOTE:*** This tag is required if value in `<chkOutActivityFile>` is Y.

Exhibit 3-53. LIBTYPE PKG LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<targetLoadLibType>	Optional	0 - 1	String (3), variable	Name of "like-Load" target library type associated with this source library. ***NOTE:*** This tag is required if value in `<likeLibType>` is 5.
<triggerSubType>	Optional, DB2 only	0 - 1	String (1)	**Y** = Yes, DB2 trigger library subtype **N** = No, not DB2 trigger library subtype

List Package Promotion History - PACKAGE PRM_HIST LIST

Promotion history records for a *package as a whole* can be listed using the package promotion history list function. Promotion history listings for *components* in a package require a different Serena XML function. (See *Package Promoted Component List - PACKAGE PRM_CMP LIST*.)

The Serena XML service/scope/message names for a package promotion history list message are:

```
<service name="PACKAGE">
<scope name="PRM_HIST">
<message name="LIST">
```

These tags appear in both requests and replies.

PACKAGE PRM_HIST LIST — Request

This function supports two promotion history request types:

- ***All Promotion Actions at All Sites*** — Name the desired package in the `<package>` tag and enter a "1" in the `<requestType>` tag. Returns a complete history of all promotion and demotion actions taken against the named package on all sites.
- ***Current Promotion Status at Selected Site(s)*** — Name the desired package in the `<package>` tag and enter a "2" in the `<requestType>` tag. Specify a site of interest in the `<promotionSiteName>` tag; for all sites, omit this tag or enter a "match-all" (asterisk) wildcard character. Returns the current promotion status of the named package at the specified sites. If the site has prior promotion history, then that information is returned. If the only history associated with the site is the 'submitted' request, then no information is returned for the site.

To further narrow either type of request, specify a promotion level or site of interest. You can also filter promotion status and promotion action using appropriate yes/no flag tags.

Note

Yes/no flags for status and action filtering each take default values as a group. The default changes based on whether or not you enter explicit values in these tags, as follows:

- If ***no*** status flag in a group has an explicitly typed value, the default for all tags in that group is “Y”.
- If ***any*** status flag in a group has an explicitly typed value, the default for the remaining tags in the group is “N”.

The following example shows how you might code a request to list the full package promotion history for all promotion sites where a “selective promote” or “selective demote” is the most recent promotion action taken for a package. Data structure details follow in *Exhibit 3-54*.

Example XML — PACKAGE PRM_HIST LIST Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="PRM_HIST">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>IMSQ000012</package>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-54. PACKAGE PRM_HIST LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), variable	ZMF application name. Same as first 4 bytes of `<package>`.
<firstPromotion>	Optional	0 - 1	String (1)	**Y** = Yes, include first promotes **N** = No, omit first promotes ***NOTE:*** Member of promotion action flag tag group. If *no* tag in group has explicit value, default value is Y. If *any* tag in group has explicit value, default is N.

Exhibit 3-54. PACKAGE PRM_HIST LIST <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<fullDemotion>	Optional	0 - 1	String (1)	**Y** = Yes, include full demotes **N** = No, omit full demotes ***NOTE:*** Member of promotion action flag tag group. If *no* tag in group has explicit value, default value is Y. If *any* tag in group has explicit value, default is N.
<fullPromotion>	Optional	0 - 1	String (1)	**Y** = Yes, include full promotes **N** = No, omit full promotes ***NOTE:*** Member of promotion action flag tag group. If *no* tag in group has explicit value, default value is Y. If *any* tag in group has explicit value, default is N.
<jobBuilt>	Optional	0 - 1	String (1)	**Y** = Yes, include built jobs **N** = No, omit built jobs ***NOTE:*** Member of promotion status flag tag group. If *no* tag in group has explicit value, default value is Y. If *any* tag in group has explicit value, default is N.
<jobCompleted>	Optional	0 - 1	String (1)	**Y** = Yes, include completed jobs **N** = No, omit completed jobs ***NOTE:*** Member of promotion status flag tag group. If *no* tag in group has explicit value, default value is Y. If *any* tag in group has explicit value, default is N.
<jobFailed>	Optional	0 - 1	String (1)	**Y** = Yes, include failed jobs **N** = No, omit failed jobs ***NOTE:*** Member of promotion status flag tag group. If *no* tag in group has explicit value, default value is Y. If *any* tag in group has explicit value, default is N.
<jobSubmitted>	Optional	0 - 1	String (1)	**Y** = Yes, include submitted jobs **N** = No, omit submitted jobs ***NOTE:*** Member of promotion status flag tag group. If *no* tag in group has explicit value, default value is Y. If *any* tag in group has explicit value, default is N.
<package>	Required	1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of `<package>`.
<promotionLevel>	Optional	0 - 1	String (2), variable	Numeric promotion level for which promotion history is requested.

Exhibit 3-54. PACKAGE PRM_HIST LIST <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<promotionName>	Optional	0 - 1	String (8), variable	ZMF promotion level nickname for which promotion history is requested. ***NOTE:*** Accepts standard wildcards & patterns using question mark (?) & asterisk (*).
<promotionSiteName>	Optional	0 - 1	String (8), variable	ZMF name of promotion site for which promotion history or status requested. ***NOTE:*** Accepts standard wildcards & patterns using question mark (?) & asterisk (*). ***NOTE:*** Omit tag or use asterisk (*) wildcard to list all promotion sites, regardless of request type.
<requestType>	Optional	0 - 1	String (255, variable	Code for type of promotion history list requested. Values: **1** = Full promotion history (default) **2** = Site promotion status **3** = Full promotion history, including site locks.
<selectiveDemotion>	Optional	0 - 1	String (1)	**Y** = Yes, include selective demotes **N** = No, omit selective demotes ***NOTE:*** Member of promotion action flag tag group. If *no* tag in group has explicit value, default value is Y. If *any* tag in group has explicit value, default is N.
<selectivePromotion>	Optional	0 - 1	String (1)	**Y** = Yes include selective promotes **N** = No, omit selective promotes ***NOTE:*** Member of promotion action flag tag group. If *no* tag in group has explicit value, default value is Y. If *any* tag in group has explicit value, default is N.
<siteLock>	Optional	0 - 1	String (1)	**Y** = site is locked **N** = site is not locked

PACKAGE PRM_HIST LIST — Reply

The XML reply to a promotion history list request includes zero to many `<result>` tags. For full promotion history lists (i.e., the value in `<requestType`> is “1”), each `<result>` contains a package promotion or demotion record for a particular site and level. For promotion site status lists (i.e., the value in `<requestType>` is “2”), each `<result>` contains current package promotion status for a site.

A standard `<response>` tag follows the last `<result>` tag, if any, to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last tag returned in the reply message, the `<response>` tag may serve as an end-of-list marker.

An example XML reply to a package promotion history list request appears below. Data structure details for the `<result>` tag follow in *Exhibit 3-55*.

Example XML — PACKAGE PRM_HIST LIST Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="PRM_HIST">
  <message name="LIST">
   <result>
    <package>IMSQ000012</package>
    <applName>IMSQ</applName>
    <packageId>000012</packageId>
    <promotionSiteName>SERT8</promotionSiteName>
    <promotionLevel>10</promotionLevel>
    <promotionName>C001AUT</promotionName>
    <promoter>USER24</promoter>
    <promotionDate>20081019</promotionDate>
    <promotionTime>201225</promotionTime>
    <fullPromotion>Y</fullPromotion>
    <fullDemotion>N</fullDemotion>
    <selectivePromotion>N</selectivePromotion>
    <selectiveDemotion>N</selectiveDemotion>
    <firstPromotion>N</firstPromotion>
    <jobSubmitted>N</jobSubmitted>
    <jobCompleted>Y</jobCompleted>
    <jobFailed>N</jobFailed>
    <jobBuilt>N</jobBuilt>
    <componentCount>0000020</componentCount>
   </result>
   <result>
    <package>IMSQ000012</package>
    <applName>IMSQ</applName>
    <packageId>000012</packageId>
    <promotionSiteName>SERT8</promotionSiteName>
    <promotionLevel>10</promotionLevel>
    <promotionName>C001AUT</promotionName>
    <promoter>USER24</promoter>
    <promotionDate>20081019</promotionDate>
    <promotionTime>201633</promotionTime>
    <fullPromotion>N</fullPromotion>
    <fullDemotion>Y</fullDemotion>
    <selectivePromotion>N</selectivePromotion>
    <selectiveDemotion>N</selectiveDemotion>
    <firstPromotion>N</firstPromotion>
    <jobSubmitted>N</jobSubmitted>
    <jobCompleted>Y</jobCompleted>
    <jobFailed>N</jobFailed>
    <jobBuilt>N</jobBuilt>
    <componentCount>0000019</componentCount>
   </result>
.
.
```

```
.
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-55. PACKAGE PRM_HIST LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of `<package>`.
<componentCount>	Optional	0 - 1	Integer (7), variable	Number of components included in promotion action.
<firstPromotion>	Optional	0 -1	String (1)	**Y** = Yes, action is first promote **N** = No, not first promote
<fullDemotion>	Optional	0 -1	String (1)	**Y** = Yes, action is full demote **N** = No, not full demote
<fullPromotion>	Optional	0 -1	String (1)	**Y** = Yes, action is full promote **N** = No, not full promote
<jobBuilt>	Optional	0 -1	String (1)	**Y** = Yes, promotion job built **N** = No, job not built
<jobCompleted>	Optional	0 -1	String (1)	**Y** = Yes, promotion job completed **N** = No, job not completed
<jobFailed>	Optional	0 -1	String (1)	**Y** = Yes, promotion job failed **N** = No, job did not fail
<jobSubmitted>	Optional	0 -1	String (1)	**Y** = Yes, promotion job submitted **N** = No, job not submitted
<package>	Optional	0 -1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of `<package>`.
<promoter>	Optional	0 - 1	String (8), variable	TSO user ID of package promoter for reported promotion action.
<promotionDate>	Optional	0 - 1	Date, yyyymmdd	Date of reported promotion action.
<promotionLevel>	Optional	0 - 1	String (2), variable	Numeric promotion level for which promotion action & status are reported.
<promotionName>	Optional	0 - 1	String (8), variable	ZMF nickname of promotion level for which action & status are reported

Exhibit 3-55. PACKAGE PRM_HIST LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<promotionSiteName>	Optional	0 - 1	String (8), variable	ZMF name of promotion site for which promotion action & status are reported.
<promotionSuccessDate>	Optional	0 - 1	Date, yyyymmdd	Date of successful promotion.
<promotionSuccessTime>	Optional	0 - 1	Time, hhmms	Time of successful promotion.
<promotionTime>	Optional	0 - 1	Time, hhmmss	Time of reported promotion action.
<selectiveDemotion>	Optional	0 -1	String (1)	**Y** = Yes, action is selective demote **N** = No, not selective demote
<selectivePromotion>	Optional	0 -1	String (1)	**Y** = Yes, action is selective promote **N** = No, not selective promote
<siteLock>	Optional	0 - 1	String (1)	**Y** = Site is locked. **N** = Site is not locked.

Package Promoted Component List - PACKAGE PRM_CMP LIST

List the promotion history of all *components* in a named package using the Serena XML component promotion history list. A promotion history list for the *package as a whole* requires a different Serena XML function. (See *List Package Promotion History - PACKAGE PRM_HIST LIST*.)

The Serena XML service/scope/message names for a component promotion history list are:

```
<service name="PACKAGE">
<scope name="PRM_CMP">
<message name="LIST">
```

These tags appear in both requests and replies.

PACKAGE PRM_CMP LIST — Request

This component promotion history function requests all component promotion history records for a specific package at a specific promotion site and level. No further filtering options exist.

The following example shows how you might code a component promotion history request in Serena XML. Data structure details follow the example in *Exhibit 3-56*.

Example XML — PACKAGE PRM_CMP LIST Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="PRM_CMP">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>IMSQ000012</package>
    <promotionSiteName>SERT8</promotionSiteName>
    <promotionLevel>10</promotionLevel>
    <promotionName>*</promotionName>
    <shortSelectList>Y</shortSelectList>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-56. PACKAGE PRM_CMP LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of `<package>`.
<package>	Required	1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of `<package>`.
<promotionLevel>	Required	1	Integer (2), variable	Numeric promotion level for which promotion action & status are requested.
<promotionName>	Required	1	String (8), variable	ZMF nickname of promotion level for which action & status are requested
<promotionSiteName>	Required	1	String (8), variable	ZMF name of promotion site for which promotion action & status are requested.
<shortSelectList>	Required	1	String (1)	**Y** = Limit the selection list for selective promotion to package components that are not currently promoted to the target level, including components that may have been re-staged, newly activated into the package, or overlaid by promotion of another package. **N** = Display all package components on the selective promotion selection list.

PACKAGE PRM_CMP LIST — Reply

The reply message for the Serena XML component promotion history list function returns zero to many `<result>` tags. Each `<result>` contains promotion action and status information for one component in the named package at the named promotion site and level. If no components have been promoted to that site and level, no `<result>` tags are returned.

A standard `<response>` tag follows the last `<result>` tag to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last tag returned in the reply message, the `<response>` tag also serves as an end-of-list marker.

An example Serena XML reply to a component promotion history list request follows. Data structure details for the `<result>` tag appear in *Exhibit 3-57*.

Example XML — Package Prm_Cmp List Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="PRM_CMP">
  <message name="LIST">
   <result>
    <package>IMSQ000012</package>
    <applName>IMSQ</applName>
    <packageId>000012</packageId>
    <component>IM2Q101</component>
    <componentType>DBB</componentType>
    <stagedDate>20081019</stagedDate>
    <stagedTime>200843</stagedTime>
    <stager>USER24</stager>
    <componentStatus>0</componentStatus>
    <promotionSiteName>SERT8</promotionSiteName>
    <promotionName>C001AUT</promotionName>
    <promotionLevel>10</promotionLevel>
    <promoter>USER24</promoter>
    <promotionDate>20081212</promotionDate>
    <promotionTime>100135</promotionTime>
    <isComponentRestaged>N</isComponentRestaged>
    <cleanupComponent>N</cleanupComponent>
    <isComponentOverlayed>N</isComponentOverlayed>
    <isComponentDeleted>N</isComponentDeleted>
   </result>
.
.
.
    <response>
    <statusMessage>CMN8600I - The Promotion list is complete.</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8600</statusReasonCode>
   </response>
  </message>
  </scope>
</service>
```

Exhibit 3-57. PACKAGE PRM_CMP LIST <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF package ID number. Same as last 6 bytes of `<package>`.
<cleanupComponent>	Optional	0 - 1	String (1)	**Y** = Yes, clean up component **N** = No, don't clean up
<component>	Optional	0 - 1	String (256), variable	Name of component for which promotion action and status are reported. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentStatus>	Optional	0 - 1	String (1)	Status code for listed component. Values: **0** = Active **1** = Approved **2** = Checked out **3** = Demoted **4** = Frozen **5** = Inactive **6** = Incomplete **7** = Promoted **8** = Refrozen **9** = Rejected **A** = Remote promoted **B** = Submitted **C** = Unfrozen
<componentType>	Optional	0 - 1	String (3), variable	Library type for listed component.
<isComponentDeleted>	Optional	0 - 1	String (1)	**Y** = Yes, component deleted **N** = No, not deleted
<isComponentOverlayed>	Optional	0 - 1	String (1)	**Y** = Yes, component is overlaid **N** = No, not overlaid
<isComponentRestaged>	Optional	0 - 1	String (1)	**Y** = Yes, component is restaged **N** = No, not restaged
<package>	Optional	0 - 1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF application name. Same as first 4 bytes of `<package>`.
<promoter>	Optional	0 - 1	String (8), variable	TSO user ID of component promoter.

Exhibit 3-57. PACKAGE PRM_CMP LIST <result> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<promotionDate>	Optional	0 - 1	Date, yyyymmdd	Date component promoted to this site and level.
<promotionLevel>	Optional	0 - 1	Integer (2), variable	Numeric promotion level for which promotion action & status are reported.
<promotionName>	Optional	0 - 1	String (8), variable	ZMF nickname of promotion level for which action & status are reported
<promotionSiteName>	Optional	0 - 1	String (8), variable	ZMF name of promotion site for which promotion action & status are reported.
<promotionTime>	Optional	0 - 1	Time, hhmmss	Time component promoted to this site and level, 24-hour format.
<stagedDate>	Optional	0 - 1	Date, yyyymmdd	Date component staged.
<stagedTime>	Optional	0 - 1	Time, hhmmss	Time component staged, 24-hour format.
<stager>	Optional	0 - 1	String (8), variable	TSO user ID of developer who staged component.

List Reasons for Backout or Revert - PACKAGE REASONS LIST

The package reasons list function lists backout or revert reasons for a specific package.

The Serena XML service/scope/message tags for a package reasons list message are:

```
<service name="PACKAGE">
<scope name="REASONS">
<message name="LIST">
```

These tags appear in both requests and replies.

PACKAGE REASONS LIST Requests

An example of how you might code a Serena XML request to list backout or revert reasons appears below. Data structure details for the `<request>` tag appear in *Exhibit 3-58*.

Example XML — PACKAGE REASONS LIST Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="REASONS">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000012</package>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 3-58. PACKAGE REASONS LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<fromDate>	Optional	0 - 1	Date, yyyymmdd	Start date in desired range of backout/revert dates.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6)	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<reasonType	Optional	0 - 1	String (1)	Reason type: **B** = Backout **R** = Revert
<siteName>	Optional	0 - 1	String (8), variable	Site name.
<toDate>	Optional	0 - 1	Date, yyyymmdd	End date in desired range of backout/revert dates.
<updater>	Optional	0 - 1	String (8)	TSO user ID of last user to back out or revert the package.

PACKAGE REASONS LIST — Reply

The reply message for the Serena XML package reasons list function returns zero to one `<result>` tags. If there are no backout or revert reasons for the requested package, no `<result>` tags are returned.

A standard `<response>` tag follows the last `<result>` tag to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

An example Serena XML reply to a package reasons list request follows. Data structure details for the `<result>` tag appear in *Exhibit 3-59*.

Example XML — Package Reasons List Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="REASONS">
  <message name="LIST">
   <result>
    <package>ACTP000012</package>
    <applName>ACTP</applName>
    <packageId>000012</packageId>
    <reasonType>R</reasonType>
    <siteName>SERT8</siteName>
    <updater>USER109</updater>
    <updateDate>20120718</updateDate>
    <updateTime>073744</updateTime>
    <reasons>reverted</reasons>
    <reason01>reverted</reason01>
   </result>
   <response>
    <statusMessage>CMN8600I - LIST Reasons service completed.</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8600</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 3-59. PACKAGE REASONS LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<package>	Optional	0 -1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6)	ZMF package ID number. Same as last 6 bytes of package name.
<reason01>	Optional	0 - 1	String (72), variable	Reason line - 1.
<reason02>	Optional	0 - 1	String (72), variable	Reason line - 2.
<reason03>	Optional	0 - 1	String (72), variable	Reason line - 3.

Exhibit 3-59. PACKAGE REASONS LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<reason04>	Optional	0 - 1	String (72), variable	Reason line - 4.
<reason05>	Optional	0 - 1	String (72), variable	Reason line - 5.
<reason06>	Optional	0 - 1	String (72), variable	Reason line - 6.
<reason07>	Optional	0 - 1	String (72), variable	Reason line - 7.
<reason08>	Optional	0 - 1	String (72), variable	Reason line - 8.
<reason09>	Optional	0 - 1	String (72), variable	Reason line - 9.
<reasonType	Optional	0 - 1	String (1)	Reason type: **B** = Backout **R** = Revert
<reasons>	Optional	0 - 9	String (72), variable	Reasons lines 1 - 9. ***NOTE:*** The <reasons> tag is deprecated and contains the same information as <reason01> -- <reason09>.
<siteName>	Optional	0 - 1	String (8), variable	Site name.
<updateDate>	Optional	0 - 1	Date, yyyymmdd	Date that the package was backed out or reverted.
<updateTime>	Optional	0 - 1	Time, hhmmss	Time that the package was backed out or reverted.
<updater>	Optional	0 - 1	String (8)	TSO user ID of user who backed out or reverted the package.

Component Management 4

Component management tasks supported by Serena XML fall into the following categories:

- ***Component Lifecycle Tasks*** — Development tasks that comprise or enable a significant step in the component lifecycle. Typical commands are *checkout, checkin, browse, compare, build, recompile, relink, lock, unlock, scratch,* and *rename*.
- ***Component Staging Version Management*** — Control and information retrieval tasks for multiple component versions maintained concurrently in the staging/development library. Such commands include *list* and *retrieve*.
- ***Component Information Management Tasks*** — Tasks that retrieve or manage descriptive metadata or control information about a component, such as component descriptions or staging version change descriptions. Typical commands include *list*.
- ***Component Security Tasks*** — Tasks that validate or manage component access security. Typical commands are *check* and *list*.

Component Management Message Syntax

Identifying Component Messages

All Serena XML component management messages have syntax that tells ChangeMan ZMF to perform a task against a component rather than some other object. In all such messages, the `name` attribute in the `<service>` takes the value "`cmponent`", as follows:

```
<service name="CMPONENT">
```

Tip

Note the abbreviated spelling of "`cmponent`" in the `name` attribute! This value is truncated because ChangeMan ZMF limits `name` attributes to eight bytes in length.

In addition, a component management task takes a value in the `name` attribute of the `<scope>` tag that is consistent with work at the level of individual components. For example, any Serena XML component message with a `name` attribute of "`service`" in the `<scope>` tag is a component-only, component-level task. For example:

```
<service name="cmponent">
<scope name="service">
```

Other purely component-level `name` attributes for the `<scope>` tag include "`ssv_ver`", "`history`", "`gen_desc`", "`chg_desc`", and the like.

Component tasks performed at a higher level of aggregation — such as the package level — indicate their higher-level scope in the `<scope>` tag. For example, attribute values such as "`pkg_src`" or "`pkg_lod`" broaden the scale of a component management function to include the components of an entire package as a group. At the same time, these attributes exclude any shared components that reside in packages not named in the request. This behavior classifies such requests as package-level component tasks rather than component-only tasks — even though, for technical reasons, they are performed by the low-level component service.

For the purposes of this manual, then, syntax such as the following identifies a package-level component function rather than a component-only function:

```
<service name="CMPONENT">
<scope name="PKG_SRC">
```

Such tasks are discussed in the package management topic.

COMPONENT LIFECYCLE TASKS

The following component lifecycle tasks are supported by Serena XML for general use:

- *Check Out a Component - CMPONENT SERVICE CHECKOUT*
- *Component Service Checkin - CMPONENT SERVICE CHECKIN*
- *Check Designated Build Procedures - CMPONENT APL_DPRC CHECK*
- *Find Designated Build Procedure - CMPONENT APL_DPRC FIND*
- *List Designated Build Procedures - CMPONENT APL_DPRC LIST*
- *List Global Designated Build Procedures - CMPONENT GBL_DPRC LIST*
- *Component Service Build - CMPONENT SERVICE BUILD*
- *Recompile a Component - CMPONENT SERVICE RECOMP*
- *Relink a Component - CMPONENT SERVICE RELINK*
- *Browse a Component - CMPONENT SERVICE BROWSE*
- *Compare Components - CMPONENT SERVICE COMPARE*
- *Rename a Component - CMPONENT SERVICE RENAME*
- *Scratch a Component - CMPONENT SERVICE SCRATCH*
- *Lock or Unlock a Component - CMPONENT SERVICE LOCK/UNLOCK*
- *List Load Module Subroutines - CMPONENT LOD_SUBR LISTt*
- *List Copybook Names in Source - CMPONENT SRC_INCL LIST*

Check Out a Component - CMPONENT SERVICE CHECKOUT

The Serena XML service/scope/message tags and attributes for component *checkout* messages are:

```
<service name="CMPONENT">
<scope name="SERVIVE">
<message name="CHECKOUT">
```

These tags appear in both requests and replies.

CMPONENT SERVICE CHECKOUT Requests

Serena XML permits concurrent checkout of one or many components. Checkout options apply equally to all named components in the checkout request. For example, all components must have the same library type, must be checked out from the same source (baseline or promotion), and must be checked out to the same target (staging or a personal development library).

The example below shows how you might code a checkout request in Serena XML. In this example, a component is checked out from level 001 of the baseline library into a personal library.

As in all XML examples in this manual, items in bold type are required. A selection of optional subtags is shown in regular type. Nested subtags of a higher-level complex tag are indented relative to that tag. Repeatable tags appear twice for illustration.

Data structure details for the component checkout `<request>` tag appear in *Exhibit 4-1*, following the example.

Example XML — CMPONENT SERVICE CHECKOUT Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="SERVICE">
  <message name="CHECKOUT">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>TES5000003</package>
    <componentType>CPY</componentType>
    <chkOutSourceLocation>4</chkOutSourceLocation>
    <chkOutMode>O</chkOutMode>
    <basePromoLibLevel>001</basePromoLibLevel>
    <chkOutTargetLocation>1</chkOutTargetLocation>
    <personalLibStorageMeans>6</personalLibStorageMeans>
    <personalLib>USER24.SETQUERY.WORKLOAD</personalLib>
    <jobCard01>//XMLX029B JOB (RWM,T),'DUMP',CLASS=A,MSGCLASS=A</jobCard01>
    <jobCard02>//* JOBCARD2</jobCard02>
       <jobCard03>//* JOBCARD3</jobCard03>
       <jobCard04>//* JOBCARD4</jobCard04>
       <listCount>0001</listCount>
       <component>ACPCPY00</component>
      </request>
     </message>
    </scope>
   </service>
```

Exhibit 4-1. CMPONENT SERVICE CHECKOUT <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), fixed	ZMF application name. Same as first 4 bytes of `<package>` tag. ***NOTE:*** Trailing blanks required.
<basePromoLibLevel>	Required	1	Integer (3), variable	Baseline or promotion library level from which component is checked out. Allowed values (must be positive): • Baseline checkouts - 0 to 99 • Promotion checkouts - 1 to 999 ***NOTES:*** • `<chkOutSourceLocation>` is required with this tag to determine whether value is read as a “negative” baseline level or a “positive” promotion level. • If checkout is from baseline (that is, if `<chkOutSourceLocation>` = 4), default value is 0. • If baseline level for a checkout is not zero, the checkout must be performed in batch mode (that is, with <chkOutMode> = B).
<chkOutMode>	Required	1	String (1), fixed	Code for component checkout processing mode. Valid values: **O** = Online checkout (letter O) **B** = Batch checkout ***NOTES:*** • Batch checkout required (value must be B) if personal library for checkout is Librarian or Panvalet (that is, if `<personalLibStorageMeans>` = 4 or 5). • Batch checkout required (value must be B) if checking out from a backlevel baseline (that is, if <basePromoLib-Level> is not zero and <chkOutSour-ceLocation> = 4).

Exhibit 4-1. CMPONENT SERVICE CHECKOUT <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<chkOutSourceLocation>	Required	1	String (1), fixed	Code for location component is checked out ***from***. Valid values: **3** = Checkout from promotion **4** = Checkout from baseline **8** = ERO component Checkout from a prior release
<chkOutTargetLocation>	Required	1	String (1), fixed	Code for location component is checked out ***to***. Valid values: **1** = Checkout to personal development library **2** = Checkout to staging library
<component>	Required	1 - ∞	String (256), variable	ZMF name of component to check out. Repeatable to accommodate multi-component checkouts. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root. ***NOTE:*** Number of instances must equal value in the `<listCount>` tag.
<componentType>	Required	1	String (3), fixed	Must be valid ZMF library type. Typical values: •COB •CPY •JCL •SRC •LOD
<jobCard01> . . . <jobCard04>	Optional	0 - 1 each	String (72), fixed	JCL statements needed to set job parameters, allocate data sets, & define library concatenations during checkout. If used, all four tags are required. Tags not needed for JCL should be coded as comment (//*). ***NOTE:*** Required for batch checkout — that is, if `<chkOutMode>` = B
<listCount>	Required	1	Integer	Number of components to be checked out. Must equal the number of `<component>` tags that follow.
<lockComponent>	Optional	0 - 1	String (1), fixed	**Y** = Yes, lock after checkout **N** = No, don't lock after checkout

Exhibit 4-1. CMPONENT SERVICE CHECKOUT <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<overlayPriorVersion>	Optional	0 - 1	String (1), fixed	**Y** = Yes, overlay preexisting component copy in package **N** = No, don't overlay preexisting component copy in package ***NOTE:*** This tag affects the active component copy currently residing in a change package. It does NOT affect any staged versions of that component created by ZMF's Save Staging Versions (SSV) feature.
<package>	Required	1	String (10), fixed	Fixed-format name of ZMF package where component resides. First 4 bytes correspond to `<applName>`. Final 6 bytes correspond to `<packageId>`. ***NOTE:*** See *ChangeMan ZMF User's Guide* for format of package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of `<package>` tag. ***NOTE:*** Leading zeroes required.
<personalLib>	Optional	0 - 1	String (44), variable	Name of personal development library for component checkout. ***NOTE:*** Required if checked out to personal development library – that is, if `<chkOutTargetLocation>` = 1.
<personalLibStorageMeans>	Optional	0 - 1	String (1), fixed	Code for data set organization of checkout target location. Values: **4** = CA-Librarian **5** = CA-Panvalet **6** = PDS **8** = Sequential **9** = PDSE **H** = HFS ***NOTE:*** Required if checked out to personal development library – that is, if `<chkOutTargetLocation>` = 1. ***NOTE:*** If value is 4 or 5, batch checkout required – that is, `<chkOutMode>` = B
<promotionSiteName>	Optional	0 - 1	String (8), variable	ZMF name of promotion site from which component is checked out. ***NOTE:*** Required if checked out *from* promotion – that is, if value in `<chkOutSourceLocation>` = 3.
<release>	Optional	0 - 1	String (8), variable	Name of ZMF release (ERO Option only).

Exhibit 4-1. CMPONENT SERVICE CHECKOUT <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<releaseArea>	Optional	0 - 1	String (8), variable	Name of the ZMF release area (ERO Option only).
<savePriorVersion>	Optional	0 - 1	String (1), fixed	**Y** = Yes, save staging version of preexisting component **N** = No, don't save staging version of preexisting component ***NOTE:*** This tag applies only if ZMF's Save Staging Versions (SSV) feature is installed. It has no effect on whether or not a component checked out from baseline will overlay a preexisting copy of that component in the package.
<suppressNotify>	Optional	0 - 1	String (1), fixed	**Y** = Suppress batch notification **N** = Allow batch notification
<userVariable01> . . . <userVariable05>	Optional	0 - 1 each	String (8), variable	Five 8-byte custom user variables for component checkout established by ZMF administrator.
<userVariable06> . . . <userVariable10>	Optional	0 - 1 each	String (72), variable	Five 72-byte custom user variables for component checkout established by ZMF administrator.

Tip

Tags: <userVariable01> to <userVariable10>: See topic "Custom V01-V10 Variables" in the ChangeMan ZMF Customization Guide.

CMPONENT SERVICE CHECKOUT Replies

No `<result>` data structure is returned in the component checkout reply message. However, the standard `<response>` data structure is returned to indicate the success or failure of the checkout request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Component Service Checkin - CMPONENT SERVICE CHECKIN

The Serena XML service/scope/message tags and attributes for component *checkin* messages are:

```
<service name="CMPONENT">
<scope name="SERVICE">
<message name="CHECKIN">
```

These tags appear in both requests and replies.

Batch Component Checkin Versus Online Component Staging

The Serena XML *checkin* function performs a subset of the “stage” functions of the ISPF user interface. Check-in simply copies files from a development location into a staged change package and updates the component status information — the first step of three in the staging flow. Check-in does not compile or link-edit the checked-in component. Neither does it log source-to-load and other relationships within a package or build any JCL install jobs.

By deferring many elements of the interactive “stage” function to a later time and a different XML function, the *checkin* function gains the advantage of speed. Speed is vital for batch-mode check-in requests that import a large number of components from other environments to ChangeMan ZMF. Up to 9999 components can be checked in to a ChangeMan ZMF change package via a single XML checkin request.

Tip

Batch component checkin for a large number of components should be specified in native Serena XML and submitted for execution via the SERXMLBC batch execution client. (See *Appendix B, “SERXMLBC – Executing Native XML Service Calls.”*) Source component name tags should be populated via a table-driven preprocessing script or similar automated means if the number of components is very large.

To complete the full “stage to development” process using Serena XML, the *checkin* function, if successful, should be followed by an XML request to *check* designated build procedures, which (if successful) should then be followed by an XML component *build* request. (See *“Check Designated Build Procedures - CMPONENT APL_DPRC CHECK”* and *“Component Service Build - CMPONENT SERVICE BUILD”* later in this chapter.)

CMPONENT SERVICE CHECKIN Requests

Serena XML permits concurrent checkin of one to many components. Checkin options apply equally to all named components in the checkin request. For example, all components must have the same library type, must be checked out from the same source location, and must be checked in to the same package.

The example below shows how you might code a Serena XML request to check in a source component from a personal library.

Data structure details for the component checkin `<request>` tag follow in *Exhibit 4-2*.

Example XML — CMPONENT SERVICE CHECKIN Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="SERVICE">
  <message name="CHECKIN">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>TES5000003</package>
    <component>ACPCPY00</component>
    <componentType>CPY</componentType>
    <chkInSourceLocation>1</chkInSourceLocation>
    <sourceStorageMeans>6</sourceStorageMeans>
    <sourceLib>USER24.SETQUERY.WORKLOAD</sourceLib>
    <changeDesc>TEST CMPONENT SERVICE CHECKIN</changeDesc>
    <listCount>0001</listCount>
    <targetComponent>ACPCPY00</targetComponent>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 4-2. CMPONENT SERVICE CHECKIN <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), fixed	ZMF application name. Same as first 4 bytes of `<package>` tag. ***NOTE:*** Trailing blanks required.
<changeDesc>	Optional	1	String (35)	Component change description to include with all newly checked in components.
<chkInSourceLocation>	Required	1	String (1)	Code for location from which component is checked in. Valid values: **1** = Checkin from development dataset **5** = Checkin from package **7** = Checkin from a temporary sequential dataset (for example, using ZDD Checkin) **E** = Edit from package library

Exhibit 4-2. CMPONENT SERVICE CHECKIN <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<component>	Optional	0 - 1	String (256), variable	Source component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root. ***NOTE:*** If checking in from a sequential file, this tag is required.
<componentType>	Required	1	String (3)	Library type to assign to checked-in component(s). Must be valid ZMF library type. Typical values: •COB •CPY •JCL •SRC •LOD
<listCount>	Required	1	Integer	Number of components to be checked in. Must immediately precede the first of one or more `<targetComponent>` tags. Value must equal the number of `<targetComponent>` tags that follow.
<lockAfterChkin>	Optional	1	String (1)	**Y** = Yes, lock component after checkin **N** = No, don't lock component after check in
<package>	Required	1	String (10)	ZMF fixed-format package name where component should reside.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of `<package>` tag. ***NOTE:*** Leading zeroes required.
<savePriorStagingVersion>	Optional	0 - 1	String (1)	**Y** = Yes, create staging version of preexisting component (if staging versions enabled) **N** = No, don't create a staging version of component
<sourceLib>	Optional	0 - 1	String (44)	Data set name of library holding component(s) to check in. NOTE: For HFS component, path and subdirectory where component resides.

Exhibit 4-2. CMPONENT SERVICE CHECKIN <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<sourceStorageMeans>	Optional	0 - 1	String (1)	Code for data set organization of checkout target location. Values: **4** = CA-Librarian **5** = CA-Panvalet **6** = PDS **8** = Sequential data set **9** = PDS/Extended **H** = HFS
<suppressNotify>	Optional	0 - 1	String (1)	**Y** = Yes, suppress notify messages **N** = No, don't suppress notify messages
<targetComponent>	Required	1 - ∞	String (256), variable	ZMF name of component to check in. Repeatable for multiple concurrent check-ins. Must be preceded by `<listCount>` tag. ***NOTE:*** Number of instances must equal value in `<listCount>`. ***NOTE:*** HFS components must also designate a target subdirectory for checkin in `<targetSubDirectory>`. For HFS components, all instances must belong to the same subdirectory.
<targetSubDirectory>	Optional	0 - 1	String (256), variable	Name of the target HFS subdirectory where components are to be checked in, prefixed by path from installation root (that is, path as it is defined in the baseline library). NOTE: Required for HFS component if `<useSourceLibSubDirectory>` = `N`. NOTE: Only one subdirectory is supported per request. For bulk checkins, all `<targetComponent>` tagsmust contain components that belong in the same subdirectory.
<userOption01> . . . <userOption20>	Optional	0 - 1 each	String (1), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 01 to 20 on the ISPF user options panel for component build.
<userOption0101> . . . <userOption0105>	Optional	0 - 1 each	String (1), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0101 to 0105 on the ISPF user options panel for component build.

Exhibit 4-2. CMPONENT SERVICE CHECKIN <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<userOption0201> . . . <userOption0203>	Optional	0 - 1 each	String (2), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0201 to 0203 on the ISPF user options panel for component build.
<userOption0301> . . . <userOption0303>	Optional	0 - 1 each	String (3), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0301 to 0303 on the ISPF user options panel for component build.
<userOption0401> . . . <userOption0403>	Optional	0 - 1 each	String (4), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0401 to 0403 on the ISPF user options panel for component build.
<userOption0801> . . . <userOption0805>	Optional	0 - 1 each	String (8), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0801 to 0805 on the ISPF user options panel for component build.
<userOption1001> . . . <userOption1002>	Optional	0 - 1 each	String (10), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 1001 to 1002 on the ISPF user options panel for component build.
<userOption1601> . . . <userOption1602>	Optional	0 - 1 each	String (16), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 1601 to 1602 on the ISPF user options panel for component build.
<userOption3401> . . . <userOption3402>	Optional	0 - 1 each	String (34), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 3401 to 3402 on the ISPF user options panel for component build.
<userOption4401> . . . <userOption4402>	Optional	0 - 1 each	String (44), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 4401 to 4402 on the ISPF user options panel for component build.
<userOption6401> . . . <userOption6405>	Optional	0 - 1 each	String (64), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 6401 to 6405 on the ISPF user options panel for component build.

Exhibit 4-2. CMPONENT SERVICE CHECKIN <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<userOption7201> . . . <userOption7205>	Optional	0 - 1 each	String (72), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 7201 to 7205 on the ISPF user options panel for component build.
<useSourceLibSubDirectory>	Optional	0 - 1	String (1)	Should target subdirectory & path match source library directory & path? **Y** = Use value in `<sourceLib>` for `<targetSubDirectory>`. **N** = Do not use `<sourceLib>` for `<targetSubDirectory>`; supply explicit value instead. NOTE: `<targetSubDirectory>` is required if value is `N`.

Component Service Checkin Reply

No `<result>` data structure is returned in the component checkin reply message. However, the standard `<response>` data structure is returned to indicate the success or failure of the checkin request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Check Designated Build Procedures - CMPONENT APL_DPRC CHECK

The Serena XML service/scope/message tags and attributes for a message to *check designated component build procedures* are:

```
<service name="CMPONENT">
<scope name="APL_DPRC">
<message name="CHECK">
```

The Serena XML *check* function performs a subset of the "stage" functions of the ISPF user interface. It checks for the existence of designated build procedures associated with a checked-in component — the second step of three in the staging flow. It does not copy files from a development location into a staged change package. It does not compile or link the checked-in component. Neither does it log source-to-load relationships within a package or build any JCL install jobs.

Tip

To complete the full “stage to development” cycle using Serena XML, first check in the affected component using *checkin* function. If check-in is successful, followed it with an XML request to *check* designated build procedures, which (if successful) should then be followed by an XML component *build* request. (See *“Check Designated Build Procedures - CMPONENT APL_DPRC CHECK”* and *“Component Service Build - CMPONENT SERVICE BUILD”* later in this chapter.)

CMPONENT APL_DPRC CHECK — Requests

The Serena XML example below shows how you might code a request to check designated component build procedures. Note that you can check build procedures for only one component per request. Data structure details for the `<request>` tag appear in *Exhibit 4-3*.

Example XML — CMPONENT APL_DPRC CHECK Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="APL_DPRC">
  <message name="CHECK">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <component>ACPSRS00</component>
    <componentType>SRS</componentType>
    <applName>ACTP</applName>
    <buildProc>CMNCOB2</buildProc>
    <language>COBOL2</language>
    <useDb2PreCompileOption>N</useDb2PreCompileOption>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 4-3. Check Component Designated Build Procedures <request>

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Required	0 - 1	String (4), fixed	ZMF application name. Same as first 4 bytes of `<package>` tag. ***NOTE:*** Trailing blanks required.
<buildProc>	Required	0 - 1	String (8), variable	ZMF name for designated build procedure.

Exhibit 4-3. Check Component Designated Build Procedures <request> *(Continued)*

Subtag	Use	Instances	Data Type & Length	Values
<compileOptions>	Optional	0 - 1	String (34), variable	Custom compile parameters for named component.
<component>	Required	1	String (256), variable	ZMF name of component to be checked. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally pre-fixed by path from installation root.
<componentType>	Required	1	String (3), fixed	Library type of component named in `<component>`. Must be valid ZMF library type of "like-source." Typical values: •COB •CPY •SRC
<language>	Required	0 - 1	String (8), variable	Source language of component.
<linkOptions>	Optional	0 - 1	String (34), variable	Custom link-edit parameters for named component.
<useDb2PreCompileOption>	Required	0 - 1	String (1)	**Y** = Yes, use DB2 precompile **N** = No, don't precompile DB2
<userOption01> . . . <userOption20>	Optional	0 - 1 each	String (1)	Administrator-defined build options assigned to component. Each tag corresponds to User Option 01 to 20 on the ISPF user options panel for component build.
<userOption0101> . . . <userOption0105>	Optional	0 - 1 each	String (1), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0101 to 0105 on the ISPF user options panel for component build.
<userOption0201> . . . <userOption0203>	Optional	0 - 1 each	String (2), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0201 to 0203 on the ISPF user options panel for component build.
<userOption0301> . . . <userOption0303>	Optional	0 - 1 each	String (3), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0301 to 0303 on the ISPF user options panel for component build.

Exhibit 4-3. Check Component Designated Build Procedures <request> *(Continued)*

Subtag	Use	Instances	Data Type & Length	Values
<userOption0401> . . . <userOption0403>	Optional	0 - 1 each	String (4), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0401 to 0403 on the ISPF user options panel for component build.
<userOption0801> . . . <userOption0805>	Optional	0 - 1 each	String (8), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0801 to 0805 on the ISPF user options panel for component build.
<userOption1001> . . . <userOption1002>	Optional	0 - 1 each	String (10), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 1001 to 1002 on the ISPF user options panel for component build.
<userOption1601> . . . <userOption1602>	Optional	0 - 1 each	String (16), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 1601 to 1603 on the ISPF user options panel for component build.
<userOption3401> . . . <userOption3402>	Optional	0 - 1 each	String (34), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 3401 to 3402 on the ISPF user options panel for component build.
<userOption4401> . . . <userOption4402>	Optional	0 - 1 each	String (44), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 4401 to 4402 on the ISPF user options panel for component build.
<userOption6401> . . . <userOption6405>	Optional	0 - 1 each	String (64), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 6401 to 6405 on the ISPF user options panel for component build.
<userOption7201> . . . <userOption7205>	Optional	0 - 1 each	String (72), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 7201 to 7205 on the ISPF user options panel for component build.

Tip

Tags: <userOption01> to <userOption20>, <userOptionsPart1>, <userOptionsPart2>, <userOption0101> to <userOption7205>. See topic "Staging User Options" in the ChangeMan ZMF Customization Guide.

Check Component Designated Build Procedures — Replies

No `<result>` data structure is returned in the reply message to a *check component designated procedures* request. However, the standard `<response>` data structure is returned to indicate the success or failure of the check. In general, the following rules of thumb apply to return codes for the designated build procedure check:

- ***00*** – Request successful and designated build procedures found.
- ***04*** – Informational message; designated build procedures not found.
- ***08 or higher*** – Failure to execute request.

Find Designated Build Procedure - CMPONENT APL_DPRC FIND

The Serena XML service/scope/message tags and attributes for messages to *find a component's designated build procedure* are:

```
<service name="CMPONENT">
<scope name="APL_DPRC">
<message name="FIND">
```

These tags appear in both requests and replies.

CMPONENT APL_DPRC FIND — Requests

The Serena XML request to find the designated component build procedure retrieves detailed information from the component history file for a specific component. Data structure details for CMPONENT APL_DPRC FIND request are shown in *Exhibit 4-4*.

Exhibit 4-4. CMPONENT APL_DPRC FIND <request>

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Required	0 - 1	String (4), fixed	ZMF application name. Same as first 4 bytes of `<package>` tag. ***NOTE:*** Trailing blanks required.
<component>	Required	0 - 1	String (256), variable	ZMF component for which to find the designated build procedure. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally pre-fixed by path from installation root.
<componentType>	Required \	1	String (3), fixed	ZMF library type of component(s) for which to list designated build procedures. Must be "like-source." Typical values: •COB •CPY •SRC

CMPONENT APL_DPRC FIND — Replies

The reply message returns zero or one result tag, detailing the procedure if it exists.The reply data structure is the same as that for CMPONENT APL_DPRC LIST and is described in *Exhibit 4-6*.

List Designated Build Procedures - CMPONENT APL_DPRC LIST

The Serena XML service/scope/message tags and attributes for messages to *list designated component build procedures* at the application level are:

```
<service name="CMPONENT">
<scope name="APL_DPRC">
<message name="LIST">
```

These tags appear in both requests and replies.

CMPONENT APL_DPRC LIST — Requests

The Serena XML request to list designated component build procedures retrieves detailed information from the component history file about all such procedures for any of the following:

- One explicitly named component.
- All components of the same type.
- Any components with names that match a wildcard pattern.

The Serena XML example below shows how you might code a request to list designated component build procedures for all components of type "SRS". The request to *list* designated component build procedures also supports wildcards and patterns. Data structure details for the `<request>` tag appear in *Exhibit 4-5*.

Example XML — CMPONENT APL_DPRC LIST Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="APL_DPRC">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <applName>ACTP</applName>
    <component>*</component>
    <componentType>SRS</componentType>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 4-5. CMPONENT APL_DPRC LIST <request>

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Required	0 - 1	String (4), fixed	ZMF application name. Same as first 4 bytes of `<package>` tag. ***NOTE:*** Trailing blanks required.
<component>	Required	0 - 1	String (256), variable	ZMF component for which to list designated build procedures. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally pre-fixed by path from installation root. ***NOTE:*** May be masked using standard wildcard characters.

Exhibit 4-5. CMPONENT APL_DPRC LIST <request> *(Continued)*

Subtag	Use	Instances	Data Type & Length	Values
<componentType>	Optional	1	String (3), fixed	ZMF library type of component(s) for which to list designated build procedures. Must be "like-source." Typical values: •COB •CPY •SRC
<exactMatch>	Optional	0 - 1	String (1)	**Y** = Yes- exact match no filtering **N** = No - use filtering

CMPONENT APL_DPRC LIST — Replies

The reply message listing designated component build procedures returns zero to many `<result>` data elements. Each `<result>` tag contains information about one component, taken from the component history file. This information includes component name and type, build procedure name, component source language, compile and link options, and the like.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Example XML — CMPONENT APL_DPRC LIST Reply

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="APL_DPRC">
  <message name="LIST">
   <result>
    <component>ACPSRS00</component>
    <componentType>SRS</componentType>
    <applName>ACTP</applName>
    <buildProc>CMNCOB2</buildProc>
    <language>COBOL2</language>
    <useDb2PreCompileOption>N</useDb2PreCompileOption>
    <forceAssignedBuildProc>2</forceAssignedBuildProc>
   </result>
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Data structure details for the `<result>` tag appear in *Exhibit 4-6*.

Exhibit 4-6. CMPONENT APL_DPRC LIST <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), fixed	ZMF application name. Same as first 4 bytes of `<package>` tag. ***NOTE:*** Trailing blanks required.
<buildProc>	Optional	0 - 1	String (8), variable	ZMF name for designated build procedure.
<compileOptions>	Optional	0 - 1	String (34), variable	Custom compile parameters for named component.
<component>	Optional	0 - 1	String (256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally pre-fixed by path from installation root.
<componentType>	Required	1	String (3), variable	ZMF library type of component.
<forceAssignedBuildProc>	Optional	0 - 1	String (1)	Force level for enforcement of designated build procedure with this component. Values: **1** = Force before freeze only **2** = Always force
<language>	Optional	0 - 1	String (8), variable	Source language of component.
<linkOptions>	Optional	0 - 1	String (34), variable	Custom link-edit parameters for named component.
<useDb2PreCompileOption>	Required	0 - 1	String (1)	**Y** = Yes, use DB2 precompile **N** = No, don't precompile DB2
<userOption01> . . . <userOption20>	Optional	0 - 1 each	String (1)	Administrator-defined build options assigned to component. Each tag corresponds to User Option 01 to 20 on the ISPF user options panel for component build.
<userOption0101> . . . <userOption0105>	Optional	0 - 1 each	String (1), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0101 to 0105 on the ISPF user options panel for component build.

Exhibit 4-6. CMPONENT APL_DPRC LIST <result> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<userOption0201> . . . <userOption0203>	Optional	0 - 1 each	String (2), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0201 to 0203 on the ISPF user options panel for component build.
<userOption0301> . . . <userOption0303>	Optional	0 - 1 each	String (3), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0301 to 0303 on the ISPF user options panel for component build.
<userOption0401> . . . <userOption0403>	Optional	0 - 1 each	String (4), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0401 to 0403 on the ISPF user options panel for component build.
<userOption0801> . . . <userOption0805>	Optional	0 - 1 each	String (8), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0801 to 0805 on the ISPF user options panel for component build.
<userOption1001> . . . <userOption1002>	Optional	0 - 1 each	String (10), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 1001 to 1002 on the ISPF user options panel for component build.
<userOption1601> . . . <userOption1602>	Optional	0 - 1 each	String (16), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 1601 to 1603 on the ISPF user options panel for component build.
<userOption3401> . . . <userOption3402>	Optional	0 - 1 each	String (34), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 3401 to 3402 on the ISPF user options panel for component build.
<userOption4401> . . . <userOption4402>	Optional	0 - 1 each	String (44), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 4401 to 4402 on the ISPF user options panel for component build.

Exhibit 4-6. CMPONENT APL_DPRC LIST <result> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<userOption6401> . . . <userOption6405>	Optional	0 - 1 each	String (64), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 6401 to 6405 on the ISPF user options panel for component build.
<userOption7201> . . . <userOption7205>	Optional	0 - 1 each	String (72), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 7201 to 7205 on the ISPF user options panel for component build.

Tip

Tags: <userOption01> to <userOption20>, <userOptionsPart1>, <userOptionsPart2>, <userOption0101> to <userOption7205>. See topic "Staging User Options" in the ChangeMan ZMF Customization Guide.

List Global Designated Build Procedures - CMPONENT GBL_DPRC LIST

The Serena XML service/scope/message tags and attributes for messages to *list designated component build procedures* at the global level are:

```
<service name="CMPONENT">
<scope name="GBL_DPRC">
<message name="LIST">
```

These tags appear in both requests and replies.

The CMPONENT GBL_DPRC LIST service is identical to the CMPONENT APL_DPRC LIST service except that the `<appl_Name>` tag is omitted. Refer to *"List Designated Build Procedures - CMPONENT APL_DPRC LIST" on page 224*.

Component Service Build - CMPONENT SERVICE BUILD

The Serena XML service/scope/message tags and attributes for component *build* messages are:

```
<service name="CMPONENT">
<scope name="SERVICE">
<message name="BUILD">
```

These tags appear in both requests and replies.

The Serena XML component *build* function performs a subset of the "stage" functions of the ISPF user interface. It compiles (or assembles) and link-edits a package component; logs source-to-load and other relationships between modules within a package; and builds any JCL install jobs. Together these comprise the third step of three in the staging flow. The component *build* function does not copy files from a development location into a staged change package. Neither does it check for the existence of designated build procedures associated with a checked-in component.

Tip

To perform the full "stage to development" process using Serena XML, start with the *check in* function. If successful, follow check-in with an XML request to *check* designated build procedures. If this, too, is successful, submit a Serena XML request to *build* the component. (See *"Component Service Checkin - CMPONENT SERVICE CHECKIN"* and *"Check Designated Build Procedures - CMPONENT APL_DPRC CHECK"* earlier in this chapter.)

CMPONENT SERVICE BUILD Request

The following example shows how you might code a build request with Serena XML. Data structure details for the component service build `<request>` tag follow the example in *Exhibit 4-7*.

Example XML — CMPONENT SERVICE BUILD Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="SERVICE">
  <message name="BUILD">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>TES5000001</package>
    <componentType>SRS</componentType>
    <buildProc>CMNCOB2</buildProc>
    <language>COBOL2</language>
   <jobCard01>//XMLX029B JOB (RWM,T),'DUMP',CLASS=A,MSGCLASS=A</jobCard01>
    <jobCard02>//* JOBCARD2</jobCard02>
    <jobCard03>//* JOBCARD3</jobCard03>
    <jobCard04>//* JOBCARD4</jobCard04>
    <listCount>001</listCount>
    <component>ACPSRS00</component>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 4-7. CMPONENT SERVICE BUILD <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), fixed	ZMF application name. Same as first 4 bytes of `<package>` tag. ***NOTE:*** Trailing blanks required.
<buildProc>	Optional	0 - 1	String (8), variable	8-byte ZMF name for designated build procedure.
<compileOptions>	Optional	0 - 1	String (34), variable	Compile parameters not set elsewhere (e.g. in component history) or by default. ***NOTE:*** The <useHistory> tag must be set to N to use this tag.
<component>	Required	1 - ∞	String (256), variable	ZMF name of component to build. Repeatable to accommodate multiple components. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root. NOTE: Number of instances must equal the value in the `<listCount>` tag.
<componentType>	Required	1	String (3), fixed	Library type of all component(s) to be built. Must be valid ZMF library type of "like-source." Typical values: •COB •CPY •SRC
<db2PreCompileLinkLib>	Optional	0 - 1	String (44), variable	Data set name of DB2 library to be used in build process. ***NOTE:*** The <useHistory> tag must be set to N to use this tag.
<db2PreCompileVersion>	Optional	0 - 1	String (64), variable	DB2 DBRM version to use when building components. ***NOTE:*** The <useHistory> tag must be set to N to use this tag.
<db2SubSystemId>	Optional	0 - 1	String (4), variable	4-byte physical subsystem ID of DB2 instance to use in build. ***NOTE:*** The <useHistory> tag must be set to N to use this tag.
<incrementJobname>	Optional	0 - 1	String (1)	**Y** = Yes, increment the job name **N** = No, don't increment the job name
<inputDataset>	Optional	0 - 1	String (44), variable	Data set name of staging library where like-source component(s) reside(s).

Exhibit 4-7. CMPONENT SERVICE BUILD <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<jobCard01> . . . <jobCard04>	Required	0 - 1, each	String (72), variable	JCL statements needed to set job parameters, allocate data sets, & define library concatenations. If used, all four are required. Tags not needed for JCL should be coded as comment (//*).
<language>	Optional	0 - 1	String (8), variable	Source language of component(s) to be compiled. Max 8 bytes.
<linkOptions>	Optional	0 - 1	String (34), variable	Link edit parameters not set elsewhere (e.g. in component history) or by default. ***NOTE:*** The <useHistory> tag must be set to N to use this tag.
<listCount>	Required	1	Integer	Number of components to be checked out. Must equal the number of `<component>` tags that follow.
<package>	Required	1	String (10), variable	ZMF fixed-format package name where component(s) reside(s).
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required.
<sourceLocation>	Optional	0 - 1	String (1)	This tag is used when staging “like type” OTH components from remote clients. Specify the value “7” to indicate that the build source dataset is a temporary dataset and will be deleted when the build process is completed. ***CAUTION!*** Do not use this tag unless you are sure you want to delete the input dataset.
<suppressNotify>	Optional	0 - 1	String (1)	**Y** = Yes, suppress notify messages **N** = No, don’t suppress messages
<useDb2PreCompileOption>	Optional	0 - 1	String (1)	**Y** = Yes, use DB2 precompile **N** = No, don’t precompile DB2 ***NOTE:*** The <useHistory> tag must be set to N to use this tag.
<useHistory>	Optional	0 - 1	String (1)	**Y** = Yes, use comp hist for compile params (default) **N** = No, don’t use comp history ***NOTE:*** This tag must be set to N to use the <useDb2PreCompileOption>, <compileOptions>, <linkOptions>, <db2SubSystemId>, <db2PreCompileLinkLib>, and <db2PreCompileVersion> tags.

Exhibit 4-7. CMPONENT SERVICE BUILD <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<userOption01> . . . <userOption20>	Optional	0 - 1, each	String (1)	Administrator-defined 1-byte user option variables. ***NOTE:*** See your ZMF application administrator for information.
<userOption0101> . . . <userOption0105>	Optional	0 - 1 each	String (1), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0101 to 0105 on the ISPF user options panel for component build.
<userOption0201> . . . <userOption0203>	Optional	0 - 1 each	String (2), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0201 to 0203 on the ISPF user options panel for component build.
<userOption0301> . . . <userOption0303>	Optional	0 - 1 each	String (3), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0301 to 0303 on the ISPF user options panel for component build.
<userOption0401> . . . <userOption0403>	Optional	0 - 1 each	String (4), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0401 to 0403 on the ISPF user options panel for component build.
<userOption0801> . . . <userOption0805>	Optional	0 - 1 each	String (8), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0801 to 0805 on the ISPF user options panel for component build.
<userOption1001> . . . <userOption1002>	Optional	0 - 1 each	String (10), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 1001 to 1002 on the ISPF user options panel for component build.
<userOption1601> . . . <userOption1602>	Optional	0 - 1 each	String (16), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 1601 to 1603 on the ISPF user options panel for component build.
<userOption3401> . . . <userOption3402>	Optional	0 - 1 each	String (34), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 3401 to 3402 on the ISPF user options panel for component build.

Exhibit 4-7. CMPONENT SERVICE BUILD <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<userOption4401> . . . <userOption4402>	Optional	0 - 1 each	String (44), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 4401 to 4402 on the ISPF user options panel for component build.
<userOption6401> . . . <userOption6405>	Optional	0 - 1 each	String (64), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 6401 to 6405 on the ISPF user options panel for component build.
<userOption7201> . . . <userOption7205>	Optional	0 - 1 each	String (72), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 7201 to 7205 on the ISPF user options panel for component build.
<userOptionsPart1>	Optional	0 - 1	String (10), variable	Administrator-defined component user variables.
<userOptionsPart2>	Optional	0 - 1	String (10), variable	Administrator-defined component user variables.
<userPanel>	Optional	0 - 1	String (8), variable	User panel ID.
<userVariable01> . . . <userVariable05>	Optional	0 - 1, each	String (8), variable	Administrator-defined 8-byte user variables, if any, for use with customized skeletons during build. ***NOTE:*** See your ZMF application administrator for information.
<userVariable06> . . . <userVariable10>	Optional	0 - 1, each	String (72), variable	Administrator-defined 72-byte user variables, if any, for use with customized skeletons during build. ***NOTE:*** See your ZMF application administrator for information.

Tip

Tags: <userOption01> to <userOption20>, <userOptionsPart1>, <userOptionsPart2>, <userOption0101> to <userOption7205>. See topic “Staging User Options” in the ChangeMan ZMF Customization Guide.

CMPONENT SERVICE BUILD Replies

No `<result>` data structure is returned in the component *build* reply message. However, the standard `<response>` data structure is returned to indicate the success or failure of the build request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Tip

Tags: <userVariable01> to <userVariable10>: See topic "Custom V01-V10 Variables" in the ChangeMan ZMF Customization Guide.

Recompile a Component - CMPONENT SERVICE RECOMP

The Serena XML service/scope/message tags and attributes for component *recompile* messages are:

```
<service name="CMPONENT">
<scope name="SERVICE">
<message name="RECOMP">
```

These tags appear in both requests and replies.

CMPONENT SERVICE RECOMP Requests

Unlike the component check-in and check-out functions described above, the component *recompile* function works on only one component per request message. It performs only one task: compilation (or assembly) of the named component.

Note

The component *recompile* function does not link-edit the named component. That task can be performed on a standalone basis using the Serena XML *relink* function, described later in this chapter.

The Serena XML example below shows how you might code a request to recompile a component from baseline. For illustration, the example requests a DB2 precompile. Note that DB2-related tags apply only to customers who install the ChangeMan ZMF DB2 Option. Administrator-defined user options and variables with hypothetical values also appear in the example. Check with your ChangeMan ZMF administrator for further information about custom user variables for components. They may not apply to your installation.

Data structure details for the recompile `<request>` tag appear in *Exhibit 4-8*.

Example XML — CMPONENT SERVICE RECOMP Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="SERVICE">
  <message name="RECOMP">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>TES5000001</package>
    <componentType>SRS</componentType>
     <jobCard01>//XMLX034  JOB (RWM,T),'DUMP',CLASS=A,MSGCLASS=A,REGION=0M</
jobCard01>
    <listCount>0001</listCount>
    <component>ACPSRS1B</component>
    <language>COBOL2</language>
    <buildProc>CMNCOB2</buildproc>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 4-8. CMPONENT SERVICE RECOMP <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks.
<buildProc>	Optional	0 - 1	String (8), variable	8-byte ZMF name for designated build procedure.
<compileOptions>	Optional	0 - 1	String (34), variable	Compile parameters not set elsewhere (e.g. in component history) or by default.
<component>	Required	1	String (256), variable	ZMF name of component to recompile. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentType>	Required	1	String (3), fixed	ZMF “like-source” library type of component to be recompiled. Typical values: •COB •CPY •SRC
<db2PreCompileLinkLib>	Optional	0 - 1	String (44), variable	Data set name of DB2 library to be used in build process.
<db2PreCompileVersion>	Optional	0 - 1	String (64), variable	DB2 DBRM version to use when recompiling components.
<db2SubSystemId>	Optional	0 - 1	String (4), variable	Physical subsystem ID of DB2 instance to use in build.
<jobCard01> <jobCard02> <jobCard03> <jobCard04>	Required Optional Optional Optional	1 0 - 1 0 - 1 0 - 1	String (72) String (72) String (72) String (72)	JCL statements needed to set job parameters, allocate data sets, & define library concatenations. <jobCard01> is required. Tags not needed for JCL may be omitted.

Exhibit 4-8. CMPONENT SERVICE RECOMP <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<language>	Optional	0 - 1	String (8), variable	Source language of component(s) to be compiled. If omitted, ZMF retrieves from component history.
<libLevel>	Optional	1	String (2), variable	Numeric library level of source code to recompile. Values: **0** = Baseline library **1 to 99** = Promotion library ***NOTE:*** `<promotionSiteName>` tag also required if value > 0.
<linkOptions>	Optional	0 - 1	String (34), variable	Link edit parameters not set elsewhere (e.g. in component history) or by default. ***NOTE:*** The <useHistory> tag must be set to N to use this tag.
<listCount>	Optional	0 - 1	Integer	Number of components to be recompiled.
<lockComponent>	Optional	0 - 1	String (1)	**Y** = Yes, lock after recompile **N** = No, don't lock component
<overlayPriorVersion>	Optional	0 - 1	String (1)	**Y** = Yes, overlay load module in staging library **N** = No, don't overlay
<package>	Required	1	String (10), variable	ZMF fixed-format package name where component resides.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required.
<promotionSiteName>	Optional	0 - 1	String (8), variable	ZMF promotion library site name. ***NOTE:*** If `<libLevel>` = 1 to 99, this tag is required.
<release>	Optional	0 - 1	String (8), variable	Name of ZMF release (ERO Option only).
<releaseArea>	Optional	0 - 1	String (8), variable	Name of the ZMF release area (ERO Option only).
<suppressNotify>	Optional	0 - 1	String (1)	**Y** = Yes, suppress notify messages **N** = No, don't suppress messages
<useDb2PreCompileOption>	Optional	0 - 1	String (1)	**Y** = Yes, use DB2 precompile **N** = No, don't precompile DB2

Exhibit 4-8. CMPONENT SERVICE RECOMP <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<useHistory>	Optional	0 - 1	String (1)	**Y** = Yes, use comp hist for compile params (default) **N** = No, don't use comp history ***NOTE:*** This tag must be set to N to use the <useDb2PreCompileOption>, <compileOptions>, <linkOptions>, <db2SubSystemId>, <db2PreCompileLinkLib>, and <db2PreCompileVersion> tags.
<userOption01> . . . <userOption20>	Optional	0 - 1, each	String (1)	Administrator-defined 1-byte user option variables. ***NOTE:*** See your ZMF application administrator for information.
<userOption0101> . . . <userOption0105>	Optional	0 - 1 each	String (1), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0101 to 0105 on the ISPF user options panel for component build.
<userOption0201> . . . <userOption0203>	Optional	0 - 1 each	String (2), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0201 to 0203 on the ISPF user options panel for component build.
<userOption0301> . . . <userOption0303>	Optional	0 - 1 each	String (3), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0301 to 0303 on the ISPF user options panel for component build.
<userOption0401> . . . <userOption0403>	Optional	0 - 1 each	String (4), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0401 to 0403 on the ISPF user options panel for component build.
<userOption0801> . . . <userOption0805>	Optional	0 - 1 each	String (8), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0801 to 0805 on the ISPF user options panel for component build.
<userOption1001> . . . <userOption1002>	Optional	0 - 1 each	String (10), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 1001 to 1002 on the ISPF user options panel for component build.

Exhibit 4-8. CMPONENT SERVICE RECOMP <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<userOption1601> . . . <userOption1602>	Optional	0 - 1 each	String (16), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 1601 to 1603 on the ISPF user options panel for component build.
<userOption3401> . . . <userOption3402>	Optional	0 - 1 each	String (34), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 3401 to 3402 on the ISPF user options panel for component build.
<userOption4401> . . . <userOption4402>	Optional	0 - 1 each	String (44), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 4401 to 4402 on the ISPF user options panel for component build.
<userOption6401> . . . <userOption6405>	Optional	0 - 1 each	String (64), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 6401 to 6405 on the ISPF user options panel for component build.
<userOption7201> . . . <userOption7205>	Optional	0 - 1 each	String (72), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 7201 to 7205 on the ISPF user options panel for component build.
<userOptionsPart1>	Optional	0 - 1	String (10), variable	Administrator-defined component user variables.
<userOptionsPart2>	Optional	0 - 1	String (10), variable	Administrator-defined component user variables.
<userPanel>	Optional	0 - 1	String (8), variable	User panel ID.
<userVariable01> . . . <userVariable05>	Optional	0 - 1	String (8), variable	Administrator-defined 8-byte user variables for recompile.
<userVariable06> . . . <userVariable10>	Optional	0 - 1	String (72), variable	Administrator-defined 72-byte user variables, if any, for use with recompile.

Tip

Tags: <userOption01> to <userOption20>, <userOptionsPart1>, <userOptionsPart2>, <userOption0101> to <userOption7205>. See topic “Staging User Options” in the ChangeMan ZMF Customization Guide.

CMPONENT SERVICE RECOMP Replies

No `<result>` data structure is returned in the component *recompile* reply message. However, the standard `<response>` data structure is returned to indicate the success or failure of the recompile request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Tip

Tags: <userVariable01> to <userVariable10>: See topic “Custom V01-V10 Variables” in the ChangeMan ZMF Customization Guide.

Relink a Component - CMPONENT SERVICE RELINK

The Serena XML service/scope/message tags and attributes for component *relink* messages are:

```
<service name=”CMPONENT”>
<scope name=”SERVICE”>
<message name=”RELINK”>
```

These tags appear in both requests and replies.

CMPONENT SERVICE RELINK Requests

The component *relink* function, like the component *recompile* function, works on only one component per request message. It performs only one task: link-editing the named component. Prior compilation is assumed.

Note

The component *relink* function does not compile the named component or change the source code in any way. Compilation (or assembly) can be performed on a standalone basis using the Serena XML *recompile* function, described earlier in this chapter.

The example below shows how you might code a request to relink a component in Serena XML. Data structure details for the recompile `<request>` tag appear in *Exhibit 4-9*.

Example XML — CMPONENT SERVICE RELINK Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="SERVICE">
  <message name="RELINK">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000014</package>
    <component>ACPSRS00</component>
    <componentType>LOS</componentType>
    <targetLoadLibType>LOD</targetLoadLibType>
    <buildProc>CMNCOB2</buildProc>
    <language>COBOL2</language>
    <linkOptions>NCAL</linkOptions>
    <useDb2PreCompileOption>N</useDb2PreCompileOption>
    <jobCard01>//XMLX035  JOB (AMW,000),'DEFINE UCAT',MSGCLASS=Y,</
jobCard01>
    <jobCard02>//            TIME=(,10),NOTIFY=USER24</jobCard02>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 4-9. CMPONENT SERVICE RELINK <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks.
<buildProc>	Optional	0 - 1	String (8), variable	8-byte ZMF name for designated build procedure.
<compileOptions>	Optional	0 - 1	String (34), variable	Compile parameters not set elsewhere (e.g. in component history) or by default.
<component>	Required	1	String (256), variable	ZMF name of component to relink. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.

Exhibit 4-9. CMPONENT SERVICE RELINK <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<componentType>	Required	1	String (3), fixed	ZMF "like-source" library type of component to be recompile. Typical values: •COB •CPY •SRC
<db2PreCompileLinkLib>	Optional	0 - 1	String (44), variable	Data set name of DB2 library to be used in build process.
<db2PreCompileVersion>	Optional	0 - 1	String (64), variable	DB2 DBRM version to use when recompiling components.
<db2SubSystemId>	Optional	0 - 1	String (4), variable	Physical subsystem ID of DB2 instance to use in build.
<jobCard01> <jobCard02> <jobCard03> <jobCard04>	Required Optional Optional Optional	1 0 - 1 0 - 1 0 - 1	String (72) String (72) String (72) String (72)	JCL statements needed to set job parameters, allocate data sets, & define library concatenations. <jobCard01> is required. Tags not needed for JCL may be omitted.
<language>	Optional	0 - 1	String (8), variable	Source language of component(s) to be compiled. If omitted, ZMF retrieves from component history.
<linkOptions>	Optional	0 - 1	String (34), variable	Link edit parameters not set elsewhere (e.g. in component history) or by default. ***NOTE:*** The <useHistory> tag must be set to N to use this tag.
<package>	Required	1	String (10), variable	ZMF fixed-format package name where component resides.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required.
<targetLoadLibType>	Required	0 - 1	String (3), variable	Target load library type.
<useDb2PreCompileOption>	Optional	0 - 1	String (1)	**Y** = Yes, use DB2 precompile **N** = No, don't precompile DB2
<useHistory>	Optional	0 - 1	String (1)	**Y** = Yes, use comp hist for compile params (default) **N** = No, don't use comp history ***NOTE:*** This tag must be set to N to use the <useDb2PreCompileOption>, <compileOptions>, <linkOptions>, <db2SubSystemId>, <db2PreCompileLinkLib>, and <db2PreCompileVersion> tags.

Exhibit 4-9. CMPONENT SERVICE RELINK <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<userOption01> . . . <userOption20>	Optional	0 - 1, each	String (1)	Administrator-defined 1-byte user option variables. ***NOTE:*** See your ZMF application administrator for information.
<userOption0101> . . . <userOption0105>	Optional	0 - 1 each	String (1), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0101 to 0105 on the ISPF user options panel for component build.
<userOption0201> . . . <userOption0203>	Optional	0 - 1 each	String (2), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0201 to 0203 on the ISPF user options panel for component build.
<userOption0301> . . . <userOption0303>	Optional	0 - 1 each	String (3), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0301 to 0303 on the ISPF user options panel for component build.
<userOption0401> . . . <userOption0403>	Optional	0 - 1 each	String (4), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0401 to 0403 on the ISPF user options panel for component build.
<userOption0801> . . . <userOption0805>	Optional	0 - 1 each	String (8), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0801 to 0805 on the ISPF user options panel for component build.
<userOption1001> . . . <userOption1002>	Optional	0 - 1 each	String (10), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 1001 to 1002 on the ISPF user options panel for component build.
<userOption1601> . . . <userOption1602>	Optional	0 - 1 each	String (16), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 1601 to 1603 on the ISPF user options panel for component build.
<userOption3401> . . . <userOption3402>	Optional	0 - 1 each	String (34), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 3401 to 3402 on the ISPF user options panel for component build.

Exhibit 4-9. CMPONENT SERVICE RELINK <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<userOption4401> . . . <userOption4402>	Optional	0 - 1 each	String (44), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 4401 to 4402 on the ISPF user options panel for component build.
<userOption6401> . . . <userOption6405>	Optional	0 - 1 each	String (64), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 6401 to 6405 on the ISPF user options panel for component build.
<userOption7201> . . . <userOption7205>	Optional	0 - 1 each	String (72), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 7201 to 7205 on the ISPF user options panel for component build.
<userOptionsPart1>	Optional	0 - 1	String (10), variable	Administrator-defined component user variables.
<userOptionsPart2>	Optional	0 - 1	String (10), variable	Administrator-defined component user variables.
<userPanel>	Optional	0 - 1	String (8), variable	User panel ID.
<userVariable01> . . . <userVariable05>	Optional	0 - 1	String (8), variable	Administrator-defined 8-byte user variables for recompile.
<userVariable06> . . . <userVariable10>	Optional	0 - 1	String (72), variable	Administrator-defined 72-byte user variables, if any, for use with recompile.

Tip

Tags: <userVariable01> to <userVariable10>: See topic “Custom V01-V10 Variables” in the ChangeMan ZMF Customization Guide.

Tip

Tags: <userOption01> to <userOption20>, <userOptionsPart1>, <userOptionsPart2>, <userOption0101> to <userOption7205>. See topic “Staging User Options” in the ChangeMan ZMF Customization Guide.

CMPONENT SERVICE RELINK Replies

No `<result>` data structure is returned in the component *relink* reply message. However, the standard `<response>` data structure is returned to indicate the success or failure of the relink request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

A successful request will generate a job with output similar to the following:

```
-STEPNAME PROCSTEP    RC   EXCP   CONN    TCB    SRB
-SSIDN                00     91     46    .00    .00
-ALLOC                00     14      6    .00    .00
-ALLOCIN              00     38     19    .00    .00
-LINK                 00    125     61    .00    .00
-BT90LOD              00    181     79    .00    .00
-COPYLOD              00    235    555    .00    .00
-SUCCESS              00    649    320    .00    .00
-CHKCOND              00     14      5    .00    .00
-FAILURE           FLUSH      0      0    .00    .00
-PRINT                00    262    126    .00    .00
-COMPLST              00    137    110    .00    .00
-ILODLST              00    561    301    .00    .00
-XMLX035  ENDED.  NAME-DEFINE UCAT          TOTAL TCB
$HASP395 XMLX035  ENDED
.
.
***************************************************************************
* DDNAME: SSIDN.SYSPRINT
****************************************************************************

ChangeMan(R)      CMNSSIDN - 6.1.0  THURSDAY FEBRUARY 19, 2009  09:17:08
 PARM=''
SYSIN: LCT=ACPSRS00
SYSIN: SSI=5C6D1B0A
SYSIN: PKG=ACTP000014
SYSIN: RLK=Y
SYSIN: UIL=Y
SYSIN: OPT=CALL
Options compiled from PARM/SYSIN follow:
 NAME        - Allow "NAME" directive.
 CALL        - Allow "INCLUDE" directives.
 RELINK      - Re-Linkage-Edit by INCLUDEing again.
END OF DATA ON "OBJ" DETECTED
STAGING "LCT" OPENED
STAGING "LCT" MEMBER NOT FOUND
FABRICATING LCT CARDS FROM SCRATCH

<...+....1....+....2....+....3....+....4....+....5....+....6....+....7
LCT:          INCLUDE INCLIB(ACPSRS00)
LCT:           SETSSI 5C6D1B0A
LCT:            NAME  ACPSRS00(R)
```

```
***************************************************************************
* DDNAME: LINK.SYSPRINT
* DDNAME: BT90LOD.BAT90LST
***************************************************************************

z/OS V1 R8 BINDER     09:17:09 THURSDAY FEBRUARY 19, 2009
BATCH EMULATOR  JOB(XMLX035 ) STEP(LINK    ) PGM= IEWL
IEW2278I B352 INVOCATION PARAMETERS - LIST,XREF,MAP,RENT,NCAL

IEW2322I 1220  1          INCLUDE INCLIB(ACPSRS00)
IEW2322I 1220  2           SETSSI 5C6D1B0A
IEW2322I 1220  3            NAME  ACPSRS00(R)
IEW2650I 5102 MODULE ENTRY NOT PROVIDED.  ENTRY DEFAULTS TO SECTION
ACPSRS00.

***************************************************************************
* DDNAME: BT90LOD.SYSPRINT
***************************************************************************

ChangeMan(R)      CMNBAT90 - 6.1.0  THURSDAY FEBRUARY 19, 2009  09:17:10
SYSIN: PKG=ACTP000014
SYSIN: SLT=LOS
SYSIN: SLT=LOS
SYSIN: SNM=ACPSRS00
SYSIN: SID=USER24
SYSIN: SSI=5C6D1B0A
SYSIN: PRC=CMNCOB2
SYSIN: RLK=YES
SYSIN: SUP=NO
SYSIN: LLT=LOD
SYSIN: SLB=ACTPLOSCMNTP.SERT8.DEV.ACTP.#000014.LOS
SYSIN: SLB=ACTPLODCMNTP.SERT8.DEV.ACTP.#000014.LOD
SYSIN: SLB=ACTPLOSCMNTP.SERT8.BASE.ACTP.LOS
SYSIN: SLB=ACTPLODCMNTP.SERT8.BASE.ACTP.LOD
SYSIN: ILB=ACTPLOSCMNTP.SERT8.DEV.ACTP.#000014.LOS
SYSIN: ILB=ACTPLOSCMNTP.SERT8.BASE.ACTP.LOS
CMN5400I - Time of day at end of job: 09:17:11 - Condition Code on exit: 00
```

Browse a Component - CMPONENT SERVICE BROWSE

The Serena XML service/scope/message tags and attributes for component service browse messages are:

```
<service name="CMPONENT">
<scope name="SERVICE">
<message name="BROWSE">
```

These tags appear in both requests and replies.

The component browse function of ChangeMan ZMF is actually a component download function when accessed via XML. Replies come back as XML documents suitable for offline browsing in a text editor or by XML-aware browser software.

CMPONENT SERVICE BROWSE Request

The example on the next page shows how you might code a component service browse request in Serena XML. Data structure details for the browse `<request>` tag appear in *Exhibit 4-10*.

Example XML — CMPONENT SERVICE BROWSE Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="SERVICE">
  <message name="BROWSE">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000007</package>
    <component>ACPCPY00</component>
    <componentType>CPY</componentType>
    <trim>Y</trim>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 4-10. CMPONENT SERVICE BROWSE <request> Data Structure

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks.
<browseFromOption>	Optional	1	String (1)	Code for component library to browse. Values: **1** = Browse from package **2** = Browse from baseline **3** = Browse from package if found, otherwise from baseline ***NOTE:*** Options 1 and 3 require the <package> tag to be specified.

Exhibit 4-10. CMPONENT SERVICE BROWSE <request> Data Structure *(Continued)*

Subtag	Use	Instances	Data Type & Length	Values
<component>	Required	1	String (256), variable	ZMF name of component to browse. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally pre-fixed by path from installation root.
<componentType>	Required	1	String (3)	ZMF library type of component to browse.
<hashToken>	Optional	0 - 1	String (16)	ZMF-generated "fingerprint" of component to browse. If component has changed since the hash token was generated, Serena XML returns a warning.
<package>	Required	1	String (10)	ZMF fixed-format package name where component resides. ***NOTE:*** This tag is required if <browseFromOption> = 1 or 3.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required.
<trim>	Optional	0 - 1	String (1)	Define N if you do not want the trailing blanks to be trimmed before </line> tag ***NOTE:*** Default is Y.

CMPONENT SERVICE BROWSE Reply

The component service browse reply returns one `<result>` tag for the component requested. Component contents are line-oriented in format; that is, each line of component text is returned in its own `<line>` tag.

Tip

If you prefer to download a file without `<line>` tags, consider using the *data download* service instead of *component browse*.

A standard `<response>` data structure follows the `<result>` tag, if any, to indicate the success or failure of the browse request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last data element returned in a Serena XML reply message, the `<response>` tag serves as an end-of-list marker.

An example reply to a component service browse request follows.

Example XML — Component Service Browse Reply

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="SERVICE">
  <message name="BROWSE">
   <result>
    <line>      *    ACPCPY00.CAP</line>
    <line>      *    ACPCPY00.CAP</line>
    <line>       01  ACPCPY00                         PIC  X(01).</line>
   </result>
   <response>
    <statusMessage>CMN8700I - Download service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Compare Components - CMPONENT SERVICE COMPARE

The Serena XML service/scope/message tags and attributes for component *compare* messages are:

```
<service name="CMPONENT">
<scope name="SERVICE">
<message name="COMPARE">
```

These tags appear in both requests and replies.

CMPONENT SERVICE COMPARE Requests

You can use Serena XML to compare a component in the staging library against a like-named component in any baseline or promotion library. The value in the `<baseLibLevel>` tag specifies the baseline level (0 to -99) or promotion level (1 to 999) for the comparison.

The example below shows how you might code a request to compare a component in a staged package with a baselined version of that component. Data structure details for the compare `<request>` tag appear in *Exhibit 4-11*.

Example XML — CMPONENT SERVICE COMPARE Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="SERVICE">
  <message name="COMPARE">
   <header>
```

```
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>TES5000001</package>
    <component>ACPCPY00</component>
    <componentType>CPY</componentType>
    <baseLibLevel>001</baseLibLevel>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 4-11. CMPONENT SERVICE COMPARE <request> Data Structure

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks.
<baseLibLevel>	Required	1	String (3), variable	Baseline or promotion library level of component compared against component in staging. *Baseline range:* 0 to -99 *Promotion range:* 1 to 999
<component>	Required	1	String (256), variable	ZMF component to be compared. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally pre-fixed by path from installation root.
<componentType>	Required	1	String (3), fixed	ZMF library type of component to be compared. Typical values: • COB • CPY • JCL • LOD • SRC
<package>	Required	1	String (10), variable	ZMF fixed-format package name where component resides.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required.

CMPONENT SERVICE COMPARE Replies

The component package browse reply returns one `<result>` tag, which contains the component comparison report. Each line of the report is bracketed by a `<line>` tag.

Example XML — CMPONENT SERVICE COMPARE Reply

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="SERVICE">
  <message name="COMPARE">
   <result>
     <line>1    S E R C M P A R  (MVS - 560 - 20080929) 2 TEXTONLY    FRIDAY
FEBRUARY 13, 2009 (2009/044) 08:36:47 PAGE 1</line>
     <line>
SYSUT1=CMNTP.A013D.#C3BDFBA.#313AB10.WORKSRD(ACPCPY00),SYSUT2=CMNTP.SERT8.D
EV.TES5.#000001.CPY(ACPCPY00)</line>
   <line>0         *   ACPCPY00.CAP
O N E  1</line>
   <line>         *   ACPCPY00.CAP
O N E  2</line>

<line>0++++++++|+++.++++1++++.++++2++++.++++3++++.++++4++++.++++5++++.++++6
++++.++++7++++.++++8++++++++++++++++++++</line>
   <line> I         *   ADDED 2/13/2009 8:35 AM
DIF  T W O  3  +</line>
     <line>
++++++++|+++.++++1++++.++++2++++.++++3++++.++++4++++.++++5++++.++++6++++.++
++7++++.++++8++++++++++++++++++++</line>
    <line></line>
   <line>         01 ACPCPY00               PIC X(01).
O N E  3</line>
  <line>0SER71I - END OF TEXT ON FILE SYSUT1</line>
  <line>0SER72I - END OF TEXT ON FILE SYSUT2</line>
     <line>-SER75I - RECORDS PROCESSED: SYSUT1(3)/
SYSUT2(4),DIFFERENCES(0,0,1)</line>
     <line>                         EXPLANATION - 0 RECORDS DIFFER THAT
SYNCHRONIZED TOGETHER</line>
    <line>                                    0 RECORDS WERE CONSIDERED
INSERTED ON SYSUT1</line>
     <line>                                     1 RECORD WAS CONSIDERED
INSERTED ON SYSUT2</line>
     <line>1   S E R C M P A R  (MVS - 560 - 20080929) 2    TEXTONLY  FRIDAY
FEBRUARY 13, 2009(2009/044) 08:36:47 PAGE 2</line>
     <line>
SYSUT1=CMNTP.A013D.#C3BDFBA.#313AB10.WORKSRD,SYSUT2=CMNTP.SERT8.DEV.TES5.#0
00001.CPY</line>
   <line>0SER71I - END OF DIRECTORY ON FILE SYSUT1</line>
   <line>0SER72I - END OF DIRECTORY ON FILE SYSUT2</line>
     <line>0SER78I - MEMBERS PROCESSED: SYSUT1(1)/
SYSUT2(1),DIFFERENCES(1),REJECTED BY FILTERS: SYSUT1(0)/SYSUT2(0)</line>
```

```
    <line>0SER80I - TIME OF DAY AT END OF JOB: 08:36:47 - CONDITION CODE ON
EXIT: 4</line>
  </result>
  <response>
 <statusMessage>CMN8700I - Download service completed</statusMessage>
  <statusReturnCode>00</statusReturnCode>
   <statusReasonCode>8700</statusReasonCode>
  </response>
 </message>
 </scope>
</service>
.
```

Rename a Component - CMPONENT SERVICE RENAME

The Serena XML service/scope/message tags and attributes for component *rename* messages are:

```
<service name=”CMPONENT”>
<scope name=”SERVICE”>
<message name=”RENAME”>
```

These tags appear in both requests and replies.

CMPONENT SERVICE RENAME Requests

The example below shows how you might code a component *rename* request in Serena XML. Data structure details for the rename `<request>` tag appear in *Exhibit 4-12*.

Example XML — CMPONENT SERVICE RENAME Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="SERVICE">
  <message name="RENAME">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000009</package>
    <componentType>CPY</componentType>
    <oldComponent>ACPCPY1B</oldComponent>
    <newComponent>ACPCPY2B</newComponent>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 4-12. CMPONENT SERVICE RENAME <request> Data Structure

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks.
<componentType>	Required	1	String (3), fixed	ZMF library type of renamed component. Typical values: • COB • CPY • JCL • LOD • SRC
<newComponent>	Required	1	String (256), variable	New ZMF name of renamed component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally pre-fixed by path from installation root.
<oldComponent>	Required	1	String (256), variable	Old ZMF name of component to rename. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally pre-fixed by path from installation root.
<package>	Required	1	String (10), variable	ZMF name of package where component resides.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required.

CMPONENT SERVICE RENAME Replies

No `<result>` data structure is returned in the component *rename* reply message. However, the standard `<response>` data structure is returned to indicate the success or failure of the rename request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Scratch a Component - CMPONENT SERVICE SCRATCH

The Serena XML service/scope/message tags and attributes for component *scratch* messages are:

```
<service name="CMPONENT">
<scope name="SERVICE">
<message name="SCRATCH">
```

These tags appear in both requests and replies.

Component Scratch Requests

The example on the next page shows how you might code a component *scratch* request in Serena XML. Data structure details for the scratch `<request>` tag appear in *Exhibit 4-13*, following the example.

Example XML — CMPONENT SERVICE SCRATCH Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="SERVICE">
  <message name="SCRATCH">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000009</package>
    <componentType>CPY</componentType>
    <oldComponent>ACPCPY1C</oldComponent>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 4-13. CMPONENT SERVICE SCRATCH <request> Data Structure

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks.

Exhibit 4-13. CMPONENT SERVICE SCRATCH <request> Data Structure *(Continued)*

Subtag	Use	Instances	Data Type & Length	Values
<componentType>	Required	1	String (3), fixed	ZMF library type of component to scratch. Typical values: • COB • CPY • JCL • LOD • SRC
<oldComponent>	Required	1	String (256), variable	Name of component to scratch. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<package>	Required	1	String (10), variable	ZMF name of package where component resides.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required.

CMPONENT SERVICE SCRATCH Replies

No `<result>` data structure is returned in the component *scratch* reply message. However, the standard `<response>` data structure is returned to indicate the success or failure of the scratch request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Lock or Unlock a Component - CMPONENT SERVICE LOCK/UNLOCK

The component lock and unlock functions in Serena XML share identical request and reply data structures nested within their `<service>`, `<scope>`, and `<message>` tags. Only the `name` attribute of the `<message>` tag differs.

The service/scope/message tags and attributes for component *lock* messages are:

```
<service name="CMPONENT">
<scope name="SERVICE">
<message name="LOCK">
```

The service/scope/message tags and attributes for component *unlock* messages are:

```
<service name="CMPONENT">
<scope name="SERVICE">
<message name="UNLOCK">
```

These tags appear in both requests and replies.

CMPONENT SERVICE LOCK/UNLOCK Requests

The example below shows how you might code a component *lock* request in Serena XML. An *unlock* request would be coded similarly, substituting the attribute `name="unlock"` in the `<message>` tag. Data structure details for the `<request>` tag used in both component lock and unlock messages appear in *Exhibit 4-14*.

Example XML — CMPONENT SERVICE LOCK Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="SERVICE">
  <message name="LOCK">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>TES5000001</package>
    <component>ACPCPY00</component>
    <componentType>CPY</componentType>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 4-14. CMPONENT SERVICE LOCK <request> Data Structure

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks.
<component>	Required	1	String (256), variable	ZMF name of component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentType>	Required	1	String (3), fixed	ZMF library type of component. Typical values: • COB • CPY • JCL • LOD • SRC

Exhibit 4-14. CMPONENT SERVICE LOCK <request> Data Structure *(Continued)*

Subtag	Use	Instances	Data Type & Length	Values
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required.

CMPONENT SERVICE LOCK/UNLOCK Replies

No `<result>` data structure is returned in component *lock* or *unlock* reply messages. However, the standard `<response>` data structure is returned to indicate the success or failure of the lock request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

List Load Module Subroutines - CMPONENT LOD_SUBR LIST

The Serena XML service/scope/message tags and attributes for messages to *list* information about load module subroutines are:

```
<service name="CMPONENT">
<scope name="LOD_SUBR">
<message name="LIST">
```

These tags appear in both requests and replies.

CMPONENT LOD_SUBR LIST Requests

The CMPONENT LOD_SUBR LIST request retrieves information about statically linked subroutines within a load module.

The example below shows how you might code a component subroutine *list* request in Serena XML. Data structure details for the `<request>` tag appear in *Exhibit 4-15*.

Example XML — CMPONENT LOD_SUBR LIST Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="LOD_SUBR">
  <message name="LIST">
   <header>
    <subsys>4</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000007</package>
    <compositeComponent>ACPSRC1A</compositeComponent>
    <compositeComponentType>LOD</compositeComponentType>
    <sourceComponent>ACPSRC1A</sourceComponent>
    <sourceComponentType>SRC</sourceComponentType>
  </request>
  </message>
 </scope>
</service>
```

Exhibit 4-15. CMPONENT LOD_SUBR LIST <request> Data Structure

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks.
<compositeComponent>	Required	1	String (256), variable	ZMF name of load module. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root. ***NOTE:*** Component name may be masked using standard wildcards.
<compositeComponentType>	Required	1	String (3), fixed	ZMF library type of composite component.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required.
<sourceComponent>	Required	1	String (256), variable	ZMF name of source component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root. ***NOTE:*** Component name may be masked using standard wildcards.
<sourceComponentType>	Required	1	String (3), fixed	ZMF library type of source component.

CMPONENT LOD_SUBR LIST Replies

The reply message listing information about a load module and its statically linked subroutines returns zero to many `<result>` data elements. Each `<result>` tag contains information about one subroutine within the composite component. This information includes the subroutine name and type, the SETSSI value, and so on.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Example XML — CMPONENT LOD_SUBR LIST Reply

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="LOD_SUBR">
  <message name="LIST">
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <compositeComponent>ACPSRC1A</compositeComponent>
    <compositeComponentType>LOD</compositeComponentType>
    <sourceComponent>ACPSRC1A</sourceComponent>
    <sourceComponentType>SRC</sourceComponentType>
    <compositeSetssi>61118F95</compositeSetssi>
    <compositeFromIDR>Y</compositeFromIDR>
    <subroutineComponent>ACPSRC1A</subroutineComponent>
    <subroutineComponentAppl>ACTP</subroutineComponentAppl>
    <subroutineComponentType>LOD</subroutineComponentType>
    <subroutinePackage>ACTP000007</subroutinePackage>
    <subroutineApplName>ACTP</subroutineApplName>
    <subroutinePackageId>000007</subroutinePackageId>
    <subroutineSetssi>61118F95</subroutineSetssi>
    <subroutineFromIDR>Y</subroutineFromIDR>
   </result>
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <compositeComponent>ACPSRC1A</compositeComponent>
    <compositeComponentType>LOD</compositeComponentType>
    <sourceComponent>ACPSRC1A</sourceComponent>
    <sourceComponentType>SRC</sourceComponentType>
    <compositeSetssi>61118F95</compositeSetssi>
    <compositeFromIDR>Y</compositeFromIDR>
    <subroutineComponent>ACPSRS1B</subroutineComponent>
    <subroutineComponentAppl>ACTP</subroutineComponentAppl>
    <subroutineComponentType>LOS</subroutineComponentType>
    <subroutinePackage>ACTP000007</subroutinePackage>
    <subroutineApplName>ACTP</subroutineApplName>
    <subroutinePackageId>000007</subroutinePackageId>
    <subroutineSetssi>60AFD725</subroutineSetssi>
    <subroutineFromIDR>Y</subroutineFromIDR>
   </result>
   .
   .
```

```
    .
    <response>
     <statusMessage>CMN8700I - LIST service completed</statusMessage>
     <statusReturnCode>00</statusReturnCode>
     <statusReasonCode>8700</statusReasonCode>
    </response>
   </message>
  </scope>
</service>
```

Data structure details for the `<result>` tag appear in *Exhibit 4-16*.

Exhibit 4-16. CMPONENT LOD_SUBR LIST <result> Data Structure

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<compositeComponent>	Required	1	String (256), variable	ZMF name of load module. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<compositeComponentType>	Required	1	String (3), fixed	ZMF library type of composite component.
<compositeFromIDR>	Optional	0 -1	String (1)	Indicates if composite information is from IDR. **Y** = Yes, information is from IDR. **N** = No, information is not from IDR.
<compositeHashToken>	Optional	0 -1	String (16)	Composite hash token.
<compositeSetssi>	Required	1	String (8)	Composite SETSSI value.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.
<sourceComponent>	Required	1	String (256), variable	ZMF name of source component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.

Exhibit 4-16. CMPONENT LOD_SUBR LIST <result> Data Structure *(Continued)*

Subtag	Use	Instances	Data Type & Length	Values
<sourceComponentType>	Required	1	String (3), fixed	ZMF library type of source component.
<subroutineApplName>	Optional	0 - 1	String (4), variable	ZMF application name of subroutine.
<subroutineComponent>	Required	1	String (256), variable	ZMF name of subroutine component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<subroutineComponentAppl>	Required	1	String (4), variable	ZMF application name of subroutine component.
<subroutineComponentType>	Required	1	String (3), fixed	ZMF library type of subroutine component.
<subroutineFromIDR>	Optional	0 -1	String (1)	Indicates if subroutine information is from IDR. **Y** = Yes, information is from IDR. **N** = No, information is not from IDR.
<subroutineHashToken>	Optional	0 -1	String (16)	Subroutine hash token.
<subroutinePackage>	Required	1	String (10), fixed	Fixed-format ZMF package name of subroutine.
<subroutinePackageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number of subroutine.
<subroutineRelease>	Optional	1	String (8), variable	Subroutine release name.
<subroutineReleaseArea>	Optional	1	String (8), variable	Subroutine release area name.
<subroutineSetssi>	Required	1	String (8)	Subroutine SETSSI value.

List Copybook Names in Source - CMPONENT SRC_INCL LIST

The Serena XML service/scope/message tags and attributes for messages to *list* information about copybooks included within a source component are:

```
<service name=”CMPONENT”>
<scope name=”SRC_INCL”>
<message name=”LIST”>
```

These tags appear in both requests and replies.

CMPONENT SRC_INCL LIST Requests

The example below shows how you might code a CMPONENT SRC_INCL LIST request in Serena XML. Data structure details for the `<request>` tag appear in *Exhibit 4-17*.

Example XML — CMPONENT SRC_INCL LIST Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="SRC_INCL">
  <message name="LIST">
   <header>
    <subsys>4</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000007</package>
    <sourceComponent>ACPSRC1A</sourceComponent>
    <sourceComponentType>SRC</sourceComponentType>
  </request>
  </message>
 </scope>
</service>
```

Exhibit 4-17. CMPONENT SRC_INCL LIST <request> Data Structure

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required.
<sourceComponent>	Required	1	String (256), variable	ZMF name of source component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root. ***NOTE:*** Component name may be masked using standard wildcards.
<sourceComponentType>	Required	1	String (3), fixed	ZMF library type of source component.

CMPONENT SRC_INCL LIST Replies

The reply message listing information about a source component and its included copybooks returns zero to many `<result>` data elements. Each `<result>` tag contains information about one copybook within the source component. This information includes the copybook name and type, the version, and so on.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Example XML — CMPONENT SRC_INCL LIST Reply

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="SRC_INCL">
  <message name="LIST">
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <sourceComponent>ACPSRC1A</sourceComponent>
    <setssi>61118FA5</setssi>
    <srcHashToken>6E1E9BDD0000035A</srcHashToken>
    <includedVersion>01</includedVersion>
    <includedModLevel>01</includedModLevel>
    <includedHashToken>6721849B000000A3</includedHashToken>
    <includedApplName>ACTP</includedApplName>
    <includedComponentType>CPY</includedComponentType>
    <includedComponent>ACPCPY00</includedComponent>
   </result>
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <sourceComponent>ACPSRC1A</sourceComponent>
    <sourceComponentType>SRC</sourceComponentType>
    <setssi>61118FA5</setssi>
    <srcHashToken>6E1E9BDD0000035A</srcHashToken>
    <includedVersion>01</includedVersion>
    <includedModLevel>01</includedModLevel>
    <includedHashToken>BDC5C909000000BE</includedHashToken>
    <includedApplName>ACTP</includedApplName>
    <includedComponentType>CPY</includedComponentType>
    <includedComponent>ACPCPY1A</includedComponent>
   </result>
   .
   .
   .
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Data structure details for the `<result>` tag appear in *Exhibit 4-18*.

Exhibit 4-18. CMPONENT INCL_SRC LIST <result> Data Structure

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<includedApplName>	Required		String (4), variable	ZMF application name of included component.
<includedComponent>	Required	1	String (256), variable	ZMF name of included component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<includedComponentType>	Required	1	String (3), fixed	ZMF library type of included component.
<includedHashToken>	Required	1	String (16)	Hash token of included component.
<includedModLevel>	Required	1	Integer (2), fixed	Modification level of included component.
<includedVersion>	Required	1	Integer (2), fixed	Version of included component.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.
<release>	Optional	0 - 1	String (8), variable	Release name.
<releaseArea>	Optional	0 - 1	String (8), variable	Release area name.
<setssi>	Required	1	String (8)	SETSSI value of source component.

Exhibit 4-18. CMPONENT INCL_SRC LIST <result> Data Structure *(Continued)*

Subtag	Use	Instances	Data Type & Length	Values
<sourceComponent>	Required	1	String (256), variable	ZMF name of source component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<sourceComponentType>	Required	1	String (3), fixed	ZMF library type of source component.
<srcHashToken>	Required	1	String (16)	Hash token of source component.

Component Staging Version Management

Staging version functions for general use include the following:

- *List Component Staging Versions - CMPONENT SSV_VER LIST*
- *Retrieve Component Staging Version - CMPONENT SSV_VER RETRIEVE*

List Component Staging Versions - CMPONENT SSV_VER LIST

The Serena XML service/scope/message tags and attributes for messages to *list* all staging versions of a component are:

```
<service name="CMPONENT">
<scope name="SSV_VER">
<message name="LIST">
```

These tags appear in both requests and replies.

CMPONENT SSV_VER LIST — Requests

The example below shows how you might code a request to *list* the staging versions of a component. Staging versions may be listed for only one component per request. Data structure details for the list staging versions `<request>` tag appear in *Exhibit 4-19*, following the example.

Example XML — CMPONENT SSV_VER LIST Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="SSV_VER">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>CISQ000030</package>
    <componentType>SRC</componentType>
    <component>CI2Q101</component>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 4-19. CMPONENT SSV_VER LIST <request> Data Structure

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks.
<component>	Required	1	String (256), variable	ZMF name of component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally pre-fixed by path from installation root.
<componentType>	Required	1	String (3), fixed	ZMF library type of component. Must be editable source code (RECFM not 'U'). Values: • COB • CPY • JCL • SRC
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required.

CMPONENT SSV_VER LIST — Replies

The Serena XML reply message to a component staging versions *list* request contains zero to many `<result>` tags. Each `<result>` contains information about one previously staged version of the requested component. The `<result>` tag repeats for each staging version of the component.

The standard `<response>` data structure follows the final `<result>` tag and indicates the success or failure of the list request. Successful requests have a return code of 00.

Unsuccessful requests have a return code of 04 or higher. As the final data element in the reply, the `<response>` tag also serves as an end-of-list marker.

An example reply to a list staging versions request follows. Data structure details for the list `<result>` tag appear in *Exhibit 4-20*.

Example XML — CMPONENT SSV_VER LIST Reply

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="SSV_VER">
  <message name="LIST">
   <result>
    <versionLocation>2</versionLocation>
    <lastUpdater>USER24</lastUpdater>
    <dateLastModified>20081126</dateLastModified>
    <timeLastModified>100700</timeLastModified>
    <changeDesc>BIG SETQUERY PACKAGE</changeDesc>
    <ispfUser>USER24</ispfUser>
    <ispfDateLastModified>20081126</ispfDateLastModified>
    <ispfTimeLastModified>095100</ispfTimeLastModified>
    <ispfUpdateSize>00094</ispfUpdateSize>
    <ispfVersion>001</ispfVersion>
    <ispfModLevel>001</ispfModLevel>
    <ispfInitialDate>20080118</ispfInitialDate>
    <ispfInitialSize>00090</ispfInitialSize>
    <ispfModSize>00000</ispfModSize>
   </result>
   <result>
    <versionLocation>4</versionLocation>
    <lastUpdater>USER24</lastUpdater>
    <dateLastModified>20081126</dateLastModified>
    <timeLastModified>095100</timeLastModified>
    <changeDesc>Baseline version</changeDesc>
    <ispfUser>USER24</ispfUser>
    <ispfDateLastModified>20081126</ispfDateLastModified>
    <ispfTimeLastModified>095100</ispfTimeLastModified>
    <ispfUpdateSize>00094</ispfUpdateSize>
    <ispfVersion>001</ispfVersion>
    <ispfModLevel>001</ispfModLevel>
    <ispfInitialDate>20080118</ispfInitialDate>
    <ispfInitialSize>00090</ispfInitialSize>
    <ispfModSize>00000</ispfModSize>
   </result>
   <response>
    <statusMessage>CMN8700I - SSV service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 4-20. CMPONENT SSV_VER LIST <result> Data Structure

Subtag	Use	Instances	Data Type & Length	Values
<backupDate>	Optional	0 - 1	Date, yyyymmdd	Date component version was last backed up.
<backupTime>	Optional	0 - 1	Time, hhmmss	Time component version was last backed up, 24-hour format.
<backupUser>	Optional	0 - 1	String (8), variable	TSO user ID of most recent user to back up component version.
<changeDesc>	Optional	0 - 1	String (35), variable	Text description of changes made in this version of component.
<dateLastModified>	Optional	0 - 1	Date, yyyymmdd	Date component version was last changed.
<fileFormat>	Optional	0 - 1	Integer (1)	For regular HFS files, the Unix numeric code for data organization and record delimiter. Allowed values: `0` = Not specified `1` = Binary data `2` = New line (NL) `3` = Carriage return (CR) `4` = Line feed (LF) `5` = CR & LF `6` = LF & CR `7` = CR & NL NOTE: Always supplied for HFS data files. Irrelevant and omitted for HFS directories, links or aliases, pipes, or sockets, or for non-HFS components.
<ispfDateLastModified>	Optional	0 - 1	Date, yyyymmdd	Date component version was last changed or staged by ISPF user.
<ispfInitialDate>	Optional	0 - 1	Date, yyyymmdd	Date component version was created.
<ispfInitialSize>	Optional	0 - 1	Integer (5), variable	Lines of code in version before change.
<ispfModLevel>	Optional	0 - 1	Integer (3), variable	ISPF 2-byte modification level for component when last staged.
<ispfModSize>	Optional	0 - 1	Integer (5), variable	Lines of code changed in version.
<ispfTimeLastModified>	Optional	0 - 1	Time, hhmmss	Time component version was last changed or staged by ISPF user, 24-hour format.

Exhibit 4-20. CMPONENT SSV_VER LIST <result> Data Structure *(Continued)*

Subtag	Use	Instances	Data Type & Length	Values
<ispfUpdateSize>	Optional	0 - 1	Integer (5), variable	Lines of code in version after change.
<ispfUser>	Optional	0 - 1	String (8), variable	TSO user ID of last ISPF user to change or stage this component when it was the STG version.
<ispfVersion>	Optional	0 - 1	Integer (3), variable	ISPF 2-byte version number for component when last staged.
<lastUpdater>	Optional	0 - 1	String (8), variable	TSO user ID of last component updater for this version.
<permissions>	Optional	0 - 1	Integer (3), fixed	Unix access permissions for HFS file or directory, coded as 3-digit integer, where: 1st digit = owner permissions 2nd digit = group permissions 3rd digit = permissions for all others Each digit takes one of the following values: 7 - Read, write/rename/delete, execute 6 - Read, write/rename/delete 5 - Read, execute 4 - Read only 3 - Write/rename/delete, execute 2 - Write/rename/delete only 1 - Execute only 0 - No access permitted NOTE: Always listed for HFS components. Irrelevant and omitted for non-HFS components.
<promotionLevel>	Optional	0 - 1	Integer (4), variable	ZMF promotion level to which this component was last promoted when it was the STG version.
<promotionName>	Optional	0 - 1	String (8), variable	ZMF name of promotion level corresponding to level number in `<promotionLevel>`.
<promotionSiteName>	Optional	0 - 1	String (8), variable	ZMF name of site to which this component was last promoted when it was the STG version.
<timeLastModified>	Optional	0 - 1	Time, hhmmss	Time component version was last changed, 24-hour format.

Exhibit 4-20. CMPONENT SSV_VER LIST <result> Data Structure *(Continued)*

Subtag	Use	Instances	Data Type & Length	Values
<versionLocation>	Optional	0 - 1	Integer (1)	Code for library where this version of the component resides. Values: **2** = Staging library **4** = Baseline library **6** = Staging version backup lib
<versionStamp>	Optional	0 - 1	String (8), fixed	Hexadecimal time-and-date stamp that uniquely identifies this version of this component.

Retrieve Component Staging Version - CMPONENT SSV_VER RETRIEVE

The Serena XML service/scope/message tags and attributes for messages to *retrieve* a specific staging version of a component are:

```
<service name="CMPONENT">
<scope name="SSV_VER">
<message name="RETRIEVE">
```

These tags appear in both requests and replies.

The staging version *retrieve* function is a subset of the staging version *recover* function seen in the ISPF user interface. The retrieve function finds the desired version of a component in the staging version VSAM master, then copies it to a temporary data set for use. The name of the temporary data set is auto-generated and returned in the Serena XML reply message.

The ISPF *recover* function goes one step further: it copies the auto-generated data set to any data set chosen by the user. The XML *retrieve* function does not perform this second step.

CMPONENT SSV_VER RETRIEVE — Requests

Data structure details for the staging version retrieve `<request>` tag appear in *Exhibit 4-21*.

Exhibit 4-21. CMPONENT SSV_VER RETRIEVE <request>

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks.
<component>	Required	1	String (256), variable	ZMF name of component to retrieve from staging version. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally pre-fixed by path from installation root.
<componentType>	Required	1	String (3), fixed	ZMF library type of component. Must be editable source code (RECFM not 'U'). Values: • COB • CPY • JCL • SRC
<package>	Required	1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required.
<versionStamp>	Required	1	String (8), fixed	Hexadecimal date/time stamp of particular component version to retrieve.

CMPONENT SSV_VER RETRIEVE — Replies

The Serena XML reply message for the staging version *retrieve* function returns one `<result>` for the requested component version, if found. The standard `<response>` data structure follows the `<result>` tag and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Data structure details for the retrieval `<result>` tag appear in *Exhibit 4-22*.

Exhibit 4-22. CMPONENT SSV_VER RETRIEVE <result>

Subtag	Use	Instances	Data Type & Length	Values
<backupDate>	Optional	0 - 1	Date, yyyymmdd	Date component version was last backed up.
<backupTime>	Optional	0 - 1	Time, hhmmss	Time component version was last backed up, 24-hour format.
<backupUser>	Optional	0 - 1	String (8), variable	TSO user ID of most recent user to back up component version.
<changeDesc>	Optional	0 - 1	String (35), variable	Text description of changes made in this version of component.
<dateLastModified>	Optional	0 - 1	Date, yyyymmdd	Date component version was last changed.
<fileFormat>	Optional	0 - 1	Integer (1)	For regular HFS files, the Unix numeric code for data organization and record delimiter. Allowed values: `0` = Not specified `1` = Binary data `2` = New line (NL) `3` = Carriage return (CR) `4` = Line feed (LF) `5` = CR & LF `6` = LF & CR `7` = CR & NL NOTE: Always supplied for HFS data files. Irrelevant and omitted for HFS directories, links or aliases, pipes, or sockets, or for non-HFS components.
<ispfDateLastModified>	Optional	0 - 1	Date, yyyymmdd	Date component version was last changed or staged by ISPF user.
<ispfInitialDate>	Optional	0 - 1	Date, yyyymmdd	Date component version was created.
<ispfInitialSize>	Optional	0 - 1	Integer (5), variable	Lines of code in version before change.
<ispfModLevel>	Optional	0 - 1	Integer (3), variable	ISPF 2-byte modification level for component when last staged.
<ispfModSize>	Optional	0 - 1	Integer (5), variable	Lines of code changed in version.
<ispfTimeLastModified>	Optional	0 - 1	Time, hhmmss	Time component version was last changed or staged by ISPF user, 24-hour format.
<ispfUpdateSize>	Optional	0 - 1	Integer (5), variable	Lines of code in version after change.

Exhibit 4-22. CMPONENT SSV_VER RETRIEVE <result> *(Continued)*

Subtag	Use	Instances	Data Type & Length	Values
<ispfUser>	Optional	0 - 1	String (8), variable	TSO user ID of last ISPF user to change or stage this component when it was the STG version.
<ispfVersion>	Optional	0 - 1	Integer (3), variable	ISPF 2-byte version number for component when last staged.
<lastUpdater>	Optional	0 - 1	String (8), variable	TSO user ID of last component updater for this version.
<permissions>	Optional	0 - 1	Integer (3), fixed	Unix access permissions for HFS file or directory, coded as 3-digit integer, where: 1st digit = owner permissions 2nd digit = group permissions 3rd digit = permissions for all others Each digit takes one of the following values: 7 - Read, write/rename/delete, execute 6 - Read, write/rename/delete 5 - Read, execute 4 - Read only 3 - Write/rename/delete, execute 2 - Write/rename/delete only 1 - Execute only 0 - No access permitted NOTE: Always listed for HFS components. Irrelevant and omitted for non-HFS components.
<promotionLevel>	Optional	0 - 1	Integer (4), variable	ZMF promotion level to which this component was last promoted when it was the STG version.
<promotionName>	Optional	0 - 1	String (8), variable	ZMF name of promotion level corresponding to level number in `<promotionLevel>`.
<promotionSiteName>	Optional	0 - 1	String (8), variable	ZMF name of site to which this component was last promoted when it was the STG version.
<tempLib>	Optional	0 - 1	String (256), variable	Temporary data set name where version is returned.
<timeLastModified>	Optional	0 - 1	Time, hhmmss	Time component version was last changed, 24-hour format.

Exhibit 4-22. CMPONENT SSV_VER RETRIEVE <result> *(Continued)*

Subtag	Use	Instances	Data Type & Length	Values
<versionLocation>	Optional	0 - 1	Integer (1)	Code for library where this version of the component resides. Values: **2** = Staging library **4** = Baseline library **6** = Staging version backup lib
<versionStamp>	Optional	0 - 1	String (8), fixed	Hexadecimal time-and-date stamp that uniquely identifies this version of this component.

Component Information Management Tasks

Component information management tasks retrieve or manage control information and descriptive metadata about one or more components. Such tasks include:

- *List Component Change Description - CMPONENT CHG_DESC LIST*
- *Find Component Description - CMPONENT APL_CDSC FIND*
- *List Component Description - CMPONENT APL_CDSC LIST*
- *List Global Component Description - CMPONENT GBL_CDSC LIST*
- *List Component Promotion History - CMPONENT PRM_HIST LIST*
- *Component History List - CMPONENT HISTORY LIST*
- *List Short Component History - CMPONENT HISTORY LISTSHRT*
- *List Current Component History - CMPONENT HISTORY LISTCURR*
- *List Concurrent Comp. History - CMPONENT HISTORY LISTCONC*
- *List Baselined Component History - CMPONENT HISTORY LISTBASE*
- *List Comp. User Worklist Records - CMPONENT PKG_WRKL LIST*

Note that some component information management functions, such as listing all the components in a package, are described with ***package-level*** component tasks. See *"Package-Level Component Change Management" in Chapter 3, Package Management*, for those functions.

List Component Change Description - CMPONENT CHG_DESC LIST

The Serena XML service/scope/message tags and attributes for messages that *list change descriptions* for one or more components are:

```
<service name="CMPONENT">
<scope name="CHG_DESC">
<message name="LIST">
```

These tags appear in both requests and replies.

CMPONENT CHG_DESC LIST — Request

The following example below shows how you might code a Serena XML request to list the change description for a single component in a package. Data structure details for the list `<request>` tag appear in *Exhibit 4-23*.

Example XML — CMPONENT CHG_DESC LIST Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="CHG_DESC">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000007</package>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 4-23. CMPONENT CHG_DESC LIST <request> Data Structure

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<component>	Optional	1	String (256), variable	ZMF name of component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally pre-fixed by path from installation root. ***NOTE:*** May be masked using the following wildcard characters: **?** = Any one alphanumeric character. ***** = Any number of alphanumeric characters. Used alone, it matches all values.
<componentType>	Optional	1	String (3), fixed	ZMF library type of component. ***NOTE:*** May be masked using the asterisk (*) wildcard character.

Exhibit 4-23. CMPONENT CHG_DESC LIST <request> Data Structure *(Continued)*

Subtag	Use	Instances	Data Type & Length	Values
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.

CMPONENT CHG_DESC LIST — Reply

The Serena XML reply message for the component *change description list* function returns zero to many `<result>` tags for the requested component(s). Each result contains a component change description for the component name and library type in the reply.

The standard `<response>` data structure follows the final `<result>` tag and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the final data structure in the reply, the `<response>` tag also serves as an end-of-list marker.

An example reply to a component change description list request follows. Data structure details for the list `<result>` tag appear in *Exhibit 4-24*.

Example XML — CMPONENT CHG_DESC LIST Reply

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="CHG_DESC">
  <message name="LIST">
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <component>ACPCPY00</component>
    <componentType>CPY</componentType>
    <changeDesc>SER5904E</changeDesc>
   </result>
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <component>ACPCPY1A</component>
    <componentType>CPY</componentType>
    <changeDesc>SER5904E</changeDesc>
   </result>
.
.
.
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 4-24. CMPONENT CHG_DESC LIST<result> Data Structure

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<changeDesc>	Optional	0 - 1	String (35), variable	Text description of change to component.
<component>	Optional	1	String (256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally pre-fixed by path from installation root.
<componentType>	Optional	1	String (3), fixed	ZMF library type of component.
<package>	Optional	1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.

Find Component Description - CMPONENT APL_CDSC FIND

The Serena XML service/scope/message tags and attributes for messages that *find* the application-level description for a specific component are:

```
<service name="CMPONENT">
<scope name="APL_CDSC">
<message name="FIND">
```

These tags appear in both requests and replies.

CMPONENT APL_CDSC FIND — Request

This function first tries to find the application-level description for a specific component. If no description is found at the application level, the global records are searched.

Data structure details for the component description find `<request>` tag appear in *Exhibit 4-25*.

Exhibit 4-25. CMPONENT APL_CDSC FIND <request> Data Structure

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Required	1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<component>	Required	1	String (256), variable	ZMF name of component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally pre-fixed by path from installation root.
<componentType>	Required	1	String (3), fixed	ZMF library type of component.

CMPONENT APL_CDSC FIND— Reply

The Serena XML reply message for the component description *find* function returns one `<result>` tag for the requested component. It contains the component description and library type in the reply.

The standard `<response>` data structure follows the `<result>` tag and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Data structure details for the list `<result>` tag appear in *Exhibit 4-26*.

Exhibit 4-26. CMPONENT APL_CDSC FIND <result> Data Structure

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<component>	Optional	0 - 1	String (256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally pre-fixed by path from installation root.
<componentDesc>	Optional	0 - 48	String (72), variable	Text description of component.
<componentType>	Optional	0 - 1	String (3), fixed	ZMF library type of component.

List Component Description - CMPONENT APL_CDSC LIST

The Serena XML service/scope/message tags and attributes for messages that *list* the application-level description for one or more components are:

```
<service name="CMPONENT">
<scope name="APL_CDSC">
<message name="LIST">
```

These tags appear in both requests and replies.

CMPONENT APL_CDSC LIST — Request

Data structure details for the component description list `<request>` tag appear in *Exhibit 4-27*.

Exhibit 4-27. CMPONENT APL_CDSC LIST <request> Data Structure

Subtag	Use	Instances	Data Type & Length	Values
<applName>	Required	1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<component>	Required	1	String (256), variable	ZMF name of component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally pre-fixed by path from installation root. ***NOTE:*** May be masked using the following wildcard characters: **?** = Any one alphanumeric character. ***** = Any number of alphanumeric characters. Used alone, it matches all values.
<componentType>	Required	1	String (3), fixed	ZMF library type of component. ***NOTE:*** May be masked using the asterisk (*) wildcard character.

CMPONENT APL_CDSC LIST — Reply

The Serena XML reply message for the component description *list* function returns zero to many `<result>` tags for the requested component(s). Each result contains the component description and library type in the reply.

The standard `<response>` data structure follows the `<result>` tags and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the final data structure in the reply, the `<response>` tag also serves as an end-of-list marker.

Data structure details for the list `<result>` tag are identical to those of CMPONENT APL_CDSC FIND; refer to *Exhibit 4-26*.

List Global Component Description - CMPONENT GBL_CDSC LIST

The Serena XML service/scope/message tags and attributes for messages that *list* the global description for a component are:

```
<service name="CMPONENT">
<scope name="GBL_CDSC">
<message name="LIST">
```

These tags appear in both requests and replies.

CMPONENT GBL_CDSC LIST — Request

Data structure details for the component global description list `<request>` tag appear in *Exhibit 4-28*.

Exhibit 4-28. CMPONENT GBL_CDSC LIST <request> Data Structure

Subtag	Use	Instances	Data Type & Length	Values
<component>	Required	1	String (256), variable	ZMF name of component. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally pre-fixed by path from installation root. ***NOTE:*** May be masked using the following wildcard characters: **?** = Any one alphanumeric character. ***** = Any number of alphanumeric characters. Used alone, it matches all values.
<componentType>	Required	1	String (3), fixed	ZMF library type of component. ***NOTE:*** May be masked using the asterisk (*) wildcard character.

CMPONENT GBL_CDSC LIST — Reply

The Serena XML reply message for the component global description *list* function returns zero to many `<result>` tags for the requested component(s). Each result contains the component global description and library type in the reply.

The standard `<response>` data structure follows the `<result>` tags and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the final data structure in the reply, the `<response>` tag also serves as an end-of-list marker.

Data structure details for the list `<result>` tag are similar to those of CMPONENT APL_CDSC FIND; the only difference is that the <applName> subtag is omitted. Refer to *Exhibit 4-26*.

List Component Promotion History - CMPONENT PRM_HIST LIST

A Serena XML message to *list* component promotion history has the following service/scope/message names:

```
<service name="CMPONENT">
<scope name="PRM_HIST">
<message name="LIST">
```

These tags appear in both request and reply messages.

CMPONENT PRM_HIST LIST — Requests

Serena XML provides component promotion history for selected components in a named package. Component name, component type, and promotion site name may be masked using a wildcard pattern.

Data structure details for the `<request>` tag of this message appear in *Exhibit 4-29*.

Exhibit 4-29. CMPONENT PRM_HIST LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks.
<component>	Optional	0 -1	String (256), variable	One ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root. ***NOTE:*** May be masked using asterisk (*) wildcard character.
<componentType>	Optional	0 -1	String (3), variable	ZMF component library type. ***NOTE:*** May be masked using asterisk (*) wildcard character.
<package>	Required	1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required.
<promotionSiteName>	Optional	0 -1	String (8), variable	Site where promotion library of interest resides. ***NOTE:*** May be masked using asterisk (*) wildcard character.

CMPONENT PRM_HIST LIST — Replies

The Serena XML reply message for a component promotion history list contains zero to many `<result>` tag, each of which provides promotion history and status information for one component that matches your request criteria.

The standard `<response>` data structure follows the final `<result>` tag and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the final data structure in the reply, the `<response>` tag also serves as an end-of-list marker.

Data structure details for the `<request>` tag of this message appear in *Exhibit 4-30*.

Exhibit 4-30. CMPONENT PRM_HIST LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<cleanupComponent>	Optional	0 - 1	String (1)	**Y** = Yes, clean up component **N** = No, don't clean up component
<component>	Optional	1	String (256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentType>	Optional	1	String (3), fixed	ZMF component library type.
<isComponentDeleted>	Optional	0 - 1	String (1)	**Y** = Yes, component deleted **N** = No, not deleted
<isComponentOverlaid>	Optional	0 - 1	String (1)	**Y** = Yes, component overlaid **N** = No, not overlaid
<isComponentRestaged>	Optional	0 - 1	String (1)	**Y** = Yes, component restaged **N** = No, component not restaged
<package>	Optional	1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.
<priorPromoter>	Optional	0 -1	String (8), variable	TSO ID of prior promoter.
<priorPromotionDate>	Optional	0 - 1	Date, yyyymmdd	Prior promotion/demotion date.
<priorPromotionLevel>	Optional	0 - 1	Integer (2), fixed	Numeric promotion level for prior component promotion library.

Exhibit 4-30. CMPONENT PRM_HIST LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<priorPromotionName>	Optional	0 - 1	String (8), variable	Name of library corresponding to hierarchical level number in `<priorPromotionLevel>` tag.
<priorPromotionTime>	Optional	0 - 1	Time, hhmmss	Prior promotion/demotion time.
<promoter>	Optional	0 - 1	String (8), variable	TSO ID of most recent promoter.
<promotionDate>	Optional	0 - 1	Date, yyyymmdd	Date component was last promoted.
<promotionJobStatus>	Optional	0 -1	String (1)	Code for status of promotion job. Values: **1** = Submitted **2** = Succeeded **3** = Failed
<promotionLevel>	Optional	0 - 1	Integer (2), fixed	Numeric promotion level for current component promotion library.
<promotionName>	Optional	0 - 1	String (8), variable	Name of library corresponding to hierarchical level number in `<promotionLevel>` tag.
<promotionSiteName>	Optional	0 - 1	String (8), variable	Promotion/demotion site.
<promotionTime>	Optional	0 - 1	Time, hhmmss	Time component was last promoted, in 24-hour format.

Component History List - CMPONENT HISTORY LIST

A Serena XML message to retrieve and list a comprehensive history of selected components has the following service/scope/message names:

```
<service name="CMPONENT">
<scope name="HISTORY">
<message name="LIST">
```

These tags appear in both request and reply messages.

CMPONENT HISTORY LIST — Request

Requests may be filtered by the following component status tags:

<baselinedStatus>

<checkedOutStatus>

<delArchStatus>

<deletedStatus>

<demotedStatus>

<promotedStatus>

Yes/no flags for component status filtering take default values as a group. The default changes based on whether or not you enter explicit values in these tags, as follows:

- If ***no*** status flag has an explicitly typed value, the default for all tags is "Y".
- If ***any*** status flag has an explicitly typed value, the default for the remaining tags is "N".

The following example shows how you might code a Serena XML request for a comprehensive component history list. Data structure details follow the example in *Exhibit 4-31*.

Example XML — CMPONENT HISTORY LIST Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="HISTORY">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <component>ACPSRS00</component>
    <componentType>SRS</componentType>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 4-31. CMPONENT HISTORY LIST <request>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name.
<baselinedStatus>	Optional	0 - 1	String (1)	**Y** = Include baselined components **N** = Omit baselined components ***NOTE:*** Part of component status flag group. If *no* tag in group has explicit value, default is Y. If *any* tag in group has explicit value, default is N.
<buildProc>	Optional	0 - 1	String (8), variable	8-byte ZMF name for designated build procedure.

Exhibit 4-31. CMPONENT HISTORY LIST <request> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<checkedOutStatus>	Optional	0 - 1	String (1)	**Y** = Include checked-out components **N** = Omit checked-out components ***NOTE:*** Part of component status flag group. If *no* tag in group has explicit value, default is Y. If *any* tag in group has explicit value, default is N.
<component>	Required	1	String (256), variable	ZMF component name. • If component is a PDS member, this is the member name (max 8 bytes, no qualifiers). • If component is an HFS file, this is the Unix-style long file name, optionally prefixed by path from installation root, max 256 bytes. ***NOTE:*** Component name may be masked using standard wildcards.
<componentType>	Required	1	String (3)	3-byte component library type to include in results. ***NOTE:*** Component type may be masked using standard wildcards.
<delArchStatus>	Optional	0 - 1	String (1)	**Y** = Include del-archived components **N** = Omit del-archived components ***NOTE:*** Part of component status flag group. If *no* tag in group has explicit value, default is Y. If *any* tag in group has explicit value, default is N.
<deletedStatus>	Optional	0 - 1	String (1)	**Y** = Include deleted components **N** = Omit deleted components ***NOTE:*** Part of component status flag group. If *no* tag in group has explicit value, default is Y. If *any* tag in group has explicit value, default is N.
<demotedStatus>	Optional	0 - 1	String (1)	**Y** = Include demoted components **N** = Omit demoted components ***NOTE:*** Part of component status flag group. If *no* tag in group has explicit value, default is Y. If *any* tag in group has explicit value, default is N.
<fromBaselinePkgDate>	Optional	0 -1	Date, yyyymmdd	Baseline package date.
<fromDateLastModified>	Optional	0 -1	Date, yyyymmdd	Start of range for desired component modification dates.
<language>	Optional	0 - 1	String (8), variable	Source language of component(s) to be compiled. If omitted, ZMF retrieves from component history.

Exhibit 4-31. CMPONENT HISTORY LIST <request> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<package>	Optional	0 - 1	String (10), variable	ZMF package name. ***NOTE:*** May be masked using standard wildcards.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.
<promotedStatus>	Optional	0 - 1	String (1)	**Y** = Include promoted components **N** = Omit promoted components ***NOTE:*** Part of component status flag group. If *no* tag in group has explicit value, default is Y. If *any* tag in group has explicit value, default is N.
<toBaselinePkgDate>	Optional	0 -1	Date, yyyymmdd	To baseline package date.
<toDateLastModified>	Optional	0 -1	Date, yyyymmdd	End of range desired component modification dates.
<updater>	Optional	0 -1	String (8), variable	TSO ID of last user to update component.

CMPONENT HISTORY LIST — Reply

The reply to a Serena XML component history list request returns zero to many `<result>` tags. Each `<result>` tag contains change history information about one component that meets the search criteria in your request. If one component appears in multiple packages, a `<result>` tag will appear for each instance of each component in each package.

A standard `<response>` data structure follows the `<result>` tags, if any, to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last data element returned in a Serena XML reply message, the `<response>` tag serves as an end-of-list marker.

An example Serena XML reply message for a component history list request appears below. Data structure details for the `<result>` tag follow in *Exhibit 4-32*.

Example XML — CMPONENT HISTORY LIST Reply

```
<?xml version=”1.0”?>
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="HISTORY">
  <message name="LIST">
   <result>
    <component>ACPSRS00</component>
    <componentType>SRS</componentType>
    <package>ACTP000002</package>
    <applName>ACTP</applName>
    <packageId>000002</packageId>
    <updater>SERT</updater>
    <dateLastModified>20081019</dateLastModified>
    <timeLastModified>204620</timeLastModified>
    <versionModLevel>0101</versionModLevel>
    <version>01</version>
    <modLevel>01</modLevel>
    <hashToken>E1C3A019000001FD</hashToken>
    <checkedOutStatus>N</checkedOutStatus>
    <deletedStatus>N</deletedStatus>
    <promotedStatus>N</promotedStatus>
    <demotedStatus>N</demotedStatus>
    <baselinedStatus>N</baselinedStatus>
    <delArchStatus>N</delArchStatus>
    <baselinePkgDate>20081019</baselinePkgDate>
    <baselinePkgTime>204620</baselinePkgTime>
    <language>COBOL2</language>
    <buildProc>CMNCOB2</buildProc>
    <linkOptions>NCAL</linkOptions>
    <useDb2PreCompileOption>N</useDb2PreCompileOption>
    <size>00000020</size>
    <setssi>5BCB7948</setssi>
   </result>
.
.
.
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 4-32. CMPONENT HISTORY LIST <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name.
<baselinePkgDate>	Optional	0 -1	Date, yyyymmdd	Baseline package date.
<baseLinePkgTime>	Optional	0 - 1	Time, hhmmss	Time component was last baselined, in 24-hour format.
<baselinedStatus>	Optional	0 - 1	String (1)	**Y** = Yes, component baselined **N** = No, component not baselined
<buildProc>	Optional	0 - 1	String (8), variable	8-byte ZMF name for designated build procedure.

Exhibit 4-32. CMPONENT HISTORY LIST <result> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<checkedOutStatus>	Optional	0 - 1	String (1)	**Y** = Yes, component checked out **N** = No, not checked out
<compileOptions>	Optional	0 - 1	String (34)	Compile options for component not stored elsewhere.
<component>	Required	1	String (256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentBuildNumber>	Optional	1	String (10), variable	Component build number
<componentType>	Required	1	String (3)	ZMF component library type.
<dateLastModified>	Optional	0 -1	Date, yyyymmdd	Date component was last changed.
<delArchStatus>	Optional	0 - 1	String (1)	**Y** = Yes, component del-archived. **N** = No, component not del-archived
<deletedStatus>	Optional	0 - 1	String (1)	**Y** = Yes, component deleted. **N** = No, component not deleted.
<demotedStatus>	Optional	0 - 1	String (1)	**Y** = Yes, component demoted. **N** = No, component not demoted.
<hashToken>	Optional	0 - 1	String (16), variable	**Hash Token**
<language>	Optional	0 - 1	String (8)	Name of source code language for component.
<linkOptions>	Optional	0 - 1	String (34)	Link options for component not stored elsewhere.
<modLevel>	Optional	0 - 1	Integer	Modification level
<package>	Optional	1	String (10)	ZMF name of package that includes this component. ***NOTE:*** If a component appears in multiple packages, a `<result>` tag for each package is returned with package-specific component change information.
<packageId>	Optional	0 - 1	Integer (6)	ZMF package ID number. Same as last 6 bytes of package name.
<processingtype>	Optional	0 - 1	String (1)	**Processing type.**
<promotedStatus>	Optional	0 - 1	String (1)	**Y** = Yes, component promoted **N** = No, not promoted
<promoter>	Optional	0 - 1	String (8)	TSO ID of most recent promoter.

Exhibit 4-32. CMPONENT HISTORY LIST <result> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<promotionDate>	Optional	0 - 1	Date, yyyymmdd	Date component was last promoted.
<promotionLevel>	Optional	0 - 1	Integer	Numeric promotion level for current component promotion library.
<promotionName>	Optional	0 - 1	String (8)	Name of library corresponding to hierarchical level number in `<promotionLevel>` tag.
<promotionSite>	Optional	0 - 1	String (8), variable	Promotion site
<promotionTime>	Optional	0 - 1	Time, hhmmss	Time component was last promoted, in 24-hour format.
<renameFrom>	Optional	0 - 1	String (256), variable	Rename FROM component name.
<renameTo>	Optional	0 - 1	String (256), variable	Rename TO component name.
<setssi>	Optional	0 - 1	String (8)	Component SETSII date (seconds since 1/1/1960)
<size>	Optional	0 - 1	Integer(8)	Number of lines of code.
<timeLastModified>	Optional	0 -1	Time, hhmmss	Time component was last changed, in 24-hour format.
<updater>	Optional	0 -1	String (8), variable	TSO ID of last user to update component.
<useDb2PreCompileOption>	Optional	0 - 1	String (1)	**Y** = Yes, use DB2 precompile **N** = No, don't precompile for DB2
<userOption01> . . . <userOption20>	Optional	0 - 1	String (1)	Set of up to 20 one-byte, custom, administrator-defined variables. Values: **Y** = Yes **N** = No
<userOption0101> . . . <userOption0105>	Optional	0 - 1 each	String (1), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0101 to 0105 on the ISPF user options panel for component build.
<userOption0201> . . . <userOption0203>	Optional	0 - 1 each	String (2), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0201 to 0203 on the ISPF user options panel for component build.

Exhibit 4-32. CMPONENT HISTORY LIST <result> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<userOption0301> . . . <userOption0303>	Optional	0 - 1 each	String (3), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0301 to 0303 on the ISPF user options panel for component build.
<userOption0401> . . . <userOption0403>	Optional	0 - 1 each	String (4), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0401 to 0403 on the ISPF user options panel for component build.
<userOption0801> . . . <userOption0805>	Optional	0 - 1 each	String (8), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 0801 to 0805 on the ISPF user options panel for component build.
<userOption1001> . . . <userOption1002>	Optional	0 - 1 each	String (10), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 1001 to 1002 on the ISPF user options panel for component build.
<userOption1601> . . . <userOption1602>	Optional	0 - 1 each	String (16), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 1601 to 1603 on the ISPF user options panel for component build.
<userOption3401> . . . <userOption3402>	Optional	0 - 1 each	String (34), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 3401 to 3402 on the ISPF user options panel for component build.
<userOption4401> . . . <userOption4402>	Optional	0 - 1 each	String (44), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 4401 to 4402 on the ISPF user options panel for component build.
<userOption6401> . . . <userOption6405>	Optional	0 - 1 each	String (64), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 6401 to 6405 on the ISPF user options panel for component build.
<userOption7201> . . . <userOption7205>	Optional	0 - 1 each	String (72), variable	Administrator-defined build options assigned to component. Each tag corresponds to User Option 7201 to 7205 on the ISPF user options panel for component build.

Exhibit 4-32. CMPONENT HISTORY LIST <result> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<userOptionsPart1>	Optional	0 - 10	String (10)	Set of up to 10 one-byte, custom, administrator-defined variables. User options 01-10. Values: **Y** = Yes **N** = No
<userOptionsPart2>	Optional	0 - 10	String (10)	Set of up to 10 one-byte, custom, administrator-defined variables. User options 11 - 20. Values: **Y** = Yes **N** = No
<version>	Optional	0 - 1	Integer	ISPF version of component.
<versionModLevel>	Optional	0 - 1	Integer	ISPF modification level for component.

Tip

Tags: <userOption01> to <userOption20>, <userOptionsPart1>, <userOptionsPart2>, <userOption0101> to <userOption7205>. See topic "Staging User Options" in the ChangeMan ZMF Customization Guide.

List Short Component History - CMPONENT HISTORY LISTSHRT

The Serena XML service/scope/message names in messages to retrieve and *list* the short version of selected component history records are:

```
<service name="CMPONENT">
<scope name="HISTORY">
<message name="LISTSHRT">
```

These tags appear in both request and reply messages.

CMPONENT HISTORY LISTSHRT — Requests

The short form of the component history list request is similar to the comprehensive component history list function (CMPONENT HISTORY LIST). The differences for the short form are:

- The name attribute of the `<message>` tag is "LISTSHRT".
- The `<component>` and `<componentType>` tag values may not be masked using wildcard patterns; you must enter a single component name and component type.
- The `<package>` and `<packageId>` tags are omitted.

Data structure details for the `<request>` tag of the CMPONENT HISTORY LIST message appeared previously in *Exhibit 4-31*.

CMPONENT HISTORY LISTSHRT — Replies

Other than the name attribute of the `<message>` tag, replies to a short component history list are identical in syntax to a comprehensive component history list. Data structure details for the `<result>` tag of this message appeared previously in *Exhibit 4-32*.

List Current Component History - CMPONENT HISTORY LISTCURR

The Serena XML service/scope/message names in messages to retrieve and *list* component history for the current, active version of a selected component are:

```
<service name=”CMPONENT”>
<scope name=”HISTORY”>
<message name=”LISTCURR”>
```

These tags appear in both request and reply messages.

CMPONENT HISTORY LISTCURR — Requests

Data structure details for this request message appear in *Exhibit 4-33*.

Exhibit 4-33. CMPONENT HISTORY LISTCURR <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name.
<component>	Required	1	String (256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentType>	Required	1	String (3), fixed	ZMF component library type.
<package>	Optional	0 - 1	String (10), variable	ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.

CMPONENT HISTORY LISTCURR — Replies

Other than the name attribute of the `<message>` tag, reply messages for a current component history list are identical to those for a comprehensive component history list. Data structure details for the `<result>` tag of this message appeared previously in *Exhibit 4-32*.

List Concurrent Comp. History - CMPONENT HISTORY LISTCONC

The Serena XML service/scope/message names in messages to retrieve and *list* history records for components in concurrent development are:

```
<service name="CMPONENT">
<scope name="HISTORY">
<message name="LISTCONC">
```

These tags appear in both request and reply messages.

CMPONENT HISTORY LISTCONC — Requests

Data structure details for this request message appear in *Exhibit 4-34*.

Exhibit 4-34. CMPONENT HISTORY LISTCONC <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	1	String (4), variable	ZMF application name.
<component>	Required	1	String (256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentType>	Required	1	String (3), fixed	ZMF component library type.

CMPONENT HISTORY LISTCONC — Replies

Other than the name attribute of the `<message>` tag, and the additional tag `<packageType>`, the reply messages for a concurrent component history list are identical to those for a comprehensive component history list. Data structure details for the `<result>` tag of this message appeared previously in *Exhibit 4-32*.

The `<packageType>` tag is a one-byte string and may contain a value of 1, 2, 3, or 4:

1 = Planned Permanent

2 = Planned Temporary

3 = Unplanned Permanent

4 = Unplanned Temporary

List Baselined Component History - CMPONENT HISTORY LISTBASE

The Serena XML service/scope/message names in messages to retrieve and *list* component history for the baselined version of a selected component are:

```
<service name="CMPONENT">
<scope name="HISTORY">
<message name="LISTBASE">
```

These tags appear in both request and reply messages.

CMPONENT HISTORY LISTBASE — Requests

Data structure details for this request message appear in *Exhibit 4-35*.

Exhibit 4-35. CMPONENT HISTORY LISTBASE <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	1	String (4), variable	ZMF application name.
<component>	Required	1	String (256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentType>	Required	1	String (3), fixed	ZMF component library type.

CMPONENT HISTORY LISTBASE — Replies

Other than the name attribute of the `<message>` tag, reply messages for a baselined component history list are identical to those for a comprehensive component history list. Data structure details for the `<result>` tag of this message appeared previously in *Exhibit 4-32*.

List Comp. User Worklist Records - CMPONENT PKG_WRKL LIST

The Serena XML service/scope/message names in messages to *list* user work records for developers who have worked on a component are:

```
<service name="CMPONENT">
<scope name="PKG_WRKL">
<message name="LIST">
```

These tags appear in both request and reply messages.

CMPONENT PKG_WRKL LIST — Requests

Requests for a list of user work records concerning a component in a package — the so-called ICWK records in the package master — consist almost entirely of required tags. You must supply the package name, component name, component type, and the TSO ID of the component updater whose work records you want to list. No wildcard patterns for these values are supported in Serena XML.

Example XML — CMPONENT PKG_WRKL LIST Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="PKG_WRKL">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000007</package>
    <component>ACPCPY00</component>
    <componentType>CPY</componentType>
    <updater>USER24</updater>
   </request>
  </message>
 </scope>
</service>
```

Data structure details for the `<request>` tag of this request appear in *Exhibit 4-36*.

Exhibit 4-36. CMPONENT PKG_WRKL LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks.
<component>	Required	1	String (256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentType>	Required	1	String (3), fixed	ZMF component library type.
<package>	Required	1	String (10), variable	Fixed-format ZMF package name.

Exhibit 4-36. CMPONENT PKG_WRKL LIST <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required.
<updater>	Required	1	String (8), variable	TSO ID of component updater.

List Component User Worklist Records — Replies

The Serena XML reply for a component user work record list request returns zero to many `<result>` tags. Each `<result>` tag contains information about the last action taken by the user on the component in question, the date and time of that action, a flag identifying this user as the last user to modify the component (if applicable), and the last action performed on the component by any user. Other control information, if any, related to the user audit trail also appears in the `<result>`.

A standard `<response>` data structure follows the `<result>` tags, if any, to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last data element returned in a Serena XML reply message, the `<response>` tag serves as an end-of-list marker.

Example XML — CMPONENT PKG_WRKL LIST Reply

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="PKG_WRKL">
  <message name="LIST">
   <result>
    <package>ACTP000007</package>
    <applName>ACTP</applName>
    <packageId>000007</packageId>
    <component>ACPCPY00</component>
    <componentType>CPY</componentType>
    <updater>USER24</updater>
    <lastAction>9</lastAction>
    <setssi>5C4EBCC3</setssi>
    <updateDate>20090127</updateDate>
    <updateTime>082643</updateTime>
    <useCount>0001</useCount>
    <isComponentDeleted>N</isComponentDeleted>
    <mostRecentUpdate>Y</mostRecentUpdate>
   </result>
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Data structure details for the `<result>` tag follow in *Exhibit 4-37*.

Exhibit 4-37. CMPONENT PKG_WRKL LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<component>	Optional	1	String (256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentType>	Optional	1	String (3), fixed	ZMF component library type.
<isComponentDeleted>	Optional	0 - 1	String (1)	**Y** = Yes, component deleted **N** = No, component not deleted
<lastAction>	Optional	0 - 1	Integer (1)	Code for user function performed by last component updater. Values: **1** = Browse **2** = Check Out **3** = Create **4** = Delete **5** = Edit **6** = Edit & Stage **7** = Recompile **8** = Relink **9** = Stage **A** = Update **B** = Checkin **C** = Build
<mostRecentUpdate>	Optional	0 - 1	String (1)	**Y** = this is the most recent update **N** = this is not the most recent update
<package>	Optional	1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.
<setssi>	Optional	0 - 1	String (8), variable	setssi.
<updateDate>	Optional	0 -1	Date, yyyymmdd	Update date.

Exhibit 4-37. CMPONENT PKG_WRKL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<updateTime>	Optional	0 -1	Time, hhmmss	Update time
<updater>	Optional	0 - 1	String (8), variable	Userid of updater.
<usecount>	Optional	0 - 1	Integer(4)	Use count

COMPONENT SECURITY TASKS

For general use, Serena XML supports the following application-level component security tasks:

- *Check Component Security - CMPONENT APL_SECR CHECK*
- *Find Component Authorized Users - CMPONENT APL_SECR FIND*
- *List Component Authorized Users - CMPONENT APL_SECR LIST*

These application-level security tasks share the following scope `name` attribute:

```
<scope name="apl_secr">
```

Serena XML supports the following global-level component security task:

List Global Component Authorized Users - CMPONENT GBL_SECR LIST

This global-level security task has the following scope `name` attribute:

```
<scope name="gbl_secr">
```

Check Component Security - CMPONENT APL_SECR CHECK

Serena XML lets you determine whether a specific user is authorized to access a particular component. The service/scope/message `name` attributes for this message are:

```
<service name="CMPONENT">
<scope name="APL_SECR">
<message name="CHECK">
```

These tags appear in both request and reply messages.

CMPONENT APL_SECR CHECK — Requests

Example XML — CMPONENT APL_SECR CHECK Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="APL_SECR">
```

```
  <message name="CHECK">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <component>ACPSRS00</component>
    <componentType>SRS</componentType>
    <applName>ACTP</applName>
    <user>USER24</user>
   </request>
  </message>
 </scope>
</service>
```

CMPONENT APL_SECR CHECK — Requests

A request to check a user's security authorization for access to a component must satisfy the data structure requirements in *Exhibit 4-38*.

Exhibit 4-38. Check Component Security <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<component>	Required	1	String (256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentType	Required	1	String (3), variable	ZMF component library type.
<user>	Optional	1	String (8), variable	TSO ID of user to be checked for authorization to access component.

CMPONENT APL_SECR CHECK — Replies

No `<result>` tag is returned in response to a request to check a user's authorization to access a component. However, the standard `<response>` data structure of the reply message contains the necessary information in the `<statusReturnCode>` tag and its associated `<statusMessage>` tag.

CMPONENT APL_SECR CHECK — Reply

Example XML — CMPONENT APL_SECR CHECK Reply

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="APL_SECR">
  <message name="CHECK">
   <response>
    <statusMessage>CMN8700I - CHECK service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
</service>
```

For more information about specific return code values and what they mean in the context of this function, see the documentation for your mainframe security system.

Find Component Authorized Users - CMPONENT APL_SECR FIND

A Serena XML message to *find* authorized users for a specific component in an application has the following service/scope/message `name` attributes:

```
<service name=”CMPONENT”>
<scope name=”APL_SECR”>
<message name=”FIND”>
```

These tags appear in both request and reply messages.

CMPONENT APL_SECR FIND — Requests

Example XML — CMPONENT APL_SECR FIND Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="APL_SECR">
  <message name="FIND">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <component>ACPSRS00</component>
    <componentType>SRS</componentType>
    <applName>ACTP</applName>
  </request>
  </message>
 </scope>
</service>
```

The `<request>` tag in a Serena XML request to find the authorized users of a specific component in an application has the data structure requirements in *Exhibit 4-39*.

Exhibit 4-39. CMPONENT APL_SECR FIND <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<component>	Required	1	String (256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentType	Required	1	String (3), variable	ZMF component library type.

CMPONENT APL_SECR FIND— Reply

The Serena XML reply message for a request to find the authorized users of a specific component in an application returns zero to many `<result>` tags. Each `<result>` tag contains security information for a particular TSO user ID authorized to access the named component.

A standard `<response>` data structure follows the `<result>` tags, if any, to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last data element returned in a Serena XML reply message, the `<response>` tag serves as an end-of-list marker.

Example XML — CMPONENT APL_SECR FIND Reply

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="APL_SECR">
  <message name="FIND">
   <result>
    <component>ACPSRS00</component>
    <componentType>SRS</componentType>
    <applName>ACTP</applName>
    <user>USER24</user>
    <isEntity>N</isEntity>
   </result>
```

```
    <response>
     <statusMessage>CMN8700I - FIND service completed</statusMessage>
     <statusReturnCode>00</statusReturnCode>
     <statusReasonCode>8700</statusReasonCode>
    </response>
  </message>
 </scope>
</service>
```

Data structure details for the <result> tag are identical to those for CMPONENT APL_SECR LIST. See *Exhibit 4-41*.

List Component Authorized Users - CMPONENT APL_SECR LIST

A Serena XML message to *list* authorized users for components in a particular application has the following service/scope/message `name` attributes:

```
<service name="CMPONENT">
<scope name="APL_SECR">
<message name="LIST">
```

These tags appear in both request and reply messages.

CMPONENT APL_SECR LIST — Requests

Example XML — CMPONENT APL_SECR LIST Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="APL_SECR">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <component>ACPSRS00</component>
    <componentType>SRS</componentType>
    <applName>ACTP</applName>
   </request>
  </message>
 </scope>
</service>
```

The `<request>` tag in a Serena XML request to list the authorized users of a component in a particular application has the data structure requirements in *Exhibit 4-40*.

Exhibit 4-40. CMPONENT APL_SECR LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<component>	Required	1	String (256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root. ***NOTE:*** May be masked using asterisk (*) wildcard character.
<componentType	Required	1	String (3), variable	ZMF component library type.
<exactMatch>	Optional	0 - 1	String (1)	**Y** = exact match, no filtering **N** = filtering, no exactmatch

CMPONENT APL_SECR LIST— Reply

The Serena XML reply message for a request to list the authorized users of a component in a particular application returns zero to many `<result>` tags. Each `<result>` tag contains security information for a particular TSO user ID authorized to access the named component.

A standard `<response>` data structure follows the `<result>` tags, if any, to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last data element returned in a Serena XML reply message, the `<response>` tag serves as an end-of-list marker.

Example XML — CMPONENT APL_SECR LIST Reply

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="APL_SECR">
  <message name="LIST">
   <result>
    <component>ACPSRS00</component>
    <componentType>SRS</componentType>
    <applName>ACTP</applName>
    <user>USER24</user>
    <isEntity>N</isEntity>
   </result>
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Data structure details for the <result> tag appear in *Exhibit 4-41*.

Exhibit 4-41. CMPONENT APL_SECR LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<component>	Optional	1	String (256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentType>	Optional	1	String (3), fixed	ZMF component library type.
<isEntity>	Optional	1	String (1)	**Y** = Yes, security entity (group) **N** = No, not a security entity
<user>	Optional	1	String (8), variable	TSO ID of authorized component user or security entity (group).

List Global Component Authorized Users - CMPONENT GBL_SECR LIST

A Serena XML message to *list* authorized users for components in all applications has the following service/scope/message `name` attributes:

```
<service name="CMPONENT">
<scope name="GBL_SECR">
<message name="LIST">
```

These tags appear in both request and reply messages.

CMPONENT GBL_SECR LIST — Requests

Example XML — CMPONENT GBL_SECR LIST Request

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="GBL_SECR">
```

```
   <message name="LIST">
    <header>
     <subsys>8</subsys>
     <product>CMN</product>
    </header>
   <request>
     <component>ACPSRS00</component>
     <componentType>SRS</componentType>
    </request>
   </message>
  </scope>
</service>
```

The `<request>` tag in a Serena XML request to list the authorized users of a component in any application has the data structure requirements in *Exhibit 4-42*.

Exhibit 4-42. CMPONENT GBL_SECR LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<component>	Required	1	String (256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root. ***NOTE:*** May be masked using asterisk (*) wildcard character.
<componentType	Required	1	String (3), variable	ZMF component library type.
<exactMatch>	Optional	0 - 1	String (1)	**Y** = exact match, no filtering **N** = filtering, no exactmatch

CMPONENT GBL_SECR LIST— Reply

The Serena XML reply message for a request to list the authorized users of a component in any application returns zero to many `<result>` tags. Each `<result>` tag contains security information for a particular TSO user ID authorized to access the named component.

A standard `<response>` data structure follows the `<result>` tags, if any, to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last data element returned in a Serena XML reply message, the `<response>` tag serves as an end-of-list marker.

Example XML — CMPONENT GBL_SECR LIST Reply

```
<?xml version="1.0"?>
<service name="CMPONENT">
 <scope name="GBL_SECR">
  <message name="LIST">
   <result>
    <component>ACPSRS00</component>
    <componentType>SRS</componentType>
    <user>USER24</user>
    <isEntity>N</isEntity>
   </result>
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Data structure details for the <result> tag appear in *Exhibit 4-43*.

Exhibit 4-43. CMPONENT GBL_SECR LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<component>	Optional	1	String (256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<componentType>	Optional	1	String (3), fixed	ZMF component library type.
<isEntity>	Optional	1	String (1)	**Y** = Yes, security entity (group) **N** = No, not a security entity
<user>	Optional	1	String (8), variable	TSO ID of authorized component user or security entity (group).

Search, Summary, and Analysis Tasks

5

XML Services supports search, summary, and analysis tasks by developers, managers, and administrators. The scope of these services encompasses multiple packages, multiple components, or multiple servers. Most support complex filters, flags, and processing options that require some experience using ChangeMan ZMF. All can be performed interactively or in batch mode using the appropriate Serena XML client application.

The following tasks are supported for general use:

- ***Package Search and Summary Tasks*** — Query multiple packages, search for packages awaiting approval, or compute multi-package summary statistics. Typical commands are *search* and *summary*.
- ***Audit Trail Management*** — Work with ChangeMan ZMF log files. Typical commands include *create* and *list*.
- ***Impact Analysis Functions*** — Component Impact Analysis.

Syntax Conventions for Search, Summary, and Analysis

Serena XML adopts certain syntax conventions that apply in all search, summary, and analysis contexts. For example, a few commonly used tags support alternate uses or alternate data type conventions in a search, summary, or analysis context. Other conventions involve data dependencies and default value interactions among multiple, related tags.

Semicolon-Delimited Lists

The allowed contents of commonly used tags may be different in search contexts than in non-search contexts. For example, many user ID and security entity tags, which elsewhere permit only a single value, may accept a semicolon-delimited list of TSO user IDs or RACF security entities in a search context. Similar lists may be allowed in package or component name tags as well. For the services discussed in this chapter, the following tags support such lists:

- `<component>`
- `<package>`
- `<creator>`
- `<stager>`
- `<workRequest>`

Yes/No Flag Tags

All search, summary, and analysis services in Serena XML follow common default value conventions for yes/no flag tags. They also share common conventions for Boolean relationships among flag tag values.

The key to these conventions is the flag tag *group*. The values of all flag tags within a group are considered together; in fact, such mutual processing is the basis for identifying such tags as a group. Many flag tag groups may be supported by a single service. For example, all package status flag tags are treated as a group, as are all package type tags, and package level tags. Flag tag groups, however, are evaluated independently of each other.

Default Values Within a Group

All yes/no flag tags in a group default to the value "Y" if no tag in the group has an explicitly assigned value. But if *any* flag tag in the group is explicitly assigned a value, all other tags in the group change their default values to "N."

This is actually the behavior you would expect if you weren't thinking too hard about it. For example, if your XML request says nothing about which package types to include, the package type flag tags all default to "Y" values, and you will get *all* package types in your reply. But if your XML request explicitly states you want all planned permanent packages, the assumption is made that you *don't* want any other package types. The flag tag value that you have set to "Y" retains that value, but the defaults for all other flag tags in the group default to "N". To request multiple package types, you would set the yes/no flags for each desired package type to an explicit "Y" value.

Tip

If you explicitly set flag tag values, always include at least one "Y" value in each flag tag group. This is necessary because Serena XML does not fully support Boolean NOT operations. You can *include* an item in a given state, and you can *exclude* an item in a given state, but you cannot include in your search results an item that is *not* in a given state. If you explicitly set just one flag tag in a group to a value of "N" and take the defaults for the rest, no results will be returned.

Boolean Relationships Within a Group

The "Y" values of all tags within a yes/no flag tag group are related by Boolean OR in search, summary, or analysis contexts. Any item in any state requested by a "Y" flag is returned in the results.

Tags with a value of "N" in a group are related by Boolean AND to the other tags in the group. Any item in any state identified by a flag with a value of "N" is excluded from the results.

Package Search and Summary Tasks

Serena XML supports several types of multi-package queries and statistical analysis for general use. These are:

- *General Package Search - PACKAGE GENERAL SEARCH*
- *Search for Limbo Packages - PACKAGE LIMBO SEARCH*
- *Search for Packages Pending Approval - PACKAGE APPROVE SEARCH*
- *Search for Linked Packages - PACKAGE PKG_LINK SEARCH*
- *Package Summary Statistics - PACKAGE SERVICE SUMMARY*

General Package Search - PACKAGE GENERAL SEARCH

A general package search retrieves comprehensive information about one or more packages listed in the package master on the queried LPAR. Only one ChangeMan ZMF instance is included in the scope of this function. Limbo packages are also outside the scope of this function.

The Serena XML service/scope/message tags and attributes for messages to *search* for any package are:

```
<service name="PACKAGE">
<scope name="GENERAL">
<message name="SEARCH">
```

These tags appear in both requests and replies.

PACKAGE GENERAL SEARCH Requests

General package search criteria can include any of the following options:

- ***Semicolon-delimited name lists and/or wildcard patterns*** for package names, component names, user IDs, and work change request IDs.
- ***Yes/no flag tag sets*** for desired package level, type, status, and scheduler.
- ***Standalone yes/no flag tags*** for packages pending completion, packages with deleted staging libraries, packages checked into a release, packages linked to other packages, and the like.
- ***Date range criteria*** for lifecycle state history such as date frozen, date promoted, date approved, date installed, and date baselined. Date ranges need not include an end-of-range date if all packages after the given start date are desired in the results.
- ***Selective package search criteria*** for packages that share a common install site, requestor department, approval entity, audit return code, package master record type, component library type, online form number, complex/super package name, linked package name, or release name.
- ***String search*** for words in the package title.

The following example shows two of these options used in combination, it requests a list of packages that begin with ACTP, that are also in the Approval Pending state.

Example XML — PACKAGE GENERAL SEARCH Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="GENERAL">
  <message name="SEARCH">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP*</package>
    <searchForApprovalPending>Y</searchForApprovalPending>
   </request>
  </message>
 </scope>
</service>
```

Details of the Serena XML general package search `<request>` data structure appear in *Exhibit 5-1*.

Exhibit 5-1. PACKAGE GENERAL SEARCH <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<approvalEntity>	Optional	0 - 1	String (8)	TSO user ID of approval entity.
<auditLockUserid>	Optional	0 - 1	String (8)	TSO user ID that locked package for audit.
<auditReturnCode>	Optional	0 - 1	String (2), variable	Return code issued by package audit function. Values: **00** = No major errors found **04** = Errors found, fix suggested **08** = Major errors found **12** = Possibly fatal errors found
<complexSuperPackage>	Optional	0 - 1	String (255), variable	ZMF name of complex or super package to which a participating package belongs. NOTE: Returned only if value in `<packageLevel>` = 4.

Exhibit 5-1. PACKAGE GENERAL SEARCH <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<component>	Optional	0 - 1	String (256), variable	ZMF component names in package. ***NOTE:*** A maximum of 25 names may be specified, delimited by semicolons. The total length of the <component> subtag may not exceed 256 bytes. Each component name may be masked using asterisk (*) wildcard. • If components are PDS members, each member name is max 8 bytes, no qualifiers. • If components are HFS files, each component has Unix-style long file name, optionally prefixed by path from installation root. Each name may have a maximum of 64 bytes, but the total length of the <component> subtag may not exceed 256 bytes.
<componentType>	Optional	0 - 1	String (3)	ZMF component type in package.
<creator>	Optional	0 -1	String (255), variable	One or more 8-byte TSO user IDs for package creators, delimited by semicolons. ***NOTE:*** Each creator ID in list may be masked using asterisk (*) wildcard.
<formNumber>	Optional	0 - 1	String (3)	ZMF online form number in package.
<lastPromoter>	Optional	0 - 1	String (8), variable	TSO user ID of most recent promoter/ demoter of desired package(s).
<lastPromotionLevel>	Optional	0 - 1	String (2), variable	Most recent numeric promotion level of package in promotion hierarchy.
<lastPromotionName>	Optional	0 - 1	String (8), variable	Name corresponding to code in `<lastPromotionLevel>`.
<lastPromotionSite>	Optional	0 - 1	String (8), variable	ZMF name of site where most recent promotion action took place.
<linkPackage>	Optional	0 - 1	String (255), variable	One or more ZMF link package name(s) of up to 10 bytes each, delimited by semicolons. ***NOTE:*** Each package name in list may be masked using asterisk (*) wildcard.
<linkRequestor>	Optional	0 -1	String (20), variable	Link requestor.
<package>	Required	0 - 1	String (255), variable	One or more ZMF package name(s) of up to 10 bytes each, delimited by semicolons. ***NOTE:*** Each package name in list may be masked using asterisk (*) wildcard.

Exhibit 5-1. PACKAGE GENERAL SEARCH <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<packageTitle>	Optional	0 -1	String (255), variable	Exact search words for text search in package working title.
<release>	Optional; ZMF with ERO only	0 -1	String (255)	Name of release to which desired packages are attached.
<requestorDept>	Optional	0 -1	String (4), variable	Workgroup or department code for desired packages.
<requestorName>	Optional	0 -1	String (25), variable	Name of developer or contact person responsible for current work status of package(s) in motion.
<requestorPhone>	Optional	0 -1	String (15), variable	Phone number of developer or contact person responsible for current work status of package(s) in motion.
<searchForApprovalPending>	Optional	0 -1	String (1)	**Y** = include approval pending pkgs **N** = omit approval pending pkgs
<searchForApprovedStatus>	Optional	0 -1	String (1)	**Y** = include approved packages **N** = omit approved packages ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is “Y”. If *any* yes/no filter tag for package status is included, default value is “N”.
<searchForAuditPending>	Optional	0 -1	String (1)	**Y** = include audit pending pkgs **N** = omit audit pending pkgs
<searchForBackedOutStatus>	Optional	0 -1	String (1)	**Y** = Yes, include backed out pkgs **N** = No, omit backed out pkgs ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is “Y”. If *any* yes/no filter tag for package status is included, default value is “N”.
<searchForBackoutPending>	Optional	0 -1	String (1)	**Y** = include backout pending pkgs **N** = omit backout pending pkgs

Exhibit 5-1. PACKAGE GENERAL SEARCH <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<searchForBaselineStatus>	Optional	0 -1	String (1)	**Y** = Yes, include baselined pkgs **N** = No, omit baselined pkgs ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is "Y". If *any* yes/ no filter tag for package status is included, default value is "N".
<searchForBuildXNodePending>	Optional	0 -1	String (1)	**Y** = include build xnode pending pkgs **N** = omit build xnode pending pkgs
<searchForClosedStatus>	Optional	0 -1	String (1)	**Y** = Yes, include closed packages **N** = No, omit closed packages ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is "Y". If *any* yes/ no filter tag for package status is included, default value is "N".
<searchForCmnScheduler>	Optional	0 - 1	String (1)	**Y** = Yes, find packages that use ChangeMan scheduler **N** = No, omit ChangeMan scheduler ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is "Y". If *any* yes/ no filter tag for package status is included, default value is "N"
<searchForComplexLevel>	Optional	0 -1	String (1)	**Y** = Yes, include complex packages **N** = No, omit complex packages ***NOTE:*** Part of ***package level tag group***. If *all* yes/no filter tags for package level are omitted from request, default value is "Y". If *any* yes/no filter tag for package level is included, default value is "N".
<searchForCustComponents>	Optional	0 - 1	String (1)	**Y** = find packages with custom comp. **N** = omit packages with custom comp.
<searchForDeletedStageLib>	Optional	0 - 1	String (1)	**Y** = Yes, find deleted staging libs **N** = No, omit deleted staging libs

Exhibit 5-1. PACKAGE GENERAL SEARCH <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<searchForDeletedStatus>	Optional	0 -1	String (1)	**Y** = Yes, include deleted packages **N** = No, omit deleted packages ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is “Y”. If *any* yes/no filter tag for package status is included, default value is “N”.
<searchForDeliveredStatus> ***NOTE:*** In the context of this service, “delivered” means “distributed.”	Optional	0 -1	String (1)	**Y** = Yes, include distributed pkgs **N** = No, omit distributed pkgs ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is “Y”. If *any* yes/no filter tag for package status is included, default value is “N”.
<searchForDevelopmentStatus>	Optional	0 -1	String (1)	**Y** = Yes, include development pkgs **N** = No, omit development pkgs ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is “Y”. If *any* yes/no filter tag for package status is included, default value is “N”.
<searchForFreezePending>	Optional	0 - 1	String (1)	**Y** = find packages that are freeze pending **N** = omit packages that are freeze pending
<searchForFrozenStatus>	Optional	0 -1	String (1)	**Y** = Yes, include frozen packages **N** = No, omit frozen packages ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is “Y”. If *any* yes/no filter tag for package status is included, default value is “N”.
<searchForInstallPending>	Optional	0 - 1	String (1)	**Y** = find packages that are install pending **N** = omit packages that are install pending

Exhibit 5-1. PACKAGE GENERAL SEARCH <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<searchForInstalledStatus>	Optional	0 -1	String (1)	**Y** = Yes, include installed packages **N** = No, omit installed packages ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is "Y". If *any* yes/no filter tag for package status is included, default value is "N".
<searchForLinkedPackage>	Optional	0 - 1	String (1)	**Y** = Yes, find linked packages **N** = No, omit linked packages
<searchForLoadComponents>	Optional	0 - 1	String (1)	**Y** = Find load components **N** = Omit load components
<searchForManualScheduler>	Optional	0 - 1	String (1)	**Y** = Yes, find packages that require manual installation **N** = No, omit manual install packages ***NOTE:*** Part of ***package scheduler tag group***. If *all* yes/no filter tags for package scheduler are omitted from request, default value is "Y". If *any* yes/no filter tag for package scheduler is included, default value is "N".
<searchForNonSource Components>	Optional	0 - 1	String (1)	**Y** = Yes, find packages with non source components **N** = No, omit packages with non source components
<searchForOpenedStatus>	Optional	0 -1	String (1)	**Y** = Yes, include opened packages **N** = No, omit opened packages ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is "Y". If *any* yes/no filter tag for package status is included, default value is "N".
<searchForOtherScheduler>	Optional	0 - 1	String (1)	**Y** = Yes, find packages that used 3rd-party installation scheduler **N** = No, omit packages that use 3rd-party scheduler ***NOTE:*** Part of ***package scheduler tag group***. If *all* yes/no filter tags for package scheduler are omitted from request, default value is "Y". If *any* yes/no filter tag for package scheduler is included, default value is "N".

Exhibit 5-1. PACKAGE GENERAL SEARCH <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<searchForPackageChecked IntoRelease>	Optional; ZMF with ERO only	0 -1	String (1)	**Y** = Yes, find packages checked into a release area. **N** = No, omit packages checked into a release area.
<searchForPartLevel>	Optional	0 -1	String (1)	**Y** = Yes, include participating pkgs **N** = No, omit participating pkgs ***NOTE:*** Part of ***package level tag group***. If *all* yes/no filter tags for package level are omitted from request, default value is “Y”. If *any* yes/no filter tag for package level is included, default value is “N”.
<searchForPlannedPermType>	Optional	0 -1	String (1)	**Y** = Yes, include planned perm pkgs **N** = No, omit planned perm pkgs ***NOTE:*** Part of ***package type tag group***. If *all* yes/no filter tags for package type are omitted from request, default value is “Y”. If *any* yes/no filter tag for package type is included, default value is “N”.
<searchForPlannedTempType>	Optional	0 -1	String (1)	**Y** = Yes, include planned temp pkgs **N** = No, omit planned temp pkgs ***NOTE:*** Part of ***package type tag group***. If *all* yes/no filter tags for package type are omitted from request, default value is “Y”. If *any* yes/no filter tag for package type is included, default value is “N”.
<searchForPostApproval Pending>	Optional	0 - 1	String (1)	**Y** = Find post approval pending pkgs **N** = Omit post approval pending pkgs
<searchForPostApprovers Added>	Optional	0 - 1	String (1)	**Y** = Yes, find packages with newly added post-approvers **N** = No, omit packages with newly added post-approvers
<searchForPostRejected>	Optional	0 - 1	String (1)	**Y** = find packages rejected in post approval **N** = omit packages rejected in post approval

Exhibit 5-1. PACKAGE GENERAL SEARCH <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<searchForRecordType>	Optional	0 - 1	String (1)	Search all staging libraries for a specific record type in the package master. Values: **4** = Source or non-source **5** = Load component **6** = Non-source component **7** = Online forms or custom comp **8** = Rename utility component **9** = Scratch utility component **A** = Source component **B** = Utility component
<searchForRejectedStatus>	Optional	0 -1	String (1)	**Y** = Yes, include rejected packages **N** = No, omit rejected packages ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is "Y". If *any* yes/no filter tag for package status is included, default value is "N".
<searchForRename Components>	Optional	0 - 1	String (1)	**Y** = find packages with renamed components **N** = omit packages with renamed components
<searchForRevertPending>	Optional	0 - 1	String (1)	**Y** = Find revert pending pkgs **N** = Omit revert pending pkgs
<searchForScratch Components>	Optional	0 - 1	String (1)	**Y** = find packages with scratched components **N** = omit packages with scratched components
<searchForShortApproverList Used>	Optional	0 - 1	String (1)	**Y** = find packages using short approver list **N** = omit packages using short approver list
<searchForSimpleLevel>	Optional	0 -1	String (1)	**Y** = Yes, include simple packages **N** = No, omit simple packages ***NOTE:*** Part of ***package level tag group***. If *all* yes/no filter tags for package level are omitted from request, default value is "Y". If *any* yes/no filter tag for package level is included, default value is "N".
<searchForSource Components>	Optional	0 - 1	String (1)	**Y** = find packages with source components **N** = omit packages with source components

Exhibit 5-1. PACKAGE GENERAL SEARCH <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<searchForSuperLevel>	Optional	0 -1	String (1)	**Y** = Yes, include super packages **N** = No, omit super packages ***NOTE:*** Part of ***package level tag group***. If *all* yes/no filter tags for package level are omitted from request, default value is "Y". If *any* yes/no filter tag for package level is included, default value is "N".
<searchForTempChange CycledStatus>	Optional	0 -1	String (1)	**Y** = Yes, include TCC packages **N** = No, omit TCC packages ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is "Y". If *any* yes/no filter tag for package status is included, default value is "N".
<searchForUnplannedPerm Type>	Optional	0 -1	String (1)	**Y** = Yes, include unplanned perm **N** = No, omit unplanned perm pkgs ***NOTE:*** Part of ***package type tag group***. If *all* yes/no filter tags for package type are omitted from request, default value is "Y". If *any* yes/no filter tag for package type is included, default value is "N".
<searchForUnplanned TempType>	Optional	0 -1	String (1)	**Y** = Yes, include unplanned temp pkg **N** = No, omit unplanned temp pkgs ***NOTE:*** Part of ***package type tag group***. If *all* yes/no filter tags for package type are omitted from request, default value is "Y". If *any* yes/no filter tag for package type is included, default value is "N".
<searchForXNodeBuild Required>	Optional	0 -1	String (1)	**Y** = include build xnode required pkgs **N** = omit build xnode required pkgs
<searchFromDateApproved>	Optional	0 - 1	Date, yyyymmdd	Starting date in desired range of approval dates for package(s). ***NOTE:*** Returns all packages approved on or after this date if not paired with `<searchToDateApproved>` tag.
<searchFromDateBackedOut>	Optional	0 - 1	Date, yyyymmdd	Starting date in desired range of backout dates for package(s). ***NOTE:*** Returns all packages backed out on or after this date if not paired with `<searchToDateBackedOut>`.

Exhibit 5-1. PACKAGE GENERAL SEARCH <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<searchFromDateBaselined>	Optional	0 - 1	Date, yyyymmdd	Starting date in desired range of baselined dates for package(s). ***NOTE:*** Returns all packages baselined on or after this date if not paired with `<searchToDateBaselined>` tag.
<searchFromDateCreated>	Optional	0 - 1	Date, yyyymmdd	Starting date in desired range of creation dates for package(s). ***NOTE:*** Returns all packages created on or after this date if not paired with `<searchToDateCreated>` tag.
<searchFromDateFrozen>	Optional	0 - 1	Date, yyyymmdd	Starting date in desired range of freeze dates for package(s). ***NOTE:*** Returns all packages frozen on or after this date if not paired with `<searchToDateFrozen>` tag.
<searchFromDateInstalled>	Optional	0 - 1	Date, yyyymmdd	Starting date in desired range of install dates for package(s). ***NOTE:*** Returns all packages installed on or after this date if not paired with `<searchToDateInstalled>` tag. ***NOTE:*** Invalid with complex or super packages.
<searchFromDateRejected>	Optional	0 - 1	Date, yyyymmdd	Starting date in desired range of rejection dates for package(s). ***NOTE:*** Returns all packages rejected on or after this date if not paired with `<searchToDateRejected>` tag.
<searchFromDateReverted>	Optional	0 - 1	Date, yyyymmdd	Starting date in desired range of revert dates for package(s). ***NOTE:*** Returns all packages reverted on or after this date if not paired with `<searchToDateReverted>` tag.
<searchToDateApproved>	Optional	0 - 1	Date, yyyymmdd	Ending date in desired range of approval dates for package(s). ***NOTE:*** Returns all packages approved before or on this date if not paired with `<searchfromDateApproved>` tag.
<searchToDateBackedOut>	Optional	0 - 1	Date, yyyymmdd	Ending date in desired range of backout dates for package(s). ***NOTE:*** Returns all packages backed out before or on this date if not paired with `<searchfromDateBackedOut>`

Exhibit 5-1. PACKAGE GENERAL SEARCH <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<searchToDateBaselined>	Optional	0 - 1	Date, yyyymmdd	Ending date in desired range of baseline dates for package(s). ***NOTE:*** Returns all packages baselined before or on this date if not paired with `<searchfromDateBaselined>` tag.
<searchToDateCreated>	Optional	0 - 1	Date, yyyymmdd	Ending date in desired range of creation dates for package(s). ***NOTE:*** Returns all packages created before or on this date if not paired with `<searchfromDateCreated>` tag.
<searchToDateFrozen>	Optional	0 - 1	Date, yyyymmdd	Ending date in desired range of freeze dates for package(s). ***NOTE:*** Returns all packages frozen before or on this date if not paired with `<searchfromDateFrozen>` tag.
<searchToDateInstalled>	Optional	0 - 1	Date, yyyymmdd	Ending date in desired range of install dates for package(s). ***NOTE:*** Returns all packages installed before or on this date if not paired with `<searchfromDateInstalled>` tag. ***NOTE:*** Invalid with complex or super packages.
<searchToDateRejected>	Optional	0 - 1	Date, yyyymmdd	Ending date in desired range of rejection dates for package(s). ***NOTE:*** Returns all packages rejected before or on this date if not paired with `<searchfromDateRejected>` tag.
<searchToDateReverted>	Optional	0 - 1	Date, yyyymmdd	Ending date in desired range of revert dates for package(s). ***NOTE:*** Returns all packages reverted before or on this date if not paired with `<searchfromDateReverted>` tag.
<siteName>	Optional	0 - 1	String (8), variable	Name of site where desired packages reside.
<sourceLinkIpAddress>	Optional	0 - 1	String (255), variable	String of masked link package URLs.
<sourceLinkPortid>	Optional	0 - 1	String (8), variable	Link package port number.
<stager>	Optional	0 -1	String (255), variable	One or more 8-byte TSO user IDs for package version stagers, delimited by semicolons. ***NOTE:*** Each stager ID in list may be masked by asterisk (*) wildcard.

Exhibit 5-1. PACKAGE GENERAL SEARCH <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<tempChangeDuration>	Optional	0 -1	Integer (3)	Number of days for temporary package to remain in production before automatic deletion.
<workChangeRequest>	Optional	0 -1	String (12), variable	Work order ID or change request number for desired packages.

PACKAGE GENERAL SEARCH Replies

The Serena XML reply message to a general package search request returns zero to many `<result>` tags. Each `<result>` tag contains detailed information about any packages found that satisfied the search criteria in the request.

A standard `<response>` data structure always follows the `<result>` tags, if any, to indicate the overall success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last data element returned in a reply message, the `<response>` tag also serves as an end-of-list indicator.

An example Serena XML reply to a general package search request follows. Data structure details for the `<result>` tag appear after the example in *Exhibit 5-2*.

Example XML — PACKAGE GENERAL SEARCH Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="GENERAL">
  <message name="SEARCH">
   <result>
    <package>ACTP000002</package>
    <applName>ACTP</applName>
    <packageId>000002</packageId>
    <packageLevel>1</packageLevel>
    <packageType>1</packageType>
    <packageStatus>8</packageStatus>
    <requestorName>USER24</requestorName>
    <requestorPhone>5555555</requestorPhone>
    <creator>USER24</creator>
    <tempChangeDuration>000</tempChangeDuration>
    <isStageLibDeleted>N</isStageLibDeleted>
    <isPackageLinked>N</isPackageLinked>
    <isCmnSchedulerUsed>N</isCmnSchedulerUsed>
    <isManualSchedulerUsed>Y</isManualSchedulerUsed>
    <isOtherSchedulerUsed>N</isOtherSchedulerUsed>
    <isAuditPending>N</isAuditPending>
    <workChangeRequest>USER24</workChangeRequest>
    <requestorDept>IDD</requestorDept>
    <dateCreated>20081019</dateCreated>
    <dateInstalled>20081231</dateInstalled>
    <timeInstalled>0200</timeInstalled>
    <dateFrozen>20081112</dateFrozen>
    <dateApproved>20081019</dateApproved>
    <dateBaselined>20081019</dateBaselined>
    <dateBackedOut>20081019</dateBackedOut>
    <dateReverted>20081019</dateReverted>
    <dateDelivered>20081019</dateDelivered>
    <dateReceived>20081019</dateReceived>
    <lastPromotionLevel>00</lastPromotionLevel>
    <isFreezePending>N</isFreezePending>
    <isApprovalPending>N</isApprovalPending>
    <isXNodeBuildRequired>Y</isXNodeBuildRequired>
    <isInstallPending>N</isInstallPending>
    <isRevertPending>N</isRevertPending>
    <isBackoutPending>N</isBackoutPending>
    <isBuildXNodePending>N</isBuildXNodePending>
    <isPostApprovalPending>N</isPostApprovalPending>
    <isPostApproversAdded>N</isPostApproversAdded>
    <isPostRejected>N</isPostRejected>
    <isShortApproverListUsed>N</isShortApproverListUsed>
    <approvalEntity>ACTPLEAD</approvalEntity>
    <packageTitle>TURNOVER 17 IVP NEW CMNIAU00</packageTitle>
   </result>
.
.
.
   <response>
    <statusMessage>CMN8600I - The Package search list is complete.</
statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8600</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 5-2. PACKAGE GENERAL SEARCH <reply> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name for packages to include in statistics. ***NOTE:*** May be masked using asterisk (*) wildcard.
<approvalEntity>	Optional	0 - 1	String (8)	TSO user ID of approval entity.
<auditLockUserid>	Optional	0 - 1	String (8)	TSO user ID that locked package for audit.

Exhibit 5-2. PACKAGE GENERAL SEARCH <reply> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<auditReturnCode>	Optional	0 - 1	String (2), variable	Return code issued by package audit function. Values: **00** = No major errors found **04** = Errors found, fix suggested **08** = Major errors found **12** = Possibly fatal errors found
<complexSuperPackage>	Optional	0 - 1	String (255), variable	ZMF name of complex or super package to which a participating package belongs. NOTE: Returned only if value in `<packageLevel>` = 4.
<creator>	Optional	0 -1	String (8), variable	TSO user IDs for package creators.
<dateApproved>	Optional	0 -1	Date (8), yyyymmdd	Date package approved.
<dateBackedOut>	Optional	0 -1	Date (8), yyyymmdd	Date package backed out.
<dateBaselined>	Optional	0 -1	Date (8), yyyymmdd	Date package baselined.
<dateCreated>	Optional	0 -1	Date (8), yyyymmdd	Date package created.
<dateDelivered>	Optional	0 -1	Date (8), yyyymmdd	Date distributed.
<dateFrozen>	Optional	0 -1	Date (8), yyyymmdd	Date package frozen.
<dateInstalled>	Optional	0 -1	Date (8), yyyymmdd	Date package actually installed. ***NOTE:*** This is the *actual* install date, not the *scheduled* install date. The latter appears in the `<installDate>` tag.
<dateReceived>	Optional	0 -1	Date (8), yyyymmdd	Date package received at remote site.
<dateRejected>	Optional	0 -1	Date (8), yyyymmdd	Date package rejected.
<dateReverted>	Optional	0 -1	Date (8), yyyymmdd	Date package reverted.
<isApprovalPending>	Optional	0 -1	String (1)	**Y** =package approval outstanding **N** = package approval not outstanding
<isAuditPending>	Optional	0 -1	String (1)	**Y** =package audit outstanding **N** = package audit not outstanding

Exhibit 5-2. PACKAGE GENERAL SEARCH <reply> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<isBackoutPending>	Optional	0 -1	String (1)	**Y** =package backout outstanding **N** = package backout not outstanding
<isBuildXNodePending>	Optional	0 -1	String (1)	**Y** = build xnode outstanding **N** = build xnode not outstanding
<isCmnSchedulerUsed>	Optional	0 -1	String (1)	**Y** = CMN scheduling **N** = Non CMN scheduling
<isFreezePending>	Optional	0 -1	String (1)	**Y** = package freeze outstanding **N** = package freeze not outstanding
<isInstallPending>	Optional	0 -1	String (1)	**Y** = package install outstanding **N** = package install not outstanding
<isManualSchedulerUsed>	Optional	0 -1	String (1)	**Y** = Manual scheduling **N** = No manual scheduling
<isOtherSchedulerUsed>	Optional	0 -1	String (1)	**Y** = Other scheduling **N** = No other scheduling
<isPackageLinked>	Optional	0 -1	String (1)	**Y** = Yes, linked to remote pkg **N** = No, not linked to remote pkg
<isPostApprovalPending>	Optional	0 - 1	String (1)	**Y** = post approval pending pkg **N** = non post approval pending pkg
<isPostApproversAdded>	Optional	0 - 1	String (1)	**Y** = Yes, post-approver list added **N** = No, list not added
<isPostRejected>	Optional	0 - 1	String (1)	**Y** = Yes, package post-rejected **N** = No, not post-rejected
<isRevertPending>	Optional	0 - 1	String (1)	**Y** = revert pending pkg **N** = non revert pending pkg
<isShortApproverListUsed>	Optional	0 - 1	String (1)	**Y** = Yes, post-approver list has emergency approvers only **N** = No, not using emergency list of package approvers
<isStageLibDeleted>	Optional	0 -1	String (1)	**Y** = Yes, staging library deleted **N** = No, staging lib not deleted
<isXNodeBuildRequired>	Optional	0 -1	String (1)	**Y** = build xnode required pkg **N** = non build xnode required pkgs
<lastPromoter>	Optional	0 - 1	String (8), variable	TSO user ID of most recent promoter/ demoter.
<lastPromotionLevel>	Optional	0 - 1	String (2), variable	Previous numeric promotion level of package in promotion hierarchy.
<lastPromotionName>	Optional	0 - 1	String (8), variable	Name corresponding to level code in `<lastPromotionLevel>`.

Exhibit 5-2. PACKAGE GENERAL SEARCH <reply> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<lastPromotionSite>	Optional	0 - 1	String (8), variable	Site name where most recent promotion action took place.
<package>	Require d	1	String (10), fixed	Fixed-format ZMF package name.
<packageCheckedIntoRelease>	Optional, ZMF with ERO only	0 - 1	String (1)	**Y** = Yes, checked into release **N** = No, not checked into release
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package number, zero-filled.
<packageLevel>	Optional	0 -1	Integer (1)	Code for package complexity level. Values: **1** = Simple package **2** = Complex package **3** = Super package **4** = Participating package ***NOTE:*** If value = 4, the complex or super package in which this package participates is named in `<complexSuperPackage>`.
<packageStatus>	Optional	0 -1	String (1)	Code for status of package in change lifecycle. Values: **1** = Approved **2** = Backed out **3** = Baselined **4** = Complex/super pkg closed **5** = Deleted (memo delete) **6** = Development **7** = Distributed **8** = Frozen **9** = Installed **A** = Complex/super pkg open **B** = Rejected **C** = Temporary change cycle completed **K** = Release package blocked
<packageTitle>	Optional	0 -1	String (72), variable	Working title of package.
<packageType>	Optional	0 -1	String (1)	Package install type code. Values: **1** = Planned permanent **2** = Planned temporary **3** = Unplanned permanent **4** = Unplanned temporary

Exhibit 5-2. PACKAGE GENERAL SEARCH <reply> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<release>	Optional, ZMF with ERO only	0 -1	String (8)	Name of ERO release to which package is attached or joined.
<releaseArea>	Optional, ZMF with ERO only	0 -1	String (8)	Name of starting release area for release package check-in.
<releaseJoinedDate>	Optional, ZMF with ERO only	0 -1	Date (8), yyyymmdd	Date package joined release.
<releaseJoinedTime>	Optional, ZMF with ERO only	0 -1	Time (8), hhmmss	Time package joined release, 24-hour format.
<requestorDept>	Optional	0 -1	String (4), variable	Workgroup or department code associated with package.
<requestorName>	Optional	0 -1	String (25), variable	Name of developer or contact person responsible for package.
<requestorPhone>	Optional	0 -1	String (15), variable	Phone of developer or contact person responsible for package.
<siteName>	Optional	0 - 1	String (8), variable	Name of development site where package resides.
<tempChangeDuration>	Optional	0 -1	Integer (3)	Number of days for temporary package to remain in production before automatic deletion.
<workChangeRequest>	Optional	0 -1	String (12), variable	Work order ID or change request number for package.

Search for Limbo Packages - PACKAGE LIMBO SEARCH

The Serena XML service/scope/message names for a message to search for limbo packages are:

```
<service name=”PACKAGE”>
<scope name=”LIMBO”>
<message name=”SEARCH”>
```

These tags appear in both request and reply messages.

PACKAGE LIMBO SEARCH — Requests

The `<request>` data structure for a limbo package search request is identical to that for a general package search (*Exhibit 5-1*). The `<scope>` attribute limits the scope of what would otherwise be a general search, so that only packages in limbo appear in the results.

PACKAGE LIMBO SEARCH — Replies

The `<result>` data structure for a limbo package search request is also identical to that for a general package search (*Exhibit 5-2*). The Serena XML reply message to a limbo package search request returns zero to many `<result>` tags, each of which contains information about one limbo package that also satisfied your search criteria. If no such packages are found, no `<result>` tag is returned.

A standard `<response>` data structure always follows the `<result>` tags, if any, to indicate the overall success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last data element returned in a reply message, the `<response>` tag also serves as an end-of-list indicator.

Search for Packages Pending Approval - PACKAGE APPROVE SEARCH

A search for packages pending approval retrieves comprehensive information about all packages in the "pending approval" state for a specified application.

The Serena XML service/scope/message tags and attributes for messages to *search* for packages pending approval are:

```
<service name="PACKAGE">
<scope name="APPROVE">
<message name="SEARCH">
```

These tags appear in both requests and replies.

PACKAGE APPROVE SEARCH Requests

A request to search for pending packages for an application contains one subtag and is defined in *Exhibit 5-3*.

Exhibit 5-3. PACKAGE APPROVE SEARCH <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	1	String (4), variable	ZMF application name.

PACKAGE APPROVE SEARCH Replies

The Serena XML reply message to a Package Approve Search request returns zero to many `<result>` tags. Each `<result>` tag contains detailed information about each package found that is pending approval for the requested application. Data structure details for the `<result>` tag are identical to those for a general package search (see *Exhibit 5-2*).

A standard `<response>` data structure always follows the `<result>` tags, if any, to indicate the overall success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last data element returned in a reply message, the `<response>` tag also serves as an end-of-list indicator.

Search for Linked Packages - PACKAGE PKG_LINK SEARCH

Serena XML provides a means to search for packages on remote LPARs or non-mainframe servers that are linked (via external software) to an explicitly named ChangeMan ZMF package on the local mainframe LPAR.

The Serena XML service/scope/message tags and attributes for messages to *search* for linked packages are:

```
<service name=”PACKAGE”>
<scope name=”PKG_LINK”>
<message name=”SEARCH”>
```

These tags appear in both requests and replies.

PACKAGE PKG_LINK SEARCH Requests

Serena XML supports the following linked package search options:

- ***Semicolon-delimited name lists and/or wildcard patterns*** for package names, linked package names, and IP addresses.
- ***Yes/no flag tag sets*** for desired package level, type, and status.
- ***Standalone yes/no flag tag*** for packages pending approval.
- ***Date range criteria*** for date installed.
- ***Selective package search criteria*** for packages that share a common approval entity or for specific linked package information (such as package name, requestor, IP address, and port ID).

The following example shows how you might code a request to search for all remote, linked packages for a named local package using Serena XML.

Example XML — PACKAGE PKG_LINK SEARCH Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="PKG_LINK">
  <message name="SEARCH">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000037</package>
   </request>
  </message>
 </scope>
</service>
```

Data structure details for the linked package *search* `<request>` tag appear in *Exhibit 5-4*.

Exhibit 5-4. PACKAGE PKG_LINK SEARCH <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<approvalEntity>	Optional	0 - 1	String (8)	TSO user ID of approval entity.
<linkPackage>	Optional	0 - 1	String (255), variable	Name of a linked package on remote server. Package naming conventions are those of remote system. ***NOTE:*** May be masked using asterisk (*) wildcard.
<linkRequestor>	Optional	0 -1	String (20), variable	Name or TSO user ID of package link requestor.
<package>	Required	0 - 1	String (255), variable	One or more ZMF package name(s) of up to 10 bytes each, delimited by semicolons. ***NOTE:*** Each package name in list may be masked using asterisk (*) wildcard.
<searchForApprovalPending>	Optional	0 -1	String (1)	**Y** = include approval pending pkgs **N** = omit approval pending pkgs
<searchForApprovedStatus>	Optional	0 -1	String (1)	**Y** = include approved packages **N** = omit approved packages ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is "Y". If *any* yes/no filter tag for package status is included, default value is "N".
<searchForBackedOutStatus>	Optional	0 -1	String (1)	**Y** = Yes, include backed out pkgs **N** = No, omit backed out pkgs ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is "Y". If *any* yes/no filter tag for package status is included, default value is "N".

Exhibit 5-4. PACKAGE PKG_LINK SEARCH <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<searchForBaselineStatus>	Optional	0 -1	String (1)	**Y** = Yes, include baselined pkgs **N** = No, omit baselined pkgs ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is “Y”. If *any* yes/no filter tag for package status is included, default value is “N”.
<searchForClosedStatus>	Optional	0 -1	String (1)	**Y** = Yes, include closed packages **N** = No, omit closed packages ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is “Y”. If *any* yes/no filter tag for package status is included, default value is “N”.
<searchForComplexLevel>	Optional	0 -1	String (1)	**Y** = Yes, include complex packages **N** = No, omit complex packages ***NOTE:*** Part of ***package level tag group***. If *all* yes/no filter tags for package level are omitted from request, default value is “Y”. If *any* yes/no filter tag for package level is included, default value is “N”.
<searchForDeletedStatus>	Optional	0 -1	String (1)	**Y** = Yes, include deleted packages **N** = No, omit deleted packages ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is “Y”. If *any* yes/no filter tag for package status is included, default value is “N”.
<searchForDeliveredStatus> ***NOTE:*** In the context of this service, “delivered” means “distributed.”	Optional	0 -1	String (1)	**Y** = Yes, include distributed pkgs **N** = No, omit distributed pkgs ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is “Y”. If *any* yes/no filter tag for package status is included, default value is “N”.

Exhibit 5-4. PACKAGE PKG_LINK SEARCH <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<searchForDevelopmentStatus>	Optional	0 -1	String (1)	**Y** = Yes, include development pkgs **N** = No, omit development pkgs ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is "Y". If *any* yes/no filter tag for package status is included, default value is "N".
<searchForFrozenStatus>	Optional	0 -1	String (1)	**Y** = Yes, include frozen packages **N** = No, omit frozen packages ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is "Y". If *any* yes/no filter tag for package status is included, default value is "N".
<searchForInstalledStatus>	Optional	0 -1	String (1)	**Y** = Yes, include installed packages **N** = No, omit installed packages ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is "Y". If *any* yes/no filter tag for package status is included, default value is "N".
<searchForOpenedStatus>	Optional	0 -1	String (1)	**Y** = Yes, include opened packages **N** = No, omit opened packages ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is "Y". If *any* yes/no filter tag for package status is included, default value is "N".
<searchForPartLevel>	Optional	0 -1	String (1)	**Y** = Yes, include participating pkgs **N** = No, omit participating pkgs ***NOTE:*** Part of ***package level tag group***. If *all* yes/no filter tags for package level are omitted from request, default value is "Y". If *any* yes/no filter tag for package level is included, default value is "N".

Exhibit 5-4. PACKAGE PKG_LINK SEARCH <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<searchForPlannedPermType>	Optional	0 -1	String (1)	**Y** = Yes, include planned perm pkgs **N** = No, omit planned perm pkgs ***NOTE:*** Part of ***package type tag group***. If *all* yes/no filter tags for package type are omitted from request, default value is “Y”. If *any* yes/no filter tag for package type is included, default value is “N”.
<searchForPlannedTempType>	Optional	0 -1	String (1)	**Y** = Yes, include planned temp pkgs **N** = No, omit planned temp pkgs ***NOTE:*** Part of ***package type tag group***. If *all* yes/no filter tags for package type are omitted from request, default value is “Y”. If *any* yes/no filter tag for package type is included, default value is “N”.
<searchForRejectedStatus>	Optional	0 -1	String (1)	**Y** = Yes, include rejected packages **N** = No, omit rejected packages ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is “Y”. If *any* yes/no filter tag for package status is included, default value is “N”.
<searchForSimpleLevel>	Optional	0 -1	String (1)	**Y** = Yes, include simple packages **N** = No, omit simple packages ***NOTE:*** Part of ***package level tag group***. If *all* yes/no filter tags for package level are omitted from request, default value is “Y”. If *any* yes/no filter tag for package level is included, default value is “N”.
<searchForSuperLevel>	Optional	0 -1	String (1)	**Y** = Yes, include super packages **N** = No, omit super packages ***NOTE:*** Part of ***package level tag group***. If *all* yes/no filter tags for package level are omitted from request, default value is “Y”. If *any* yes/no filter tag for package level is included, default value is “N”.

Exhibit 5-4. PACKAGE PKG_LINK SEARCH <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<searchForTempChange CycledStatus>	Optional	0 -1	String (1)	**Y** = Yes, include TCC packages **N** = No, omit TCC packages ***NOTE:*** Part of ***package status tag group***. If *all* yes/no filter tags for package status are omitted from request, default value is "Y". If *any* yes/no filter tag for package status is included, default value is "N".
<searchForUnplannedPermType>	Optional	0 -1	String (1)	**Y** = Yes, include unplanned perm **N** = No, omit unplanned perm pkgs ***NOTE:*** Part of ***package type tag group***. If *all* yes/no filter tags for package type are omitted from request, default value is "Y". If *any* yes/no filter tag for package type is included, default value is "N".
<searchForUnplannedTempType>	Optional	0 -1	String (1)	**Y** = Yes, include unplanned temp pkg **N** = No, omit unplanned temp pkgs ***NOTE:*** Part of ***package type tag group***. If *all* yes/no filter tags for package type are omitted from request, default value is "Y". If *any* yes/no filter tag for package type is included, default value is "N".
<searchFromDateInstalled>	Optional	0 - 1	Date, yyyymmdd	Starting date in desired range of install dates for package(s). ***NOTE:*** Returns all packages installed on or after this date if not paired with `<searchToDateInstalled>` tag. ***NOTE:*** Invalid with complex or super packages.
<searchToDateInstalled>	Optional	0 - 1	Date, yyyymmdd	Ending date in desired range of install dates for package(s). ***NOTE:*** Returns all packages installed before or on this date if not paired with `<searchfromDateInstalled>` tag. ***NOTE:*** Invalid with complex or super packages.

Exhibit 5-4. PACKAGE PKG_LINK SEARCH <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<sourceLinkIpAddress>	Optional	0 - 1	String (255), variable	Network IP address for remote server where linked package resides. ***NOTE:*** May be masked using asterisk (*) wildcard. ***NOTE:*** ZMF stores address as provided by external link management software. May contain server name known to that software instead of an IP address.
<sourceLinkPortid>	Optional	0 - 1	String (8), variable	Network port ID for remote server where linked package resides.

PACKAGE PKG_LINK SEARCH — Replies

The linked package search reply contains zero to many `<result>` tags. Each `<result>` contains information about a package on the local LPAR that is linked to at least one remote package with the requested characteristics. Remote package information, such as link package name, requestor, and IP address are included, as they are stored in the package master records for the local package. Information such as package level, type, and status pertain to the local package, not the linked remote package(s).

A standard `<response>` data structure follows the `<result>` tags, if any, to indicate the success or failure of the search request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last data element returned in a Serena XML reply message, the `<response>` tag serves as an end-of-list marker.

An example reply to a linked package search request follows.

Example XML — PACKAGE PKG_LINK SEARCH Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="PKG_LINK">
  <message name="SEARCH">
   <result>
    <package>ACTP000037</package>
    <applName>ACTP</applName>
    <packageId>000037</packageId>
    <packageLevel>1</packageLevel>
    <packageType>1</packageType>
    <packageStatus>6</packageStatus>
    <installDate>20130103</installDate>
    <linkPackage>test-control-item</linkPackage>
    <sourceLinkIpAddress>TestLinkURL</sourceLinkIpAddress>
    <linkDateStamp>20121210</linkDateStamp>
    <linkTimeStamp>121525</linkTimeStamp>
    <linkRequestor>USER01</linkRequestor>
   </result>
   <response>
    <statusMessage>CMN8600I - The Package search list is complete.</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8600</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Data structure details for the linked package search `<result>` tag appear in *Exhibit 5-5*.

Exhibit 5-5. PACKAGE PKG_LINK SEARCH <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of `<package>`.
<installDate>	Optional	0 - 1	Date, yyyymmdd	Planned install date for local package.
<linkControlCodePage>	Optional	0 - 1	String (4), fixed	Link code page.
<linkDateStamp>	Optional	0 - 1	Date, yyyymmdd	Link date.
<linkPackage>	Optional	0 -1	String (255), variable	Name(s) of one or more linked package(s) on remote server, delimited by semicolons. Naming conventions are those of remote system.
<linkRequestor>	Optional	0 - 1	String (20), variable	Name or TSO user ID of package link requestor.
<linkTimeStamp>	Optional	0 - 1	Time, hhmmss	Link time.
<package>	Optional	0 - 1	String (10), fixed	ZMF fixed-format package name for local package.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of `<package>`.
<packageLevel>	Optional	0 - 1	String (1)	Code for package level of local package. Values: **1** = Simple package **2** = Complex package **3** = Super package **4** = Participating package

Exhibit 5-5. PACKAGE PKG_LINK SEARCH <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<packageStatus>	Optional	0 - 1	String (1)	Code for status of local package. Values: **1** = Approved **2** = Backed out **3** = Baselined **4** = Complex/super pkg closed **5** = Deleted (memo delete) **6** = Development **7** = Distributed **8** = Frozen **9** = Installed **A** = Complex/super pkg open **B** = Rejected **C** = Temporary change cycle completed
<packageType>	Optional	0 - 1	String (1)	Code for package install type of local package. Values: **1** = Planned permanent **2** = Planned temporary **3** = Unplanned permanent **4** = Unplanned temporary
<sourceLinkIpAddress>	Optional	0 - 1	String (255), variable	Network IP address for remote server where linked package resides. ***NOTE:*** ZMF stores address as provided by external link management software. May contain server name known to that software instead of an IP address.
<sourceLinkPortid>	Optional	0 - 1	String (8), variable	Network port ID for remote server where linked package resides.

Package Summary Statistics - PACKAGE SERVICE SUMMARY

The Serena XML service/scope/message names for a request to calculate package summary statistics are:

```
<service name=”PACKAGE”>
<scope name=”SERVICE”>
<message name=”SUMMARY”>
```

These tags appear in both request and reply messages.

PACKAGE SERVICE SUMMARY — Requests

A package summary statistics request obtains a count of the packages on the local Changeman ZMF server that conform to your search criteria. Subtotals by package level, package type, package status, and other categories are provided along with a total package count. You can request package counts by application, by release, by requestor department, by planned install date rate, and by a variety of yes/no flag tags.

An example of how you might code a Serena XML request for a package summary statistics report follows. Data structure details for the `<request>` tag appear in *Exhibit 5-6*.

Example XML —PACKAGE SERVICE SUMMARY Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="SERVICE">
  <message name="SUMMARY">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <applName>ACTP</applName>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 5-6. PACKAGE SERVICE SUMMARY <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name for packages to include in statistics. ***NOTE:*** May be masked using asterisk (*) wildcard.
<filterApprovedStatus>	Optional	0 -1	String (1)	**Y** = Yes, include approved pkgs **N** = No, omit approved pkgs ***NOTE:*** Part of ***status tag group***. If no status filter tags appear in request, default value is "Y". If any status filter tag appears in request, default value is "N".
<filterBackedOutStatus>	Optional	0 -1	String (1)	**Y** = Yes, include backed out pkgs **N** = No, omit backed out pkgs ***NOTE:*** Part of ***status tag group***. If no status filter tags appear in request, default value is "Y". If any status filter tag appears in request, default value is "N".
<filterBaselineStatus>	Optional	0 -1	String (1)	**Y** = Yes, include baselined pkgs **N** = No, omit baselined pkgs ***NOTE:*** Part of ***status tag group***. If no status filter tags appear in request, default value is "Y". If any status filter tag appears in request, default value is "N".

Exhibit 5-6. PACKAGE SERVICE SUMMARY <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<filterClosedStatus>	Optional	0 -1	String (1)	**Y** = Yes, include closed packages **N** = No, omit closed packages ***NOTE:*** Part of ***status tag group***. If no status filter tags appear in request, default value is “Y”. If any status filter tag appears in request, default value is “N”.
<filterComplexLevel>	Optional	0 -1	String (1)	**Y** = Yes, include complex packages **N** = No, omit complex packages ***NOTE:*** Part of ***level tag group***. If no level filter tags appear in request, default value is “Y”. If any level filter tag appears in request, default value is “N”.
<filterDeletedStatus>	Optional	0 -1	String (1)	**Y** = Yes, include deleted packages **N** = No, omit deleted packages ***NOTE:*** Part of ***status tag group***. If no status filter tags appear in request, default value is “Y”. If any status filter tag appears in request, default value is “N”.
<filterDeliveredStatus>	Optional	0 -1	String (1)	**Y** = Yes, include delivered pkgs **N** = No, omit delivered pkgs ***NOTE:*** Part of ***status tag group***. If no status filter tags appear in request, default value is “Y”. If any status filter tag appears in request, default value is “N”.
<filterDevelopmentStatus>	Optional	0 -1	String (1)	**Y** = Yes, include development pkgs **N** = No, omit development pkgs ***NOTE:*** Part of ***status tag group***. If no status filter tags appear in request, default value is “Y”. If any status filter tag appears in request, default value is “N”.
<filterFrozenStatus>	Optional	0 -1	String (1)	**Y** = Yes, include frozen packages **N** = No, omit frozen packages ***NOTE:*** Part of ***status tag group***. If no status filter tags appear in request, default value is “Y”. If any status filter tag appears in request, default value is “N”.
<filterInstalledStatus>	Optional	0 -1	String (1)	**Y** = Yes, include installed packages **N** = No, omit installed packages ***NOTE:*** Part of ***status tag group***. If no status filter tags appear in request, default value is “Y”. If any status filter tag appears in request, default value is “N”.

Exhibit 5-6. PACKAGE SERVICE SUMMARY <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<filterOpenedStatus>	Optional	0 -1	String (1)	**Y** = Yes, include opened packages **N** = No, omit opened packages ***NOTE:*** Part of ***status tag group***. If no status filter tags appear in request, default value is "Y". If any status filter tag appears in request, default value is "N".
<filterPartLevel>	Optional	0 -1	String (1)	**Y** = Yes, include participating pkgs **N** = No, omit participating pkgs ***NOTE:*** Part of ***level tag group***. If no level filter tags appear in request, default value is "Y". If any level filter tag appears in request, default value is "N".
<filterPlannedPermType>	Optional	0 -1	String (1)	**Y** = Yes, include planned perm pkgs **N** = No, omit planned perm pkgs ***NOTE:*** Part of ***type tag group***. If no type filter tags appear in request, default value is "Y". If any type filter tag appears in request, default value is "N".
<filterPlannedTempType>	Optional	0 -1	String (1)	**Y** = Yes, include planned temp pkgs **N** = No, omit planned temp pkgs ***NOTE:*** Part of ***type tag group***. If no type filter tags appear in request, default value is "Y". If any type filter tag appears in request, default value is "N".
<filterRejectedStatus>	Optional	0 -1	String (1)	**Y** = Yes, include rejected packages **N** = No, omit rejected packages ***NOTE:*** Part of ***status tag group***. If no status filter tags appear in request, default value is "Y". If any status filter tag appears in request, default value is "N".
<filterSimpleLevel>	Optional	0 -1	String (1)	**Y** = Yes, include simple packages **N** = No, omit simple packages ***NOTE:*** Part of ***level tag group***. If no level filter tags appear in request, default value is "Y". If any level filter tag appears in request, default value is "N".
<filterSuperLevel>	Optional	0 -1	String (1)	**Y** = Yes, include super packages **N** = No, omit super packages ***NOTE:*** Part of ***level tag group***. If no level filter tags appear in request, default value is "Y". If any level filter tag appears in request, default value is "N".

Exhibit 5-6. PACKAGE SERVICE SUMMARY <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<filterTempChangeCycled Status>	Optional	0 -1	String (1)	**Y** = Yes, include TCC packages **N** = No, omit TCC packages ***NOTE:*** Part of ***status tag group***. If no status filter tags appear in request, default value is "Y". If any status filter tag appears in request, default value is "N".
<filterUnplannedPermType>	Optional	0 -1	String (1)	**Y** = Yes, include unplanned perm **N** = No, omit unplanned perm pkgs ***NOTE:*** Part of ***type tag group***. If no type filter tags appear in request, default value is "Y". If any type filter tag appears in request, default value is "N".
<filterUnplannedTempType>	Optional	0 -1	String (1)	**Y** = Yes, include unplanned temp **N** = No, omit unplanned temp pkgs ***NOTE:*** Part of ***type tag group***. If no type filter tags appear in request, default value is "Y". If any type filter tag appears in request, default value is "N".
<fromInstallDate>	Optional	0 -1	Date, yyyymmdd	Start of range for planned install date of packages to include. ***NOTE:*** Does not apply to complex or super packages.
<release>	Optional; ZMF with ERO only	0 - 1	String (8), variable	Name of release with which packages are associated.
<requestorDept>	Optional	0 -1	String (4), variable	Workgroup or department code associated with package.
<toInstallDate>	Optional	0 -1	Date, yyyymmdd	End of range for planned install date of packages to include. ***NOTE:*** Does not apply to complex or super packages.
<workChangeRequest>	Optional	0 -1	String (12), variable	Work order ID or change request number for package.

PACKAGE SERVICE SUMMARY — Replies

A maximum of one `<result>` tag appears in a package summary statistics reply. It contains counts of the packages that meet your search criteria. Subtotals for level, type, and status should each add up to the grand total provided in the `<packageTotalCount>` tag.

Note

If you do not identify a specific requestor department, work change request, install date range, or release name in your request message, package subtotals cannot be calculated for these items. The related `<result>` subtags will be omitted from the reply.

A standard `<response>` data structure follows the `<result>` tag, if any, to indicate the success or failure of the package summary statistics request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

An example Serena XML reply message for a package summary statistics request appears below. Data structure details for the `<result>` tag follow in *Exhibit 5-7*.

Example XML — PACKAGE SERVICE SUMMARY Reply

```
<?xml version="1.0"?>
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="SERVICE">
  <message name="SUMMARY">
   <result>
    <applName>ACTP</applName>
    <packageTotalCount>00007</packageTotalCount>

<totalsByStatus>00000000000000200000000000000300000000020000000000000000000
0</totalsByStatus>
    <totalsByApprovedStatus>00000</totalsByApprovedStatus>
    <totalsByBackedOutStatus>00000</totalsByBackedOutStatus>
    <totalsByBaselineStatus>00002</totalsByBaselineStatus>
    <totalsByClosedStatus>00000</totalsByClosedStatus>
    <totalsByDeletedStatus>00000</totalsByDeletedStatus>
    <totalsByDevelopmentStatus>00003</totalsByDevelopmentStatus>
    <totalsByDeliveredStatus>00000</totalsByDeliveredStatus>
    <totalsByFrozenStatus>00002</totalsByFrozenStatus>
    <totalsByInstalledStatus>00000</totalsByInstalledStatus>
    <totalsByOpenedStatus>00000</totalsByOpenedStatus>
    <totalsByRejectedStatus>00000</totalsByRejectedStatus>
    <totalsByTempChangeCycledStatus>00000</totalsByTempChangeCycledStatus>
    <totalsByLevel>00007000000000000000</totalsByLevel>
    <totalsBySimpleLevel>00007</totalsBySimpleLevel>
    <totalsByComplexLevel>00000</totalsByComplexLevel>
    <totalsBySuperLevel>00000</totalsBySuperLevel>
    <totalsByPartLevel>00000</totalsByPartLevel>
    <totalsByType>00007000000000000000</totalsByType>
    <totalsByPlannedPermType>00007</totalsByPlannedPermType>
    <totalsByPlannedTempType>00000</totalsByPlannedTempType>
    <totalsByUnplannedPermType>00000</totalsByUnplannedPermType>
    <totalsByUnplannedTempType>00000</totalsByUnplannedTempType>
    <releaseTotals>00000</releaseTotals>
    <totalsByInstallDate>00002</totalsByInstallDate>
.
.
.
   </service>
```

Exhibit 5-7. PACKAGE SERVICE SUMMARY <reply> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name for packages summarized.
<packageTotalCount>	Optional	1	Integer (5), variable	Count of all packages that meet search criteria.
<releaseTotals>	Optional	0 - 1	Integer (5), variable	Count of packages in named release that also meet other search criteria. ***NOTE:*** Value returned only if release specified in XML request.
<totalsByApprovedStatus>	Optional	0 -1	Integer (5), variable	Count of approved packages that meet search criteria. ***NOTE:*** Part of ***status tag group***. Sum of status tag subtotals should equal `<packageTotalCount>`.
<totalsByBackedOutStatus>	Optional	0 -1	Integer (5), variable	Count of backed out packages that meet search criteria. ***NOTE:*** Part of ***status tag group***. Sum of status tag subtotals should equal `<packageTotalCount>`.
<totalsByBaselineStatus>	Optional	0 -1	Integer (5), variable	Count of baselined packages that meet search criteria. ***NOTE:*** Part of ***status tag group***. Sum of status tag subtotals should equal `<packageTotalCount>`.
<totalsByClosedStatus>	Optional	0 -1	Integer (5), variable	Count of closed packages that meet search criteria. ***NOTE:*** Part of ***status tag group***. Sum of status tag subtotals should equal `<packageTotalCount>`.
<totalsByComplexLevel>	Optional	0 -1	Integer (5), variable	Count of complex packages that meet search criteria. ***NOTE:*** Part of ***level tag group***. Sum of level tag subtotals should equal `<packageTotalCount>`.
<totalsByDeletedStatus>	Optional	0 -1	Integer (5), variable	Count of deleted packages that meet search criteria. ***NOTE:*** Part of ***status tag group***. Sum of status tag subtotals should equal `<packageTotalCount>`.

Exhibit 5-7. PACKAGE SERVICE SUMMARY <reply> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<totalsByDeliveredStatus>	Optional	0 -1	Integer (5), variable	Count of delivered packages that meet search criteria. ***NOTE:*** Part of ***status tag group***. Sum of status tag subtotals should equal `<packageTotalCount>`.
<totalsByDept>	Optional	0 -1	Integer (5), variable	Count of packages for named department that also meet other search criteria. ***NOTE:*** Returned only if department specified in XML request.
<totalsByDevelopment Status>	Optional	0 -1	Integer (5), variable	Count of packages in development that meet search criteria. ***NOTE:*** Part of ***status tag group***. Sum of status tag subtotals should equal `<packageTotalCount>`.
<totalsByFrozenStatus>	Optional	0 -1	Integer (5), variable	Count of frozen packages that meet search criteria. ***NOTE:*** Part of ***status tag group***. Sum of status tag subtotals should equal `<packageTotalCount>`.
<totalsByInstallDate>	Optional	0 -1	Integer (5), variable	Count of packages with planned install dates within named date range and that also meet other search criteria. ***NOTE:*** Returned only if install date range included in XML request.
<totalsByInstalledStatus>	Optional	0 -1	Integer (5), variable	Count of installed packages that meet search criteria. ***NOTE:*** Part of ***status tag group***. Sum of status tag subtotals should equal `<packageTotalCount>`.
<totalsByLevel>	Optional	0 -1	Integer (5), variable	Count of simple packages that meet search criteria. ***NOTE:*** Part of ***level tag group***. Sum of level tag subtotals should equal `<packageTotalCount>`.
<totalsByOpenedStatus>	Optional	0 -1	Integer (5), variable	Count of opened packages that meet search criteria. ***NOTE:*** Part of ***status tag group***. Sum of status tag subtotals should equal `<packageTotalCount>`.

Exhibit 5-7. PACKAGE SERVICE SUMMARY <reply> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<totalsByPartLevel>	Optional	0 -1	Integer (5), variable	Count of participating packages that meet search criteria. ***NOTE:*** Part of ***level tag group***. Sum of level tag subtotals should equal `<packageTotalCount>`.
<totalsByPlannedPerm Type>	Optional	0 -1	Integer (5), variable	Count of planned permanent packages that meet search criteria. ***NOTE:*** Part of ***type tag group***. Sum of type tag subtotals should equal `<packageTotalCount>`.
<totalsByPlannedTemp Type>	Optional	0 -1	Integer (5), variable	Count of planned temporary packages that meet search criteria. ***NOTE:*** Part of ***type tag group***. Sum of type tag subtotals should equal `<packageTotalCount>`.
<totalsByReasonCode>	Optional	0 -1	Integer (5), variable	Count of packages that meet search criteria.
<totalsByRejectedStatus>	Optional	0 -1	Integer (5), variable	Count of rejected packages that meet search criteria. ***NOTE:*** Part of ***status tag group***. Sum of status tag subtotals should equal `<packageTotalCount>`.
<totalsBySimpleLevel>	Optional	0 -1	Integer (5), variable	Count of simple packages that meet search criteria. ***NOTE:*** Part of ***level tag group***. Sum of level tag subtotals should equal `<packageTotalCount>`.
<totalsByStatus>	Optional	0 -1	Integer (5), variable	Totals by status.
<totalsBySuperLevel>	Optional	0 -1	Integer (5), variable	Count of super packages that meet search criteria. ***NOTE:*** Part of ***level tag group***. Sum of level tag subtotals should equal `<packageTotalCount>`.
<totalsByTempChange CycledStatus>	Optional	0 -1	Integer (5), variable	Count of temporary packages that have cycled past their automatic uninstall date & meet search criteria. ***NOTE:*** Part of ***status tag group***. Sum of status tag subtotals should equal `<packageTotalCount>`.
<totalsByType>	Optional	0 -1	Integer (5), variable	Totals by type.

Exhibit 5-7. PACKAGE SERVICE SUMMARY <reply> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<totalsByUnplannedPerm Type>	Optional	0 -1	Integer (5), variable	Count of unplanned permanent packages that meet search criteria. ***NOTE:*** Part of ***type tag group***. Sum of type tag subtotals should equal `<packageTotalCount>`.
<totalsByUnplannedTemp Type>	Optional	0 -1	Integer (5), variable	Count of unplanned temporary packages that meet search criteria. ***NOTE:*** Part of ***type tag group***. Sum of type tag subtotals should equal `<packageTotalCount>`.
<totalsByWorkChange Request>	Optional	0 -1	Integer (5), variable	Count of packages for specified work change request that also meet other search criteria. ***NOTE:*** Returned only if work change request number specified in XML request.

AUDIT TRAIL MANAGEMENT

Work with activity log file entries is supported by Serena XML. User tasks include:

- *Create Log File Entry - LOG SERVICE CREATE*
- *List Activity Log File Entries - LOG SERVICE LIST*

Create Log File Entry - LOG SERVICE CREATE

This service posts a date-and-time-stamped entry to the activity log file in order to maintain an audit trail of change activities. All activity log entry types are included in the scope of this function.

The Serena XML service/scope/message names for a request to create a log file entry are:

```
<service name="LOG">
<scope name="SERVICE">
<message name="CREATE">
```

These tags appear in both request and reply messages.

LOG SERVICE CREATE — Requests

This functions accepts a free-format text entry for the activity log file of the type identified in the `<logType>` tag. Data structure details for the `<request>` tag appear in *Exhibit 5-8*.

Exhibit 5-8. LOG SERVICE CREATE<request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<logPackage>	Required	1	String (10), variable	Package/Application.
<logText>	Required	1	String (54), variable	Free-format log file entry text.

Exhibit 5-8. LOG SERVICE CREATE<request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<logType>	Required	1	String (2), variable	Log activity type. 01 Backout a Package 02 Install a Package 03 Temporary Change Cycle 04 Distribute a Package 05 Unauthorized Member Access 07 Generate Package Information 08 Delete a Package 09 Generate Application Information 10 Revert a Package 11 Generate Global Information 12 Activate a Component 13 Package Memo Delete 14 Undelete a Package 15 Baseline Ripple 16 Baseline Reverse Ripple 18 Install Package Aged 20 Approve a Package 21 Calendar Re-Synch 22 Staging Libraries Aged 23 Backout A Release 24 Install A Release 25 Distribute a Release 26 Delete a Release 27 Revert a Release 28 Approve a Release 29 Reject a Release 30 Reject a Package 31 Memo Delete a Release 32 Undelete a Release 33 Baseline a Release 34 Install Release Aged 35 Block a Release 36 Unblock a Release 37 Create/Update a Release 40 Freeze a Package

Exhibit 5-8. LOG SERVICE CREATE<request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
	Required	1	String (2), variable	More Log activity types. 42 Selectively Unfreeze a Package 43 Demote a Component 44 Demote a Package 45 Promote a Release Area 46 Demote a Release Area 48 Promote a Package 49 Promote a Component 50 Audit a Package 51 Alter audit return code 52 Audit a Release Area 53 Approve a Release Area 54 Reject a Release Area 55 Block a Release Area 56 Unblock a Release Area 60 Link a Package 62 Unlink a Package 64 Scratch a Component 66 Rename a Component 68 Component Copied 70 File Tailoring Started 71 File Tailoring Failed 72 File Tailoring Completed 80 Create a Package 81 Checkin Component to Release 82 Checkout a Component 83 Potential Checkout Conflict 84 Stage a Component 85 Overlay Previous Module 86 Delete Module From Package 87 Checkout Component from Release 88 Copy Forward a Package 89 Retrieve Component from Release 90 Monitor Limbo, Internal Scheduler 91 Update Global Release Approvers 92 Update Release Definitions 93 Update Release Applications 94 Attach a Package to a Release 95 Detach a Package from a Release

LOG SERVICE CREATE — Replies

No `<result>` data structure is returned in the create log file entry reply message. However, the standard `<response>` data structure is returned to indicate the success or failure of the checkout request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

List Activity Log File Entries - LOG SERVICE LIST

This service requests a list of activity log file entries by activity type, date range, or both. This retrieves an audit trail of change activities. All activity log entry types are included in the scope of this function.

The Serena XML service/scope/message names for a request to list log file entries are:

```
<service name="LOG">
<scope name="SERVICE">
<message name="LIST">
```

These tags appear in both request and reply messages.

LOG SERVICE LIST — Requests

Request messages should include either a log file entry type or a date range; otherwise, all log file entries of all types and dates will be returned. If a date range is used, only a starting date is required.

Data structure details appear in *Exhibit 5-9*.

Exhibit 5-9. LOG SERVICE LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<fromDateStamp>	Optional	0 - 1	Date, yyyymmdd	Starting date of log entries desired. ***NOTE:*** Need not be paired with ending date if all entries from start date to present are desired.
<fromTimeStamp>	Optional	0 - 1	Time, HHmmssthtt	Starting time of log entries desired.
<logPackage>	Optional	1	String (10), variable	Package/Application.
<logTypes>	Optional	0 - 1	String (150), variable	Semicolon-delimited list of 2 byte activity type codes desired. Type codes are detailed in*Exhibit 5-8*
<toDateStamp>	Optional	0 - 1	Date, yyyymmdd	Ending date of log entries desired. ***NOTE:*** If used, `<fromDateStamp>` also required.

Exhibit 5-9. LOG SERVICE LIST <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<toTimeStamp>	Optional	0 - 1	Time, HHmmssthtt	Ending time of log entries desired.
<user>	Optional	0 - 1	String (8), variable	TSO user ID of desired log entries.

LOG SERVICE LIST — Replies

This function returns zero to many <result> tags. Each <result> contains a single date-and-time-stamped log file entry.

A standard `<response>` data structure follows the final `<result>` tag, if any, to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Data structure details for the <result> tag appear in *Exhibit 5-10*.

Exhibit 5-10. List Activity Log File Entries <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<logDate>	Optional	0 - 1	Date, yyyymmdd	Date of activity log entry.
<logPackage>	Optional	1	String (10), variable	Package/Application.
<logText>	Optional	0 - 1	String (64), variable	Free-format log file entry text.
<logTime>	Optional	0 - 1	Time, hhmmss	Time of activity log entry in 24-hour format.
<logType>	Optional	0 - 1	String (2), variable	Code for type of activity log entry.
<user>	Optional	0 - 1	String (8), variable	TSO user ID associated with entry.

IMPACT ANALYSIS FUNCTIONS

The XML Services CMPONENT/IMP_ANAL/LIST and CMPONENT/XAP_ANAL/LIST have been retired. New services exist as follows:

- *IMPACT BUN LIST*
- *IMPACT CMPONENT LIST*
- *IMPACT TABLE LIST*

IMPACT BUN LIST

This service is used to list information about appl/libtype/baseline correlations. There are three types of queries:

- BUN01: list entry for a specific BUN (or all BUNs if the BUN input tag is omitted)
- BUN02: list entry for a specific application and library type
- BUN03: list all entries for a specific baseline dataset name

IMPACT BUN LIST — Request

The following request supplies only the query type, BUN01. This will produce a list of all the BUNs with their corresponding data set names and library types.

Example XML — IMPACT BUN LIST Request

```
<?xml version="1.0"?>
<service name="IMPACT">
 <scope name="BUN">
  <message name="LIST">
   <header>
     <subsys>3</subsys>
     <product>CMN</product>
   </header>
  <request>
   <queryType>BUN01</queryType>
  </request>
  </message>
 </scope>
</service>
```

Data structure details appear in *Exhibit 5-11*.

Exhibit 5-11IMPACT BUN LIST<request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<appl>	Optional	0 - 1	String (4), variable	Application name.
<baseline>	Optional	0 - 1	String (44), variable	Baseline repository.
<bun>	Optional	0 - 1	String (8), variable	Baseline Unique Number.
<libLike>	Optional	0 - 1	String (1)	Library like type.
<libType>	Optional	0 - 1	String (3), variable	Library type.
<queryType>	Required	1 - 1	String (5)	Query type.

IMPACT BUN LIST — Reply

Example XML — IMPACT BUN LIST Reply

```
<?xml version="1.0"?>
<service name="IMPACT">
 <scope name="BUN">
  <message name="LIST">
   <result>
    <bun>00025000</bun>
    <appl>ACTP</appl>
    <libType>CPS</libType>
    <libLike>C</libLike>
    <baseline>CMNTP.SERT3.BASE.ACTP.CPS</baseline>
   </result>
   <result>
    <bun>0002501B</bun>
    <appl>ACTP</appl>
    <libType>CPY</libType>
    <libLike>C</libLike>
    <baseline>CMNTP.SERT3.BASE.ACTP.CPY</baseline>
   </result>
   <result>
    <bun>00025036</bun>
    <appl>ACTP</appl>
    <libType>DBR</libType>
    <libLike>P</libLike>
    <baseline>CMNTP.SERT3.BASE.ACTP.DBR</baseline>
   </result>
   <result>
     <bun>00025051</bun>
     <appl>ACTP</appl>
     <libType>LCT</libType>
     <libLike>K</libLike>
     <baseline>CMNTP.SERT3.BASE.ACTP.LCT</baseline>
    </result>
.
.
   <response>
     <statusMessage>CMN8700I - LIST service completed</statusMessage>
     <statusReturnCode>00</statusReturnCode>
     <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Data structure details appear in *Exhibit 5-12*.

Exhibit 5-12IIMPACT BUN LIST<reply> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<appl>	Optional	0 - 1	String (4), variable	Application name.
<baseline>	Optional	0 - 1	String (44), variable	Baseline repository.
<bun>	Optional	0 - 1	String (8), variable	Baseline Unique Number.
<libLike>	Optional	0 - 1	String (1)	Library like type.
<libType>	Optional	0 - 1	String (3), variable	Library type.

IMPACT CMPONENT LIST

This service can be used to list information held in the I/A table about baselined components. This will be mostly used to extract version identifier information (hash tokens, setssi values).

IMPACT CMPONENT LIST — Request

The following requests information for SRS component GNLSRS00 in the ACTP application.

Example XML — IMPACT CMPONENT LIST Request

```
<?xml version="1.0"?>
<service name="IMPACT">
 <scope name="CMPONENT">
  <message name="LIST">
   <header>
    <subsys>3</subsys>
    <product>CMN</product>
   </header>
  <request>
    <appl>ACTP</appl>
    <libType>SRS</libType>
    <component>GNLSRS00</component>
   </request>
  </message>
 </scope>
</service>
```

Data structure details appear in *Exhibit 5-13*.

Exhibit 5-13IMPACT CMPONENT LIST<request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<appl>	Required	1 - 1	String (4), variable	Application name.
<bun>	Optional	0 - 1	String (8), variable	Baseline Unique Number.
<component>	Required	1 - 1	String(256), variable	ZMF component name. • If component is PDS member, this is member name (max 8 bytes, no qualifi-ers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<currentBaseline>	Optional	0 - 1	String (44), variable	Current baseline? (Y/N)
<libType>	Required	1 - 1	String (3), variable	Library type.

IMPACT CMPONENT LIST — Reply

Example XML — IMPACT CMPONENT LIST Reply

```
<?xml version="1.0"?>
<service name="IMPACT">
 <scope name="CMPONENT">
  <message name="LIST">
   <result>
    <bun>000250D8</bun>
    <appl>ACTP</appl>
    <libType>SRS</libType>
    <component>GNLSRS00</component>
    <currentBaseline>Y</currentBaseline>
    <versionId>C8977307000002A3</versionId>
   </result>
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Data structure details appear in *Exhibit 5-14*.

Exhibit 5-14IMPACT CMPONENT LIST<reply> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<appl>	Required	1 - 1	String (4), variable	Application name.
<bun>	Optional	0 - 1	String (8), variable	Baseline Unique Number.
<component>	Required	1 - 1	String(256), variable	Component name. • If component is PDS member, this is member name (max 8 bytes, no qualifiers). • If component is HFS file, this is Unix-style long file name, optionally prefixed by path from installation root.
<currentBaseline>	Required	0 - 1	String (44), variable	Current baseline.
<libType>	Required	1 - 1	String (3), variable	Library type.
<versionid>	Required	1 - 1	String (16), variable	Version Identifier.

IMPACT TABLE LIST

Use this service to access impact analysis relationship information. Several types of query are available within this service and the user will have to know which one they wish to use. Current types are:

- IAQ01: all rows for a top level component name
- IAQ02: all rows for a top level component name and BUN
- IAQ03: all rows for a top level component name and BUN and relationship
- IAQ04: all rows for bottom level component name and relationship

There are further XAPnn query types where knowledge of the BUN is not required. In the descriptions that follow only the component names may be wild carded. All other specified predicates must be fully explicit. If a field is not listed in the description of the query then it plays no part in selection. A number of these queries will overlap in function if a name is fully wildcarded. To use these queries you need to decide which one you want to use. The ISPF dialog functions Q.B and Q.I will use the appropriate query based on the fields provided in the input panel.

- XAP00:all rows for top component name, relationship and bottom component name
- XAP01:all rows for top component name, top application, top libtype, relation and bottom component name

- XAP02:all rows for top component name, relation and bottom component name, bottom appl, bottom libtype
- XAP03:all rows for top component name, top application, top libtype, relation and bottom component name, bottom appl, bottom libtype
- XAP04:all rows for top component name, top libtype, relation and bottom component name
- XAP05:all rows for top component name, relation and bottom component name, bottom libtype
- XAP06:all rows for top component name, top libtype, relation and bottom component name, bottom libtype
- XAP07:all rows for top component name, and relation
- XAP08:all rows for top component name, top application, top libtype and relation
- XAP09:all rows for top component name, top libtype and relation
- XAP14:all rows for top component name, top libtype, relation and bottom component name, bottom appl, bottom libtype
- XAP15:all rows for top component name, top application, top libtype, relation and bottom component name, bottom libtype

IMPACT TABLE LIST — Request

The following requests information for CPY component GNLCPY00.

Example XML — IMPACT TABLE LIST Request

```
<?xml version="1.0"?>
<service name="IMPACT">
 <scope name="TABLE">
  <message name="LIST">
   <header>
    <subsys>3</subsys>
    <product>CMN</product>
   </header>
  <request>
    <queryType>XAP00</queryType>
    <topLevelComponent>*</topLevelComponent>
    <topLevelAppl>*</topLevelAppl>
    <topLevelType>*</topLevelType>
    <relation>CPY</relation>
    <bottomLevelComponent>GNLCPY00</bottomLevelComponent>
   </request>
  </message>
 </scope>
</service>
```

Data structure details appear in *Exhibit 5-13*.

Exhibit 5-15IMPACT TABLE LIST<request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<bottomLevelAppl>	Required	1 - 1	String (4), variable	Bottom level component application name.
<bottomLevelBUN>	Optional	0 - 1	String (8), variable	Bottom level BUN.
<bottomLevelComponent >	Optional	1 - 1	String(16), variable	Bottom level Component name.
<bottomLevelType>	Optional	0 - 1	String (4), variable	Bottom level library type
<bottomVerID>	Optional	0 - 1	String (16), variable	Bottom level version ID.
<queryType>	Required	1 - 1	String (5)	Query type.
<relation>	Optional	1 - 1	String (3)	Relationship.
<topLevelAppl>	Optional	1 - 1	String (4),variable.	Top level application name.
<topLevelBUN>	Optional	1 - 1	String (8),variable.	Top level BUN.
<topLevelComponent>	Optional	1 - 1	String (256), variable.	Top level component name.
<topLevelType>	Optional	1 - 1	String (3)	Top level library type.

IMPACT TABLE LIST — Reply

The following requests information for CPY component GNLCPY00.

Example XML — IMPACT TABLE LIST Reply

```
<?xml version="1.0"?>
<service name="IMPACT">
 <scope name="TABLE">
  <message name="LIST">
   <result>
    <topLevelComponent>ACPSRC1A</topLevelComponent>
    <topLevelAppl>ACTP</topLevelAppl>
    <topLevelType>SRC</topLevelType>
    <topLevelBUN>000250BD</topLevelBUN>
    <topVerID>0B5DEB4D000004BA</topVerID>
    <relation>CPY</relation>
    <bottomLevelComponent>GNLCPY00</bottomLevelComponent>
    <bottomLevelAppl>ACTP</bottomLevelAppl>
    <bottomLevelType>CPY</bottomLevelType>
    <bottomLevelBUN>0002501B</bottomLevelBUN>
    <bottomVerID>0000000000000000</bottomVerID>
```

```
      </result>
      <result>
       <topLevelComponent>ACPSRC1A</topLevelComponent>
       <topLevelAppl>ACTP</topLevelAppl>
       <topLevelType>SRS</topLevelType>
       <topLevelBUN>000250D8</topLevelBUN>
       <topVerID>0B5DEB4D000004BA</topVerID>
       <relation>CPY</relation>
       <bottomLevelComponent>GNLCPY00</bottomLevelComponent>
       <bottomLevelAppl>ACTP</bottomLevelAppl>
       <bottomLevelType>CPY</bottomLevelType>
       <bottomLevelBUN>0002501B</bottomLevelBUN>
       <bottomVerID>7CD43F6F00000155</bottomVerID>
      </result>
   .
   .
   .
      <result>
       <topLevelComponent>GNLSRS5C</topLevelComponent>
       <topLevelAppl>GENL</topLevelAppl>
       <topLevelType>SRS</topLevelType>
       <topLevelBUN>000251F2</topLevelBUN>
       <topVerID>414E1E1300000244</topVerID>
       <relation>CPY</relation>
       <bottomLevelComponent>GNLCPY00</bottomLevelComponent>
       <bottomLevelAppl>GENL</bottomLevelAppl>
       <bottomLevelType>CPY</bottomLevelType>
       <bottomLevelBUN>00025135</bottomLevelBUN>
       <bottomVerID>7CD43F6F00000155</bottomVerID>
      </result>
      <response>
       <statusMessage>CMN8700I - LIST service completed</statusMessage>
       <statusReturnCode>00</statusReturnCode>
       <statusReasonCode>8700</statusReasonCode>
      </response>
     </message>
    </scope>
   </service>                                       <?xml
```

Data structure details appear in *Exhibit 5-16*.

Exhibit 5-16IMPACT TABLE LIST<reply> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<bottomLevelAppl>	Optional	1 - 1	String (4), variable	Bottom level component application name.
<bottomLevelBUN>	Optional	0 - 1	String (8), variable	Bottom level BUN.
<bottomLevelComponent>	Optional	1 - 1	String(16), variable	Bottom level Component name.

Exhibit 5-16IMPACT TABLE LIST<reply> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<bottomLevelType>	Optional	0 - 1	String (4), variable	Bottom level library type
<bottomVerID>	Optional	0 - 1	String (16), variable	Bottom level version ID.
<queryType>	Optional	1 - 1	String (5)	Query type.
<relation>	Optional	1 - 1	String (3)	Relationship.
<topLevelAppl>	Optional	1 - 1	String (4),variable.	Top level application name.
<topLevelBUN>	Optional	1 - 1	String (8),variable.	Top level BUN.
<topLevelComponent>	Optional	1 - 1	String (256), variable.	Top level component name.
<topLevelType>	Optional	1 - 1	String (3)	Top level library type.
<topVerID>	Optional	1 - 1	String (16), variable.	Top level version ID.

The ISPF I/A query function is largely unchanged from the DB2 license version currently in force. It is sometimes difficult to comprehend the difference between the "Bill of Materials" query (Q.B) and the "Impact Analysis" query. Q.B is intended for a top down search, i.e. I know the name of the top level component I want enquire about, now show me its constituent parts (Bill of Materials). The Q.I function is focussed on a bottom up search, i.e. given this particular subcomponent show me all components that reference it. This approach has not changed but will be new for all current non-DB2 license customers.

There have been some minor changes to the functionality. We no longer accept partially wildcarded applications or library types in these queries, they can be fully explicit or omitted (or fully wildcarded).

We no longer attempt to convert applications and library types to BUNs and use those for lookup, we will use the appl/libtype as supplied by the customer (as is) and match against those recorded in the i/a table. If a customer wishes to see all relationships for a shared baseline (in a Q.I query) then they need to leave application and/or library type as fully wildcarded. As a corollary of this, we also no longer display the full list of appl/libtypes which share the same baseline as the component in question.

The preceding XML IMPACT TABLE LIST request is the same query issued by option Q.I from the ISPF interface, as shown below.

Example ISPF Request — Q.I

```
------------------ IMPACT ANALYSIS OF SUBORDINATE COMPONENTS
COMMAND ===>

SPECIFY SEARCH CRITERIA:

Subordinate component name ===> GNLCPY00
              Library type ===>
               Application ===>

      Type of relationship ===> C


SPECIFY RESULTS FILTER CRITERIA:

   Superior component name ===> *
              Library type ===> *
               Application ===> *


Press ENTER to process; Enter END command to exit.
```

Example ISPF Reply — Q.I

```
-------------------- Impact Analysis Results Selection List  Row 1 to 11
Command ===>                                                  Scroll ===

 List of components which reference
   Appl:Type  *    : *    Name  GNLCPY00
 with a relationship of         COPYBOOK
 which satisfy these criteria
   Appl:Type  *    : *    Name  *

  Found in
  Appl:Lib  Component Name
_ ACTP:SRC  ACPSRC1A
_ ACTP:SRS  ACPSRC1A
_ GENL:SRC  GNLSRC1A
_ GENL:SRS  GNLSRC1A
_ ACTP:SRS  GNLSRS00
_ GENL:SRS  GNLSRS00
_ GENL:SRS  GNLSRS1B
_ GENL:SRS  GNLSRS1C
_ GENL:SRS  GNLSRS5A
_ GENL:SRS  GNLSRS5B
_ GENL:SRS  GNLSRS5C
******************************* Bottom of data *************************
```

DATASET MANAGEMENT 6

Serena XML lets you work directly with z/OS datasets and members on the mainframe. Partitioned datasets (PDS and PDS/E), ISPF files, and baseline members in stacked reverse delta (SRD) format are supported. Tasks are provided to create, delete, or list a dataset, dataset member, or dataset directory on the host system.

The syntax that identifies a dataset management message in Serena XML appears in the name attribute of the <service> tag, as follows:

```
<service name=”DSS”>
```

z/OS Unix Hierarchical File System (HFS) files are not supported by these services. DB2 database tables and IMS database files are likewise not supported.

Services to manage z/OS Unix Hierarchical File System (HFS) files are discussed in *Chapter 7, “Hierarchical File System Services,” on page 379*.

Database files and tables managed under IMS or DB2 are discussed in *Chapter 8, “Database Management,” on page 405*.

DATASET LIFECYCLE TASKS

Serena XML supports the following dataset lifecycle tasks for general use:

- *Allocate a Dataset - DSS SERVICE ALLOCATE*
- *Delete a Dataset - DSS SERVICE DELETE*
- *Delete a Dataset Member - DSS SERVICE MBRDEL*
- *List Dataset Allocation Information - DSS SERVICE INFO*
- *List Dataset Member Directory - DSS SERVICE LIST*
- *List ISPF Dataset Allocation Information - DSS ISPFILE INFO*
- *List Statistics for Baseline Members - DSS SERVICE BASESTAT*
- *Expand Member in SRD Format - DSS SERVICE EXPAND*

Allocate a Dataset - DSS SERVICE ALLOCATE

This function allocates an empty dataset on the host. Access permissions for the resources requested must first be defined for you in your mainframe security system.

The Serena XML service/scope/message names for a message to *allocate* a host dataset are:

```
<service name="DSS">
<scope name="SERVICE">
<message name="ALLOCATE">
```

These tags appear in both requests and replies.

DSS SERVICE ALLOCATE Requests

Data structure details for the `<request>` tag appear in *Exhibit 6-1*.

Exhibit 6-1. DSS SERVICE ALLOCATE <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<blockSize>	Required	1	Integer (6), variable	Block size in bytes.

<dataOrg>	Required	1	String (8), variable	Data set organization. Values:
				DAM = Direct Access Method
				ISAM = Indexed Sequential Access Method
				MIG = Migrate
				NULL = Null File
				PDS = Partitioned Data Set
				PDSE = Partitioned Data Set, Extended
				PS = Physical Sequential
				SEQ = Sequential File
				VSAM = Virtual Sequential Access Method

<dirBlocks>	Optional	0 - 1	Integer (6), variable	Directory allocation for data set in blocks. Required if allocating a PDS
<eAttr>	Optional	0 - 1	String (1)	Extended attribute option. Values: **N** = Dataset cannot have extended attributes or reside in EAS. **O** = Dataset can have extended attributes and reside in EAS. **blank** = Use default based on data type.
<mvsLib>	Required	1	String (255), variable	Fully qualified dataset name for dataset to be allocated.
<primarySpace>	Required	1	String (8), variable	DASD space allocation for primary dataset region, in units specified by `<spaceType>`.

Exhibit 6-1. DSS SERVICE ALLOCATE <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies

<recordFormat>	Required	1	String(3), variable	Code for physical data set record format. Values:
				F = Fixed
				FA = Fixed ASA
				FM = Fixed Machine
				FB = Fixed Block
				FBA = Fixed Block ASA
				FBM = Fixed Block Machine
				FBS = Fixed Block Standard
				FS = Fixed Standard
				FSA = Fixed Standard ASA
				FSM = Fixed Standard Machine
				V = Variable
				VA = Variable ASA
				VM = Variable Machine
				VB = Variable Block
				VBA = Variable Block ASA
				VBM = Variable Block Machine
				VS = Variable Spanned
				VSA = Variable Spanned ASA
				VSM =Variable Spanned Machine
				U = Undefined
				UA = Undefined ASA
				UM = Undefined Machine
				UB = Undefined Block
				UBA = Undefined Block ASA
				UBM = Undefined Block Machine
				US = Undefined Spanned
				USA = Undefined Spanned ASA
				USM =Undefined Spanned Machine

<recordLength>	Required	1	Integer (6), variable	Record length in bytes.
<secondarySpace>	Required	1	String (8), variable	DASD space allocation for dataset extents, in units specified by `<spaceType>`.

<spaceType>	Required	1	String (3), variable	DASD space allocation type. Values:
				Blk = Blocks
				Cyl = Cylinders
				Trk = Tracks

<unitName>	Optional	0 - 1	String (8), variable	Logical unit name for DASD volume in `<volume>` tag.
<volume>	Optional	0 - 1	String (6), variable	DASD reference volume serial ID.

DSS SERVICE ALLOCATE Replies

No `<result>` data structure is returned in response to a Serena XML data allocation request. However, the reply message does return a standard `<response>` data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Delete a Dataset - DSS SERVICE DELETE

This function physically deletes an empty dataset or a sequential file on the host. Delete permissions must first be defined for you in your mainframe security system before you run this function. You may not delete a ChangeMan ZMF package with this function.

The Serena XML service/scope/message names for a message to *delete* a host dataset are:

```
<service name=”DSS”>
<scope name=”SERVICE”>
<message name=”DELETE”>
```

These tags appear in both requests and replies.

DSS SERVICE DELETE Requests

The Serena XML function to delete a dataset on the host requires only one data element in the `<request>` tag: the name of the dataset to delete. It may also require the serial ID of the DASD volume on which the data set resides if ambiguity exists.

Data structure details for the `<request>` tag appear in *Exhibit 6-2*.

Exhibit 6-2. DSS SERVICE DELETE <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<mvsLib>	Required	1	String (255), variable	Fully qualified dataset name of dataset to be deleted.
<volume>	Optional	0 - 1	String (6), variable	DASD reference volume serial ID.

DSS SERVICE DELETE Replies

No `<result>` data structure is returned in response to a Serena XML data deletion request. However, the reply message does return a standard `<response>` data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Delete a Dataset Member - DSS SERVICE MBRDEL

This function physically deletes the named member of a partitioned dataset — for example, a component in a package in the native z/OS PDS or PDSE file management system. This function does not apply to z/OS Unix Hierarchical File System (HFS) files. Delete permissions must first be defined for you in your mainframe security system before you run this function.

Note

Deleting a dataset member does *not* delete any component records associated with that member in the package and component masters. Component history information is preserved.

The Serena XML service/scope/message names for messages that *delete* a member of a host partitioned data set are:

```
<service name="DSS">
<scope name="SERVICE">
<message name="MBRDEL">
```

These tags appear in both requests and replies.

DSS SERVICE MBRDEL Requests

The Serena XML function to delete a partitioned dataset member on the host requires the dataset name (e.g., for a package) and the name of the member (e.g., package component) to delete.

Data structure details for the `<request>` tag appear in *Exhibit 6-3*.

Exhibit 6-3. DSS SERVICE MBRDEL <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<component>	Required	1	String (8), variable	Name of dataset member to delete, max 8 bytes.
<mvsLib>	Required	1	String (255), variable	Fully qualified dataset name of partitioned dataset containing member to be deleted.

DSS SERVICE MBRDEL Replies

No `<result>` data structure is returned in response to a Serena XML request to delete a dataset member. However, the reply message does return a standard `<response>` data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

List Dataset Allocation Information - DSS SERVICE INFO

This function lists DASD allocation and usage information for a previously allocated dataset. In addition to the specifications given to allocate the dataset initially, the function returns DASD space actually used, creation and expiration dates, date of last access, and a member count. Included in the scope of this function are active ChangeMan ZMF datasets and (optionally) migrated datasets.

The Serena XML service/scope/message names for to *list* DASD usage information for a dataset are:

```
<service name=”DSS”>
<scope name=”SERVICE”>
<message name=”INFO”>
```

These tags appear in both requests and replies.

DSS SERVICE INFO — Requests

The request message for this function requires the fully qualified dataset name of the dataset desired. A yes/no flag tag provides an option to include migrated datasets.

Data structure details for the `<request>` tag of the message to list data set information appear in *Exhibit 6-4*.

Exhibit 6-4. DSS SERVICE INFO <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<mvsLib>	Required	1	String (255), variable	Fully qualified dataset name for which usage information is requested.
<processMigratedDatasets>	Optional	0 - 1	String (1)	Y = Yes, include migrated datasets. N = No, omit migrated datasets.

DSS SERVICE INFO — Replies

Only one `<result>` data element is returned in a reply message for this function. It is followed by a standard `<response>` data structure, which indicates the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Data structure details for the `<result>` tag of a message to list data set information appear in *Exhibit 6-5*.

Exhibit 6-5. DSS SERVICE INFO <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<blockSize>	Optional	0 - 1	Integer (6), variable	Block size allocated to `<mvsLib>` dataset in bytes.
<createDate>	Optional	0 - 1	Date, yyyymmdd	Date dataset was created.

Exhibit 6-5. DSS SERVICE INFO <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<createJob>	Optional	0 - 1	String (8), variable	Job in which dataset was created.
<createStep>	Optional	0 - 1	String (8), variable	Job step in which dataset was created.
<createTime>	Optional	0 - 1	Time, hhmmss	Time dataset was created.

<dataOrg>	Optional	0 - 1	String (8), variable	Data set organization. Values:
				DAM = Direct Access Method
				ISAM = Indexed Sequential Access Method
				MIG = Migrate
				NULL = Null File
				PDS = Partitioned Data Set
				PDSE = Partioned Data Set, Extended
				PS = Physical Sequential
				SEQ = Sequential File
				VSAM = Virtual Sequential Access Method

<dirBlocks>	Optional	0 - 1	Integer (6), variable	Directory allocation for dataset in blocks.
<eAttr>	Optional	0 - 1	String (1)	Extended attribute option. Values: **N** = Dataset cannot have extended attributes or reside in EAS. **O** = Dataset can have extended attributes and reside in EAS. **blank** = Default based on data type.
<expirationDate>	Optional	0 - 1	Date, yyyymmdd	Expiration date for dataset.
<extent>	Optional	0 - 1	Integer (6), variable	Size of dataset extent area used (total), in units specified by `<spaceType>`.
<memberCount>	Optional	0 - 1	Integer (6), variable	Number of members in dataset.
<mvsLib>	Optional	0 - 1	String (255), variable	Fully qualified dataset name for which DASD data is returned.
<percentUsed>	Optional	0 - 1	Integer (6), variable	Percent of total DASD allocation used by dataset.
<primarySpace>	Optional	0 - 1	String (8), variable	Minimum DASD allocation for dataset, in units specified by `<spaceType>`.

Exhibit 6-5. DSS SERVICE INFO <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies

<recordFormat>	Optional	0 - 1	String(3), variable	Code for logical data set record format. Values:
				F = Fixed
				FA = Fixed ASA
				FM = Fixed Machine
				FB = Fixed Block
				FBA = Fixed Block ASA
				FBM = Fixed Block Machine
				FBS = Fixed Block Standard
				FS = Fixed Standard
				FSA = Fixed Standard ASA
				FSM = Fixed Standard Machine
				V = Variable
				VA = Variable ASA
				VM = Variable Machine
				VB = Variable Block
				VBA = Variable Block ASA
				VBM = Variable Block Machine
				VS = Variable Spanned
				VSA = Variable Spanned ASA
				VSM =Variable Spanned Machine
				U = Undefined
				UA = Undefined ASA
				UM = Undefined Machine
				UB = Undefined Block
				UBA = Undefined Block ASA
				UBM = Undefined Block Machine
				US = Undefined Spanned
				USA = Undefined Spanned ASA
				USM =Undefined Spanned Machine

<recordLength>	Optional	0 - 1	Integer (6), variable	Record length for dataset in bytes.
<referenceDate>	Optional	0 - 1	Date, yyyymmdd	Date of last access for dataset.
<secondarySpace>	Optional	0 - 1	String (8), variable	DASD allocation for extents, in units specified by `<spaceType>`.

<spaceType>	Optional	0 - 1	String (3), variable	Type of DASD space allocation for `<mvsLib>` dataset. Values:
				Blk = Blocks
				Cyl = Cylinders
				Trk = Tracks

<tracks>	Optional	0 - 1	Integer (10), variable	Total number of tracks currently allocated to dataset.
<unitName>	Optional	0 - 1	String (8), variable	Logical unit name for DASD volume in `<volume>` tag.

Exhibit 6-5. DSS SERVICE INFO <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<usedDirBlocks>	Optional	0 - 1	Integer (6), variable	Number of directory blocks actually used.
<usedPages>	Optional	0 - 1	Integer (10), variable	Total number of pages actually used by PDSE dataset.
<usedTracks>	Optional	0 - 1	Integer (10), variable	Total number of tracks actually used by non-PDSE dataset.
<volume>	Optional	0 - 1	String (6), variable	DASD reference volume serial ID for dataset in `<mvsLib>`.

List Dataset Member Directory - DSS SERVICE LIST

This function lists a directory of dataset member names or sequential file names. Optionally it provides information about DASD usage, usage change history, and access history for each member to authorized requestors. A single directory request can retrieve information about PDS dataset members, sequential files, and ISPF files. Only active ChangeMan ZMF datasets are included in the scope of this function; no directory service is available for migrated datasets.

The Serena XML service/scope/message names for a message to *list* a dataset member directory are:

```
<service name=”DSS”>
<scope name=”SERVICE”>
<message name=”LIST”>
```

These tags appear in both requests and replies.

DSS SERVICE LIST — Requests

Serena XML supports two types of dataset member directory lists:

- ***Full Directory Contents*** — Name the dataset of interest in the `<mvsLib>` tag and set the `<listComponentOnly>` flag tag value to N. If the component fingerprint or hash token is desired in the listing, set the `<returnHashToken>` flag tag to Y. The function lists a complete directory of dataset member names, DASD usage, and access history information.
- ***Member Name Only*** — Name the dataset of interest in the `<mvsLib>` tag and set the `<listComponentOnly>` flag tag value to Y. The returned listing includes only member names.

Additional filtering by component name or wildcard pattern is supported for both list types. Data structure details for the `<request>` tag appear in *Exhibit 6-6*.

Exhibit 6-6. DSS SERVICE LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<component>	Optional	0 - 1	String (32), variable	Name or wildcard pattern matching dataset members (components) to list. ***NOTE:*** Use asterisk (*) wildcard or omit tag to list all members of a dataset.
<listComponentOnly>	Optional	1	String (1)	Y = Yes, list component names only. N = No, don't restrict to component names; list all DASD info.
<mvsLib>	Required	1	String (255), variable	Fully qualified dataset name whose members are to be listed.
<returnHashToken>	Optional	1	String (1)	Y = Yes, return fingerprinting hash token for each member in list. N = No, omit hash token.
<typeOfDataset>	Optional	1	String(3), variable	Dataset organization. Values: LIB - Librarian OTH - Other PAN - Panvalet PDS - Partitioned dataset SEQ - Sequential

DSS SERVICE LIST — Replies

The Serena XML reply message for this function contains zero to many `<result>` data elements. Each `<result>` provides directory information for one dataset member.

A standard `<response>` data structure follows the final `<result>` tag to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because the `<response>` tag is the last data element returned in the reply, it also serves as an end-of-list indicator.

Data structure details for the `<result>` data structure appear in *Exhibit 6-7*.

Exhibit 6-7. DSS SERVICE LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<aliasOf>	Optional	0 - 1	String (8), variable	Name of component for which this member serves as an alias. Returned for load members only.
<amode>	Optional	0 - 1	String (10), variable	Link-edit parameters for addressing mode of component. Returned for load members only.
<attributes>	Optional	0 - 1	Integer (6), variable	System attribute flags for member. Returned for load members only.

Exhibit 6-7. DSS SERVICE LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<authCode>	Optional	0 - 1	String (2), variable	Authentication code for component. Returned for load members only.
<component>	Required	1	String (8), variable	Name of dataset member returned.
<currentSize>	Optional	0 - 1	Integer (6), variable	Current size of member in bytes. Returned for ISPF files only.
<dateCreated>	Optional	0 - 1	Date, yyyymmdd	Date component was created. Returned for ISPF files only.
<dateLastModified>	Optional	0 - 1	Date, yyyymmdd	Date component was last modified. Returned for ISPF files only.
<extCurrentSize>	Optional	0 - 1	Long	Extended size - current (for ISPF directory type)
<extInitialSize>	Optional	0 - 1	Long	Extended size - initial (for ISPF directory type)
<extModificationSize>	Optional	0 - 1	Long	Extended size - modifications (for ISPF directory type)
<hashToken>	Optional	0 -1	String (16)	Hash token (if requested by returnHashToken)
<initialSize>	Optional	0 - 1	Integer (6), variable	Size of member when first created in bytes. Returned for ISPF files only.
<lastUpdater>	Optional	0 - 1	String (8), variable	TSO ID of last updater. Returned for ISPF files only.
<linkedDate>	Optional	0 - 1	Date, yyyymmdd	Date member was last link-edited. Returned for load members only.
<linkedTime>	Optional	0 - 1	Time, hhmmss	Time component was last link-edited. Returned for load members only.
<loadSize>	Optional	0 - 1	Integer (6), variable	Size of load member in bytes. Returned for load members only.
<modLevel>	Optional	0 - 1	String (2), variable	ISPF modification level of component. Returned for ISPF files only.
<modificationSize>	Optional	0 - 1	Integer (6), variable	Size of last modification to member in bytes. Returned for ISPF files only.
<relativeTrack>	Optional	0 - 1	Integer (6), variable	Relative offset of starting track for this dataset member. Returned for load members only.
<rmode>	Optional	0 - 1	String (10), variable	Link-edit parameters for residency mode of component. Returned for load members only.
<setssi>	Optional	0 - 1	String (8)	SETSSI for LOAD directory type.
<timeLastModified>	Optional	0 - 1	Time, hhmmss	Time component was last modified. Returned for ISPF files only.

Exhibit 6-7. DSS SERVICE LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<typeOfDirectory>	Required	1	String (8), variable	Code for type of directory containing dataset member. Values: 1 = ISPF 2 = Load 3 = None ***NOTE:*** If value is 1, the following tags are also returned: `<currentSize>`, `<initialSize>`, `<modificationSize>`, `<dateCreated>`, `<dateLastModified>`, `<timeLastModified>`, `<version>`, `<modLevel>`, & `<lastUpdater>`. ***NOTE:*** If value is 2, the following tags are also returned: `<aliasOf>`, `<attributes>`, `<authCode>`, `<relativeTrack>`, `<loadSize>`, `<amodeRmode>`, `<setssi>`, `<linkedDate>`, & `<linkedTime>`.
<version>	Optional	0 - 1	String (2), variable	ISPF version number of component. Returned for ISPF files only.

List ISPF Dataset Allocation Information - DSS ISPFILE INFO

This function lists temporary dataset allocation information for a ChangeMan ZMF started task. Typically this information concerns datasets used for ISPF file tailoring. Returned information includes the dataset name, DASD unit name, and DASD allocation amounts for each temporary dataset known to the started task at the global level.

The Serena XML service/scope/message names for a message to *list* ISPF dataset allocation information are:

```
<service name="DSS">
<scope name="ISPFILE">
<message name="INFO">
```

These tags appear in both requests and replies.

DSS ISPFILE INFO — Requests

The Serena XML request message for this function includes an empty `<request>` data element (that is, one that contains no subtags). The `<request>` tag itself is required.

Note

XML syntax allows both a long form and a short form for empty tags. An empty `<request>` tag can therefore be coded in one of two ways.

Long form:

```
<request>
</request>
```

Equivalent short form:

```
<request/>
```

DSS ISPFILE INFO — Replies

The Serena XML reply message for this function contains zero to many `<result>` data elements. Each `<result>` provides DASD allocation information for one ISPF file tailoring dataset (or other temporary dataset).

A standard `<response>` data structure follows the final `<result>` tag to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because the `<response>` tag is the last data element returned in the reply, it also serves as an end-of-list indicator.

Data structure details for the `<result>` data structure appear in *Exhibit 6-8*.

Exhibit 6-8. DSS ISPFILE INFO <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<blockSize>	Optional	1	Integer (6), variable	Block size allocated to `<mvsLib>` dataset in bytes.
<eAttr>	Optional	0 - 1	String (1)	Extended attribute option. Values: **N** = Dataset cannot have extended attributes or reside in EAS. **O** = Dataset can have extended attributes and reside in EAS. **blank** = Default based on data type.
<lastNodeOrigin>	Optional	0 - 1	Integer (6), variable	Code for origin of last node in dataset name when dataset used to store a component in a temporary dataset. Values: 1 = Component library type 2 = Component language 3 = Component name
<mvsLib>	Optional	1	String (255), variable	Fully qualified dataset name for which DASD data is returned.
<primarySpace>	Optional	0 - 1	String (8), variable	Minimum DASD allocation for dataset, in units specified by `<spaceType>`.

Exhibit 6-8. DSS ISPFILE INFO <result> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<secondarySpace>	Optional	0 - 1	String (8), variable	DASD allocation for extents, in units specified by `<spaceType>`.

<spaceType>	Required	1	String (3), variable	Type of DASD space allocation for `<mvsLib>` dataset. Values:
				Blk = Blocks
				Cyl = Cylinders
				Trk = Tracks

<unitName>	Required	1	String (8), variable	Logical unit name for DASD volume in `<volume>` tag.

List Statistics for Baseline Members - DSS SERVICE BASESTAT

This function lists statistics for baseline library members stored in PDS or SRD (stacked reverse delta) format.

The Serena XML service/scope/message names for a message to *list* statistics for baseline members are:

```
<service name=”DSS”>
<scope name=”SERVICE”>
<message name=”BASESTAT”>
```

These tags appear in both requests and replies.

DSS SERVICE BASESTAT — Requests

The following example shows how you might code a Serena XML request to list the statistics for a baseline library component. Data structure details for the `<request>` tag appear in *Exhibit 6-9*.

Example XML — DSS SERVICE BASESTAT Request

```
<?xml version="1.0"?>
<service name="DSS">
 <scope name="SERVICE">
  <message name="BASESTAT">
   <header>
    <subsys>4</subsys>
    <test> </test>
    <product>CMN</product>
   </header>
  <request>
    <applName>ACTP</applName>
    <libType>SRC</libType>
    <version>001</version>
    <component>ACPSRC1A</component>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 6-9. DSS SERVICE BASESTAT <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	1	String (4), variable	Application name.
<component>	Required	1	String (8), variable	Component name.
<libType>	Required	1	String (3)	Library type.
<version>	Required	1	Integer (3)	Baseline version number.

DSS SERVICE BASESTAT — Replies

Only one `<result>` data element is returned in the reply message for this function. It is followed by a standard `<response>` data structure, which indicates the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

The following example shows what a reply message might look like. Data structure details for the `<result>` tag appear in *Exhibit 6-10*.

Example XML — DSS SERVICE BASESTAT Result

```
<?xml version="1.0"?>
<service name="DSS">
 <scope name="SERVICE">
  <message name="BASESTAT">
   <result>
    <createDate>20120407</createDate>
    <dateLastModified>20130427</dateLastModified>
    <timeLastModified>173204</timeLastModified>
    <maxVersion>001</maxVersion>
    <user>USER99</user>
   </result>
   <response>
    <statusMessage>CMN8700I - Baseline stat service completed
     </statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Exhibit 6-10. DSS SERVICE BASESTAT <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<createDate>	Optional	0 - 1	Date, yyyymmdd	Date created.
<dateLastModified>	Optional	0 - 1	Date, yyyymmdd	Date last modified.
<maxVersion>	Optional	0 - 1	Integer (3)	Total number of baseline versions. ***NOTE:*** This tag is only returned for a SRD baseline member. The maximum versions for a PDS member is not known.
<timeLastModified>	Optional	0 - 1	Time, hhmmss	Time last modified.
<user>	Optional	0 - 1	String (8), variable	User ID of user who last modified component.

Expand Member in SRD Format - DSS SERVICE EXPAND

This function expands a baseline library member stored in stacked reverse delta (SRD) format and writes it to a data set.

The Serena XML service/scope/message names for a message to expand a baseline member are:

```
<service name=”DSS”>
<scope name=”SERVICE”>
<message name=”EXPAND”>
```

These tags appear in both requests and replies.

DSS SERVICE EXPAND — Requests

The following example shows how you might code a Serena XML request to expand a baseline library component in SRD format and write it to the dataset specified in the `<targetLib>` subtag. Data structure details for the `<request>` tag appear in *Exhibit 6-11*.

Example XML — DSS SERVICE EXPAND Request

```
<?xml version="1.0"?>
<service name="DSS">
 <scope name="SERVICE">
  <message name="EXPAND">
   <header>
```

```
    <subsys>4</subsys>
    <test> </test>
    <product>CMN</product>
   </header>
  <request>
    <baseLib>WSER99.BASE0.SRC</baseLib>
    <deltaLib>WSER99.BASE1.SRC</deltaLib>
    <targetLib>WSER99.TEMP.PDS</targetLib>
    <version>002</version>
    <component>ACPSJW2</component>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 6-11. DSS SERVICE EXPAND <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<baseLib>	Required	1	String (255), variable	Baseline library name.
<component>	Required	1	String (8), variable	Component name.
<deltaLib>	Required	1	String (255), variable	Delta library name.
<targetLib>	Required	1	String (255), variable	Dataset name where expanded component will be written.
<version>	Required	1	Integer (3)	Version number.

DSS SERVICE EXPAND — Replies

No `<result>` data structure is returned for this function. However, the reply message does return a standard `<response>` data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

In a successful request, the SRD component will be expanded and written to the dataset specified in the `<targetLib>` subtag.

The following example shows what a successful reply message looks like.

Example XML — DSS SERVICE EXPAND Result

```
<?xml version="1.0"?>
<service name="DSS">
 <scope name="SERVICE">
  <message name="EXPAND">
   <response>
```

```
    <statusMessage>CMN8700I - Expand service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

HIERARCHICAL FILE SYSTEM SERVICES 7

OVERVIEW

The ZMF XML Services let you work directly with the z/OS Unix Hierarchical File System (HFS) on the mainframe. Operations on files and directories are supported.

Hierarchical File System Functions

The following HFS data management functions are supported for general use:

- ***HFS Directory Services*** — Create, delete, rename, or list the contents of a directory.
- ***HFS File Lifecycle Services*** — Create, delete, rename, or copy an HFS file, change certain file attributes, test for file existence and verify user access permissions, or scan files for strings.
- ***File Conversion Services*** — Import a z/OS PDS (Partitioned Data Set) member as an HFS file or export an HFS file as a PDS member.

High-Level Syntax

The syntax that identifies a z/OS Unix HFS message in Serena XML appears in the `name` attribute of the `<service>` tag, as follows:

```
<service name=”file”>
```

HFS directory services use the following `<scope>` tags:

```
<scope name=”dirs”>
<scope name=”files”>
<scope name=”service”>
```

HFS file lifecycle services and file transfer/conversion services both use the following `<scope>` tag:

```
<scope name=”service”>
```

Case is significant in all attribute values (delimited by double quotes).

Related Services

Package, component, and other services have been extended to support the new HFS data objects with new tags or new allowed values in existing tags. See the chapters addressing the relevant package, component, or other services for details.

Services to manage native z/OS PDS datasets are discussed in *Chapter 6, "Dataset Management," on page 361*.

Database files and tables managed under DB2 or IMS are discussed in *Chapter 8, "Database Management," on page 405*.

Note

Each Service listed requires the user ID running the service to have the appropriate access permissions for the resources requested defined for it in your mainframe security system.

HFS Directory Services

Serena XML supports the following HFS directory functions for general use:

- *Create a Directory — FILE SERVICE MKDIR*
- *Delete a Directory — FILE SERVICE RMDIR*
- *Rename a Directory — FILE SERVICE RENAME*
- *List All Directory Contents — FILE SERVICE LIST*
- *List Files in a Directory — FILE FILES LIST*
- *List Directories in a Directory — FILE DIRS LIST*

Create a Directory — FILE SERVICE MKDIR

This function allocates an empty HFS directory on the host. The resulting data object has a "file type code" of 1. Access permissions for the resources requested must first be defined for you in your mainframe security system.

The XML service/scope/message names for a message to *create* an HFS directory are:

```
<service name="file">
<scope name="service">
<message name="mkdir">
```

These tags appear in both requests and replies. Case is significant.

FILE SERVICE MKDIR Requests

Data structure details for the `<request>` tag appear in Exhibit 7-1. All subtags are required.

Exhibit 7-1. FILE SERVICE MKDIR <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies

<createInterPath>	Required	1	String (1)	Include intermediate path in `<pathName>`?
				`Y` = Yes, include full path from installation root node as prefix to new directory name in `<pathName>`
				`N` = No, omit path; include only the name of the new directory in `<pathName>`

<pathName>	Required	1	String (1024), variable	Name of new directory, optionally prefixed by path from installation root. • If `<createInterPath> = Y`, `<pathName>` includes full path name from install root plus name of new directory. • If `<createInterPath> = N`, `<pathName>` is new directory name only.
<permissions>	Required	1	Integer (3), fixed	Unix access permissions for new directory, coded as 3-digit integer, where: 1st digit = owner permissions 2nd digit = group permissions 3rd digit = permissions for all others Each digit takes one of the following values: 7 - Read, write/rename/delete, execute 6 - Read, write/rename/delete 5 - Read, execute 4 - Read only 3 - Write/rename/delete, execute 2 - Write/rename/delete only 1 - Execute only 0 - No access permitted NOTE: For directories, "execute" permissions should be read as "open directory" or "view files in directory".

FILE SERVICE MKDIR Results

No `<result>` data structure is returned in response to an XML HFS directory creation request. However, the reply message does return a standard `<response>` data structure to indicate the success or failure of the request.

Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Delete a Directory — FILE SERVICE RMDIR

This function deletes an HFS directory on the host. Access permissions for the resources requested must first be defined for you in your mainframe security system.

The XML service/scope/message names for a message to *delete* an HFS directory are:

```
<service name="file">
<scope name="service">
<message name="rmdir">
```

These tags appear in both requests and replies. Case is significant.

FILE SERVICE RMDIR Requests

Data structure details for the `<request>` tag appear in Exhibit 7-2. All subtags are required.

Exhibit 7-2. FILE SERVICE RMDIR <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<deleteContents>	Required	1	String (1)	If the directory is not empty, should the deletion go forward anyway, taking with it any subordinate files or directories? `Y` = Yes, delete even if the directory has content. `N` = No, don't delete if the directory has content,
<pathName>	Required	1	String (1024), variable	Name of directory to be deleted, prefixed by path from installation root.

FILE SERVICE RMDIR Results

No `<result>` data structure is returned in response to an XML HFS directory deletion request. However, the reply message does return a standard `<response>` data structure to indicate the success or failure of the request.

Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Rename a Directory — FILE SERVICE RENAME

This function renames an HFS directory (or file) on the host. Access permissions for the resources requested must first be defined for you in your mainframe security system.

The XML service/scope/message names for a message to *rename* an HFS directory are:

```
<service name="file">
<scope name="service">
<message name="rename">
```

These tags appear in both requests and replies. Case is significant.

FILE SERVICE RENAME Requests

Data structure details for the `<request>` tag appear in Exhibit 7-3. All subtags are required.

Exhibit 7-3. FILE SERVICE RENAME <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<fromPathName>	Required	1	String (1024), variable	Old name of directory, prefixed by path from installation root.
<replace>	Required	1	String (1)	If the new directory name is already in use, should that preexisting directory be deleted so the rename can proceed? `Y` = Yes, replace any existing directory of the same name with the renamed directory. `N` = No, don't rename if another directory with the same name already exists.
<toPathName>	Required	1	String (1024), variable	New name of directory, prefixed by path from installation root.

FILE SERVICE RENAME Results

No `<result>` data structure is returned in response to an XML HFS directory rename request. However, the reply message does return a standard `<response>` data structure to indicate the success or failure of the request.

Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

List All Directory Contents — FILE SERVICE LIST

This function lists allocation information for all Unix data object contained in an HFS directory. The listing includes both files and subdirectories. Subdirectories may be expanded recursively for listing. Other output filtering options also exist. Access permissions for the resources requested must first be defined for you in your mainframe security system.

The XML service/scope/message names for a message to *list all contents* of an HFS directory are:

```
<service name="file">
<scope name="service">
<message name="list">
```

These tags appear in both requests and replies. Case is significant.

Optimized variants of this service are:

- *List Files in a Directory — FILE FILES LIST*

  Same as FILE SERVICE LIST with subdirectories excluded from listing.
- *List Directories in a Directory — FILE DIRS LIST*

Same as FILE SERVICE LIST with `<fileType>` set to `1`.

FILE SERVICE LIST Requests

Data structure details for the `<request>` tag appear in Exhibit 7-4.

Exhibit 7-4. FILE SERVICE LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<caseSensitive>	Required	1	String (1)	Treat values in `<fileName>` and `<pathName>` as case sensitive? `Y` = Yes, respect case as typed in tags. `N` = No, normalize values to upper case.
<expandDirectory>	Required	1	String (1)	Expand and list contents of subdirectories? `Y` = Yes, expand & list subdirectories `N` = No, don't expand subdirectories
<fileName>	Optional	0 - 1	String (256), variable	Name of a particular file, subdirectory, or other Unix data object in the directory identified by `<pathName>` whose allocation information should be listed. Path is not included. Wildcards '*" and '?' may be used. If this tag is used, only the named data object is included in the returned listing. If omitted, all data objects of the type requested in `<fileType>` are listed. NOTE: Concurrent use of `<fileName>` and `<fileType>` is not recommended. If the actual file type of the data object in `<fileName>` conflicts with the value in `<fileType>`, no objects will be listed, even if an object with the requested name exists in the directory.
<fileType>	Optional	0 - 1	Integer (1)	Unix data object type desired in listing. If used, only objects of the type specified are listed. If omitted, all object types are listed. Values: `1` = Directory `2` = Character special file `3` = Regular file `4` = Named pipe (FIFO) file `5` = Symbolic link (alias/shortcut to other file) `6` = Reserved for block special file `7` = Socket file NOTE: Concurrent use of `<fileType>` and `<fileName>` is not recommended. If the actual file type of the named data object conflicts with the value supplied in `<fileType>`, no data will be returned, even if a data object with the requested name exists in the directory.
<pathName>	Required	1	String (1024), variable	Name of directory whose contents are to be listed, prefixed by path from installation root.

Exhibit 7-4. FILE SERVICE LIST <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<returnHashToken>	Required	1	String (1)	Return a ZMF hash token with any listed files? `Y` = Yes, return hash token for files `N` = No, don't return hash token

FILE SERVICE LIST Results

Data structure details for the `<result>` tag appear in Exhibit 7-5. One result data structure is returned for each file, directory, or other Unix data object listed.

A standard `<response>` data structure is returned after the final `<result>` tag to indicate success or failure of the request and the completion of the listing if successful. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Exhibit 7-5. FILE SERVICE LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<createdDate>	Required	1	Integer (8), fixed	Creation date in yyyyMMdd format.
<createdTime>	Required	1	Integer (6), fixed	Creation time in HHmmss format.
<fileFormat>	Optional	0 - 1	Integer (1)	Record formatting and delimiter convention used by listed file or data object. Values: `0` = Not specified `1` = Binary data `2` = New line (NL) `3` = Carriage return (CR) `4` = Line feed (LF) `5` = CR & LF `6` = LF & CR `7` = CR & NL NOTE: Not relevant for directories.
<fileGroupOwner>	Required	1	String (8), variable	User ID of file group owner associated with the listed file or data object.
<fileName>	Required	1	String (256), variable	Name of the current file, subdirectory, or other Unix data object listed.
<fileOwner>	Required	1	String (8), variable	User ID of owner associated with the listed file or data object.
<fileSize>	Optional	0 - 1	Integer (9), variable	File or directory size in bytes. Values 0 - 999999999. NOTE: For directories, size is for directory object itself, not for files within directory.

Exhibit 7-5. FILE SERVICE LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<fileType>	Required	1	Integer (1)	Unix data object type of listed file or subdirectory. Values: 1 = Directory 2 = Character special file 3 = Regular file 4 = Named pipe (FIFO) file 5 = Symbolic link (alias/shortcut to other file) 6 = Reserved for block special file 7 = Socket file NOTE: If `<fileType>` = 5, `<linkName>`, `<linkType>` will also be present.
<hashToken>	Optional	0 - 1	Integer (16), fixed	ZMF hash token stored with a component. Can be compared to a reference value to determine whether component has changed. NOTE: Not applicable to directories.
<lastAccessedDate>	Required	1	Integer (8), fixed	Date last accessed in yyyyMMdd format.
<lastAccessedTime>	Required	1	Integer (6), fixed	Time last accessed in HHmmss format.
<lastModifiedDate>	Required	1	Integer (8), fixed	Date last modified in yyyyMMdd format.
<lastModifiedTime>	Required	1	Integer (6), fixed	Time last modified in HHmmss format.
<linkCount>	Required	1	Integer (5), variable	Count of inbound links pointing to this data object. Allowed values 0 - 65536. • For files (`<fileType>` ≠ 1) the link count is the number of hard links pointing to the file. Symbolic or soft links (`<fileType>` = 5) are not included in this count. • For directories (`<fileType>` = 1), the link count is the number of subdirectories.
<linkName>	Optional	0 - 1	String (1024), variable	Name of file pointed to by the current link or alias, prefixed by path from installation root. NOTE: Required if `<fileType>` = 5, not applicable otherwise.

Exhibit 7-5. FILE SERVICE LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<linkType>	Optional	0 - 1	Integer (1)	Unix data object type of file identified in `<linkName>`. Values: `1` = Directory `2` = Character special file `3` = Regular file `4` = Named pipe (FIFO) file `5` = Symbolic link (points to another file) `6` = Reserved for block special file `7` = Socket file NOTE: Required if `<fileType>` = `5`, not applicable otherwise.
<permissions>	Required	1	Integer (3), fixed	Unix access permissions for listed file or subdirectory. Coded as 3-digit integer, where: 1st digit = owner permissions 2nd digit = group permissions 3rd digit = permissions for all others Each digit takes one of the following values: 7 - Read, write/rename/delete, execute 6 - Read, write/rename/delete 5 - Read, execute 4 - Read only 3 - Write/rename/delete, execute 2 - Write/rename/delete only 1 - Execute only 0 - No access permitted NOTE: For directories, "execute" permissions should be read as "open directory" or "view files in directory".
<realPath>	Optional	0 - 1	string (1024), variable	Real path name.

List Files in a Directory — FILE FILES LIST

This function lists all the files in an HFS directory. Under the covers, it is the same service as FILES SERVICE LIST with certain predefined request options and output filters built in. Access permissions for the resources requested must first be defined for you in your mainframe security system.

The XML service/scope/message names for a message to *list only files* in an HFS directory are:

```
<service name="file">
<scope name="files">
<message name="list">
```

These tags appear in both requests and replies. Case is significant.

FILE FILES LIST Requests

Data structure details for the `<request>` tag appear in Exhibit 7-6. All subtags shown are required. Request tags `<fileName>` and `<fileType>` should not be used.

Exhibit 7-6. FILE FILES LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<caseSensitive>	Required	1	String (1)	Treat values in `<fileName>` and `<pathName>` as case sensitive? `Y` = Yes, respect case as typed in tags. `N` = No, normalize values to upper case.
<expandDirectory>	Required	1	String (1)	Expand and list contents of subdirectories? `Y` = Yes, expand & list subdirectories `N` = No, don't expand subdirectories
<pathName>	Required	1	String (1024), variable	Name of directory whose contents are to be listed, prefixed by path from installation root.
<returnHashToken>	Required	1	String (1)	Return a ZMF hash token with the listed files? `Y` = Yes, return hash token for files `N` = No, don't return hash token

FILE FILES LIST Results

Data structure details for the `<result>` tag appear in Exhibit 7-7. One result data structure is returned for each file listed.

A standard `<response>` data structure is returned after the final `<result>` tag to indicate success or failure of the request and the completion of the listing if successful. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Exhibit 7-7. FILE FILES LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<createdDate>	Required	1	Integer (8), fixed	Creation date in yyyyMMdd format.
<createdTime>	Required	1	Integer (6), fixed	Creation time in HHmmss format.

Exhibit 7-7. FILE FILES LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<fileFormat>	Optional	0 - 1	Integer (1)	Record formatting and delimiter convention used by listed file. Values: `0` = Not specified `1` = Binary data `2` = New line (NL) `3` = Carriage return (CR) `4` = Line feed (LF) `5` = CR & LF `6` = LF & CR `7` = CR & NL
<fileGroupOwner>	Required	1	String (8), variable	User ID of file group owner.
<fileName>	Required	1	String (256), variable	Name of listed file.
<fileOwner>	Required	1	String (8), variable	User ID of file owner.
<fileSize>	Optional	0 - 1	Integer (9), variable	File size in bytes. Values 0 - 999999999. NOTE: For directories, size is for directory object itself, not for files within directory.
<fileType>	Required	1	Integer (1)	Unix data object type of listed file. Values: `2` = Character special file `3` = Regular file `4` = Named pipe (FIFO) file `5` = Symbolic link (points to another file) `6` = Reserved for block special file `7` = Socket file NOTE: If `<fileType>` = `5`, `<linkName>` and `<linkType>` will also be present.
<hashToken>	Optional	0 - 1	Integer (16), fixed	ZMF hash token stored with a component. Can be compared with a stored value to determine whether the component has changed.
<lastAccessedDate>	Required	1	Integer (8), fixed	Date last accessed in yyyyMMdd format.
<lastAccessedTime>	Required	1	Integer (6), fixed	Time last accessed in HHmmss format.
<lastModifiedDate>	Required	1	Integer (8), fixed	Date last modified in yyyyMMdd format.
<lastModifiedTime>	Required	1	Integer (6), fixed	Time last modified in HHmmss format.
<linkCount>	Required	1	Integer (5), variable	Count of inbound hard links pointing to this file. Symbolic or soft links (aliases/shortcuts) are not included in count. Allowed values 0 - 65536.

Exhibit 7-7. FILE FILES LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<linkName>	Optional	0 - 1	String (1024), variable	Name of file pointed to by the listed link file (ie alias or shortcut) in `<fileName>`, prefixed by path from installation root. NOTE: Required if `<fileType>` = `5`, not applicable otherwise.
<linkType>	Optional	0 - 1	Integer (1)	Unix data object type of file or data object identified in `<linkName>`. Values: `1` = Directory `2` = Character special file `3` = Regular file `4` = Named pipe (FIFO) file `5` = Symbolic link (points to another file) `6` = Reserved for block special file `7` = Socket file NOTE: Required if `<fileType>` = `5`, not applicable otherwise.
<permissions>	Required	1	Integer (3), fixed	Unix access permissions for listed file. Coded as 3-digit integer, where: 1st digit = owner permissions 2nd digit = group permissions 3rd digit = permissions for all others Each digit takes one of the following values: 7 - Read, write/rename/delete, execute 6 - Read, write/rename/delete 5 - Read, execute 4 - Read only 3 - Write/rename/delete, execute 2 - Write/rename/delete only 1 - Execute only 0 - No access permitted NOTE: For directories, “execute” permissions should be read as “open directory” or “view files in directory”.
<realPath>	Optional	0 - 1	string (1024), variable	Real path name.

List Directories in a Directory — FILE DIRS LIST

This function lists all the subdirectories in an HFS directory. Under the covers, it is the same service as FILES SERVICE LIST with certain predefined request options and output filters built in. Access permissions for the resources requested must first be defined for you in your mainframe security system.

The XML service/scope/message names for a message to *list only subdirectories* in an HFS directory are:

```
<service name="file">
<scope name="dirs">
<message name="list">
```

These tags appear in both requests and replies. Case is significant.

FILE DIRS LIST Requests

Data structure details for the `<request>` tag appear in Exhibit 7-8. All subtags shown are required. Request tags `<fileName>`, `<fileType>`, and `<returnHashToken>` should not be used.

Exhibit 7-8. FILE DIRS LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<caseSensitive>	Required	1	String (1)	Treat values in `<fileName>` and `<pathName>` as case sensitive? `Y` = Yes, respect case as typed in tags. `N` = No, normalize values to upper case.
<expandDirectory>	Required	1	String (1)	Expand and list contents of subdirectories? `Y` = Yes, expand & list subdirectories `N` = No, don't expand subdirectories
<pathName>	Required	1	String (1024), variable	Name of directory whose contents are to be listed, prefixed by path from installation root.

FILE DIRS LIST Results

Data structure details for the `<result>` tag appear in Exhibit 7-9. One result data structure is returned for each subdirectory listed. File-specific subtags such as `<fileFormat>`, `<hashToken>`, `<linkName>`, and `<linkType>` are not returned.

A standard `<response>` data structure is returned after the final `<result>` tag to indicate success or failure of the request and the completion of the listing if successful. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Exhibit 7-9. FILE DIRS LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<createdDate>	Required	1	Integer (8), fixed	Creation date in `yyyyMMdd` format.
<createdTime>	Required	1	Integer (6), fixed	Creation time in `HHmmss` format.
<fileGroupOwner>	Required	1	String (8), variable	User ID of file group owner associated with this subdirectory.

Exhibit 7-9. FILE DIRS LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<fileName>	Required	1	String (256), variable	Name of listed subdirectory.
<fileOwner>	Required	1	String (8), variable	User ID of file owner associated with this directory
<fileType>	Required	1	Integer (1)	Unix data object type of listed directory. Values: `1` = Directory
<lastAccessedDate>	Required	1	Integer (8), fixed	Date last accessed in `yyyyMMdd` format.
<lastAccessedTime>	Required	1	Integer (6), fixed	Time last accessed in `HHmmss` format.
<lastModifiedDate>	Required	1	Integer (8), fixed	Date last modified in `yyyyMMdd` format.
<lastModifiedTime>	Required	1	Integer (6), fixed	Time last modified in `HHmmss` format.
<linkCount>	Optional	0 - 1	Integer (5), variable	Count of subdirectories within the listed directory. Allowed values 0 - 65536.
<permissions>	Required	1	Integer (3), fixed	Unix access permissions for listed subdirectory. Coded as 3-digit integer, where: 1st digit = owner permissions 2nd digit = group permissions 3rd digit = permissions for all others Each digit takes one of the following values: 7 - Read, write/rename/delete, execute 6 - Read, write/rename/delete 5 - Read, execute 4 - Read only 3 - Write/rename/delete, execute 2 - Write/rename/delete only 1 - Execute only 0 - No access permitted NOTE: For directories, “execute” permissions should be read as “open directory” or “view files in directory”.
<realPath>	Optional	0 - 1	string (1024), variable	Real path name.

HFS File Lifecycle Services

The XML Services API supports the following HFS file lifecycle functions for general use:

- *Create a File — FILE SERVICE CREATE*
- *Delete a File — FILE SERVICE DELETE*
- *Rename a File — FILE SERVICE RENAME*
- *Copy a File — FILE SERVICE COPY*
- *Create a Link or Alias to a File — FILE SERVICE LINK*
- *Change File Attributes — FILE SERVICE CHANGE*
- *Check Access to a File — FILE SERVICE ACCESS*
- *Scan Files for Strings — FILE SERVICE SCAN*

Create a File — FILE SERVICE CREATE

This function allocates an empty HFS file on the host. The resulting data object has a "file type code" of 3. Access permissions for the resources requested must first be defined for you in your mainframe security system.

The XML service/scope/message names for a message to *create* an HFS file are:

```
<service name="file">
<scope name="service">
<message name="create">
```

These tags appear in both requests and replies. Case is significant.

FILE SERVICE CREATE Requests

Data structure details for the `<request>` tag appear in Exhibit 7-10. All subtags are required.

Exhibit 7-10. FILE SERVICE CREATE <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<createInterPath>	Required	1	String (1)	Include intermediate path in `<pathName>`? `Y` = Yes, include full path from installation root in `<pathName>` value `N` = No, omit path; include only the name of the new file in `<pathName>`
<pathName>	Required	1	String (1024), variable	Name of new file to be created, optionally prefixed by path from installation root. • If `<createInterPath> = Y`, `<pathName>` includes full path from installation root plus name of new file. • If `<createInterPath> = N`, `<pathName>` is new file name only.

Exhibit 7-10. FILE SERVICE CREATE <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<permissions>	Required	1	Integer (3), fixed	Unix access permissions for new file, coded as 3-digit integer, where: 1st digit = owner permissions 2nd digit = group permissions 3rd digit = permissions for all others Each digit takes one of the following values: 7 - Read, write/rename/delete, execute 6 - Read, write/rename/delete 5 - Read, execute 4 - Read only 3 - Write/rename/delete, execute 2 - Write/rename/delete only 1 - Execute only 0 - No access permitted NOTE: For directories, “execute” permissions should be read as “open directory” or “view files in directory”.

FILE SERVICE CREATE Results

No `<result>` data structure is returned in response to an XML HFS file creation request. However, the reply message does return a standard `<response>` data structure to indicate the success or failure of the request.

Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Delete a File — FILE SERVICE DELETE

This function deletes an HFS file on the host. Access permissions for the resources requested must first be defined for you in your mainframe security system.

The XML service/scope/message names for a message to *delete* an HFS file are:

```
<service name=”file”>
<scope name=”service”>
<message name=”delete”>
```

These tags appear in both requests and replies. Case is significant.

FILE SERVICE DELETE Requests

Data structure details for the `<request>` tag appear in Exhibit 7-11. All subtags are required.

Exhibit 7-11. FILE SERVICE DELETE <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<pathName>	Required	1	String (1024), variable	Name of file to be deleted, prefixed by path from installation root.

FILE SERVICE DELETE Results

No `<result>` data structure is returned in response to an XML HFS file deletion request. However, the reply message does return a standard `<response>` data structure to indicate the success or failure of the request.

Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Rename a File — FILE SERVICE RENAME

This function renames an HFS file (or directory) on the host. Access permissions for the resources requested must first be defined for you in your mainframe security system.

The XML service/scope/message names for a message to *rename* an HFS file are:

```
<service name="file">
<scope name="service">
<message name="rename">
```

These tags appear in both requests and replies. Case is significant.

FILE SERVICE RENAME Requests

Data structure details for the `<request>` tag appear in Exhibit 7-12. All subtags are required.

Exhibit 7-12. FILE SERVICE RENAME <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<fromPathName>	Required	1	String (1024), variable	Old name of file to be renamed, prefixed by path from installation root.
<replace>	Required	1	String (1)	If the new file name is already in use, should the preexisting file be deleted so the rename can proceed? `Y` = Yes, replace any existing file of the same name with the renamed file. `N` = No, don't rename if another file with the same name already exists.
<toPathName>	Required	1	String (1024), variable	New name of file, prefixed by path from installation root.

FILE SERVICE RENAME Results

No `<result>` data structure is returned in response to an XML HFS file rename request. However, the reply message does return a standard `<response>` data structure to indicate the success or failure of the request.

Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Copy a File — FILE SERVICE COPY

This function makes an identical copy of an HFS file on the host. Access permissions for the resources requested must first be defined for you in your mainframe security system.

The XML service/scope/message names for a message to *copy* an HFS file are:

```
<service name="file">
<scope name="service">
<message name="copy">
```

These tags appear in both requests and replies. Case is significant.

FILE SERVICE COPY Requests

Data structure details for the `<request>` tag appear in Exhibit 7-13. All subtags are required.

Exhibit 7-13. FILE SERVICE COPY <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<compression>	Required	1	String (1)	Compress the target file during copy? `Y` = Yes, compress the target file. `N` = No, do not compress the target file. `E` = Expand the target file if the source file is compressed.
<copyTimeStamp>	Optional	0 - 1	String (1)	Copy all source timestamps to target file? `Y` = Yes, copy all timestamps from source file to target file `N` = No, update modification and access dates/times in target file at copy time (default).
<fromPathName>	Required	1	String (1024), variable	Name of source file to be copied, prefixed by path from installation root.
<replace>	Required	1	String (1)	If a file with the desired name already exists, should the preexisting file be deleted so the copy operation can proceed? `Y` = Yes, replace any existing file of the same name with the copied file. `N` = No, cancel copy if another file with the desired name already exists.

Exhibit 7-13. FILE SERVICE COPY <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<toPathName>	Required	1	String (1024), variable	Name of the target copied file, prefixed by path from installation root.

FILE SERVICE COPY Results

No `<result>` data structure is returned in response to an XML HFS file copy request. However, the reply message does return a standard `<response>` data structure to indicate the success or failure of the request.

Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Create a Link or Alias to a File — FILE SERVICE LINK

This function creates a symbolic link — also known as an alias or shortcut — to an HFS file. The resulting data object has a "file type code" of 5. Appropriate access permissions for the resources requested must first be defined for you in your mainframe security system.

The XML service/scope/message names for a message to *create a link* to an HFS file are:

```
<service name="file">
<scope name="service">
<message name="link">
```

These tags appear in both requests and replies. Case is significant.

FILE SERVICE LINK Requests

Data structure details for the `<request>` tag appear in Exhibit 7-14. All subtags are required.

Exhibit 7-14. FILE SERVICE LINK <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<linkPathName>	Required	1	String (1024), variable	Name of target file, prefixed by path from installation root. NOTE: This file or data object must already exist for link to be created.
<pathName>	Required	1	String (1024), variable	Name of symbolic link file that will point to target, prefixed by path from installation root. NOTE: This object will be created with file type = 5.

FILE SERVICE LINK Results

No `<result>` data structure is returned in response to an XML HFS file link request. However, the reply message does return a standard `<response>` data structure to indicate the success or failure of the request.

Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Change File Attributes — FILE SERVICE CHANGE

This function changes selected attributes, such as access permissions, for an HFS file (or directory) on the host. Appropriate access permissions for the resources requested must first be defined for you in your mainframe security system. You must be the file owner or an administrator to change access permissions.

The XML service/scope/message names for a message to *change attributes* of an HFS file are:

```
<service name=”file”>
<scope name=”service”>
<message name=”change”>
```

These tags appear in both requests and replies. Case is significant.

FILE SERVICE CHANGE Requests

Data structure details for the `<request>` tag appear in Exhibit 7-15. In addition to the mandatory `<pathName>` tag, at least one of the optional subtags is required to identify an attribute to be changed. Multiple attributes may be changed at once.

Exhibit 7-15. FILE SERVICE CHANGE <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<modTime>	Optional	0 - 1	Integer (14), fixed	New value for modification date and time. Format is `yyyyMMddHHmmss`.
<pathName>	Required	1	String (1024), variable	Name of file (or directory) to change, prefixed by path from installation root.

Exhibit 7-15. FILE SERVICE CHANGE <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<permissions>	Optional	0 - 1	Integer (3), fixed	New Unix access permissions for file, coded as 3-digit integer, where: 1st digit = owner permissions 2nd digit = group permissions 3rd digit = permissions for all others Each digit takes one of the following values: 7 - Read, write/rename/delete, execute 6 - Read, write/rename/delete 5 - Read, execute 4 - Read only 3 - Write/rename/delete, execute 2 - Write/rename/delete only 1 - Execute only 0 - No access permitted NOTE: For directories, "execute" permissions should be read as "open directory" or "view files in directory". NOTE: You must be the file owner or an administrator to change access permissions.

FILE SERVICE CHANGE Results

No `<result>` data structure is returned in response to an XML HFS file parameter change request. However, the reply message does return a standard `<response>` data structure to indicate the success or failure of the request.

Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Check Access to a File — FILE SERVICE ACCESS

This function checks for the existence of an HFS file (or directory) on the host, or for access permissions available to you (or another current user) if the file exists. Access permissions for the resources requested must first be defined for you (or the other user) in your mainframe security system.

USE CASES

- To determine whether a particular file (or directory) exists on a particular path, submit this request with `<accessMode>` set to `8`. If the file exists, the return code will be `00`. If it does not exist, the return code will be `08`.
- To determine whether the current user of the service has permission to execute the named file, submit this request with `<accessMode>` set to an odd number from `1` to `7`, using Unix numeric permission conventions. In other words, set it to `1` (execute

permissions only), `3` (write/rename/delete and execute permissions), `5` (read and execute permissions), or `7` (read, write/rename/delete, and execute permissions). If the user has the requested level of access to the file, the service will reply with a return code of `00`. If the user does not have the requested level of access, the return code will be `08`.

SYNTAX

The XML service/scope/message names for a message to *check access* to an HFS file are:

```
<service name="file">
<scope name="service">
<message name="access">
```

These tags appear in both requests and replies. Case is significant.

FILE SERVICE ACCESS Requests

Data structure details for the `<request>` tag appear in Exhibit 7-16. All subtags are required.

Exhibit 7-16. FILE SERVICE ACCESS <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<accessMode>	Required	1	String (1)	Type of file access to be tested. May test file existence or access permissions for current user. Values: 8 - File exists 7 - Read, write/rename/delete, execute 6 - Read, write/rename/delete 5 - Read, execute 4 - Read only 3 - Write/rename/delete, execute 2 - Write/rename/delete only 1 - Execute only 0 - No access permitted NOTE: For directories, "execute" permissions should be read as "open directory" or "view/list files in directory".
<pathName>	Required	1	String (1024), variable	Name of file (or directory) to be accessed, prefixed by path from installation root.

FILE SERVICE ACCESS Results

No `<result>` data structure is returned in response to an XML HFS file access check request. However, the reply message does return a standard `<response>` data structure to indicate the success or failure of the request.

Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Scan Files for Strings — FILE SERVICE SCAN

This function scans HFS files for requested strings.

The XML service/scope/message names for a message to *scan* files are:

```
<service name="file">
<scope name="service">
<message name="scan">
```

These tags appear in both requests and replies. Case is significant.

FILE SERVICE SCAN Requests

Data structure details for the `<request>` tag appear in Exhibit 7-17.

Exhibit 7-17. FILE SERVICE SCAN <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<and>	Optional	0 - 1	String (1)	To connect strings 1 and 2 (<scan1> and <scan2>). **Y** = connect strings **N** = do not connect strings
<caseFile>	Optional	0 - 1	String (1)	Case-sensitivity of file name. **Y** = file name is case-sensitive. **N** = file name is not case-sensitive.
<caseSensitive>	Optional	0 - 1	String (1)	Case-sensitivity of search strings. **Y** = search strings are case-sensitive. **N** = search strings are not case-sensitive.
<endOffset>	Optional	0 - 1	Integer	Displacement end.
<fileName>	Optional	0 - 1	String (256), variable	Name of file to be scanned.
<listType>	Optional	0 - 1	String (1)	Short or long result list. **S** = Short **L** = Long
<maxHits>	Optional	0 - 1	Integer	Maximum hits. 0 = unlimited.
<pathName>	Optional	0 - 1	String (1024), variable	Name of path to be scanned.
<recurse>	Optional	0 - 1	String (1)	Search subdirectories. **Y** = Search through all subdirectories. **N** = Search only the directory specified in <pathName>.
<scan1>	Optional	0 - 1	String (40), variable	Scan string 1.
<scan2>	Optional	0 - 1	String (40), variable	Scan string 2.
<startOffset>	Optional	0 - 1	Integer	Displacement end.

FILE SERVICE SCAN Results

Data structure details for the `<result>` tag appear in Exhibit 7-18. One to many `<data>` tags are returned.

A standard `<response>` data structure is returned after the final `<result>` tag to indicate success or failure of the request and the completion of the listing if successful. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Exhibit 7-18. FILE SERVICE SCAN <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<data>	Optional	1 - ∞	String, variable	Line of data. Includes file line, summary line, and matching line.

FILE CONVERSION SERVICES

The XML Services API provides file conversion services to import/export between z/OS PDS libraries and z/OS Unix HFS directories and files on the z/OS host. The following specific HFS file conversion functions are supported for general use:

- *Import a PDS Member into HFS —*
  FILE SERVICE IMPORT
- *Export an HFS File to a PDS Library —*
  FILE SERVICE EXPORT

Import a PDS Member into HFS — FILE SERVICE IMPORT

This function imports a native z/OS PDS (Partitioned Data Set) or PDSE (PDS Extended) library member into the z/OS Unix Hierarchical File System (HFS) as a standalone HFS file on the host. Access permissions for the resources requested must first be defined for you in your mainframe security system.

The XML service/scope/message names for a message to *import* a PDS member as an HFS file are:

```
<service name="file">
<scope name="service">
<message name="import">
```

These tags appear in both requests and replies. Case is significant.

FILE SERVICE IMPORT Requests

Data structure details for the `<request>` tag appear in Exhibit 7-19. All subtags are required.

Exhibit 7-19. FILE SERVICE IMPORT <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<compression>	Required	1	String (1)	Compress the target file during import? `Y` = Yes, compress the target HFS file. `N` = No, do not compress the target file. `E` = Expand the target HFS file if the source PDS member is compressed.
<mvsDsnLib>	Required	1	String (1024), variable	Fully qualified dataset name of source z/OS PDS/PDSE library. (Omit member name.)
<pathName>	Required	1	String (1024), variable	Name of the imported target HFS file, prefixed by path from installation root.
<pdsMember>	Required	1	String (8), variable	Name of PDS/PDSE library member to be imported.
<replace>	Required	1	String (1)	If an HFS file with the desired name already exists in the target location, should it be replaced with the imported file? `Y` = Yes, replace an existing HFS file of the same name with the imported file. `N` = No, cancel import if another HFS file with the desired name already exists.

FILE SERVICE IMPORT Results

No `<result>` data structure is returned in response to an XML HFS file import request. However, the reply message does return a standard `<response>` data structure to indicate the success or failure of the request.

Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Export an HFS File to a PDS Library — FILE SERVICE EXPORT

This function exports an HFS file from the z/OS Unix Hierarchical File System (HFS) to a native z/OS PDS (Partitioned Data Set) or PDSE (PDS Extended) library member on the host. Access permissions for the resources requested must first be defined for you in your mainframe security system.

The XML service/scope/message names for a message to *export* an HFS file as a PDS library member are:

```
<service name=”file”>
<scope name=”service”>
<message name=”export”>
```

These tags appear in both requests and replies. Case is significant.

FILE SERVICE EXPORT Requests

Data structure details for the `<request>` tag appear in Exhibit 7-20. All subtags are required.

Exhibit 7-20. FILE SERVICE EXPORT <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<compression>	Required	1	String (1)	Compress the target file during import? `Y` = Yes, compress the target HFS file. `N` = No, do not compress the target file. `E` = Expand the target HFS file if the source PDS member is compressed.
<mvsDsnLib>	Required	1	String (1024), variable	Fully qualified dataset name of target z/OS PDS/PDSE library. (Omit member name.)
<pathName>	Required	1	String (1024), variable	Name of the source HFS file to export, prefixed by path from installation root.
<pdsMember>	Required	1	String (8), variable	Name of target PDS/PDSE library member to receive exported HFS file.

FILE SERVICE EXPORT Results

No `<result>` data structure is returned in response to an XML HFS file export request. However, the reply message does return a standard `<response>` data structure to indicate the success or failure of the request.

Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

DATABASE MANAGEMENT 8

Serena XML supports database development and administration with both the IMS Option and the DB2 Option of ChangeMan ZMF. User tasks are grouped by database environment:

- ***IMS Development and Administration*** — Tasks that support IMS data binding or retrieve IMS database override settings at the package, application, or global level. The typical command for general use is *list*.
- ***DB2 Development and Administration*** — Tasks that support DB2 data binding or retrieve DB2 database configuration settings at the application or global level. The typical command for general use is *list*.

The syntax that identifies these functions generally appears in the `name` attribute of the `<service>` tag, as follows:

```
<service name="IMSOVRD">
<service name="DB2ADMIN">
```

In addition, some package-level IMS functions are performed by the low-level package service, not the IMS Override service. The `<service>` tag for these functions takes the `name` attribute of "`package`". The syntax that identifies these as IMS functions appears in the `name` attribute of the `<scope>` tag. For example:

```
<scope name="IMS_CRGN">
<scope name="IMS_ACB">
```

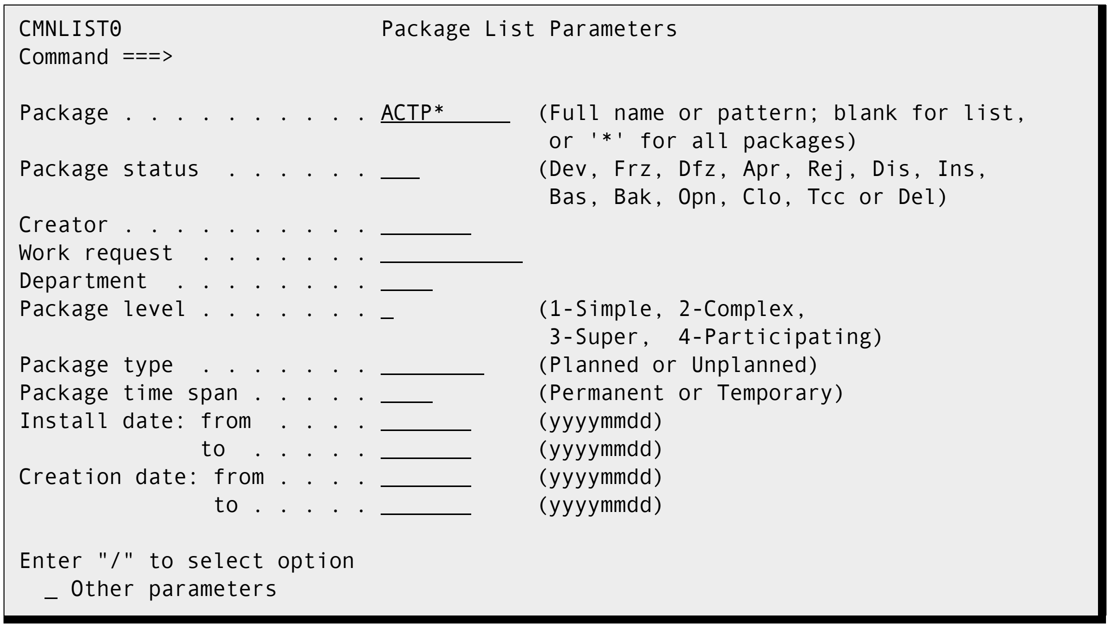
IMS DEVELOPMENT AND ADMINISTRATION

Serena XML supports the following IMS database tasks for general use:

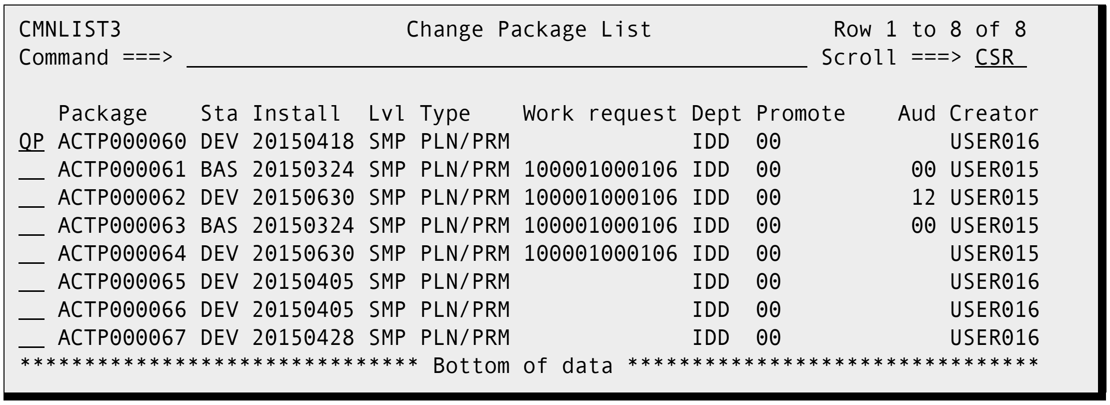
- *IMS Control Region Package Records - PACKAGE IMS_CRGN LIST*
- *Package IMS ACB List - PACKAGE IMS_ACB LIST*
- *IMS DBD Package Overrides - IMSOVRD PKG_DBD LIST*
- *IMS PSB Application Overrides - IMSOVRD APL_PSB LIST*
- *IMS DBD Global Overrides - IMSOVRD GBL_DBD LIST*
- *IMS PSB Global Overrides - IMSOVRD GBL_PSB LIST*

- *IMS PSB Package Overrides - IMSOVRD PKG_PSB LIST*
- *IMS DBD Application Overrides - IMSOVRD APL_DBD LIST*
- *IMS Control Region Application Defaults - IMSCRGN APL LIST*
- *IMS Control Region Global Defaults - IMSCRGN GBL LIST*

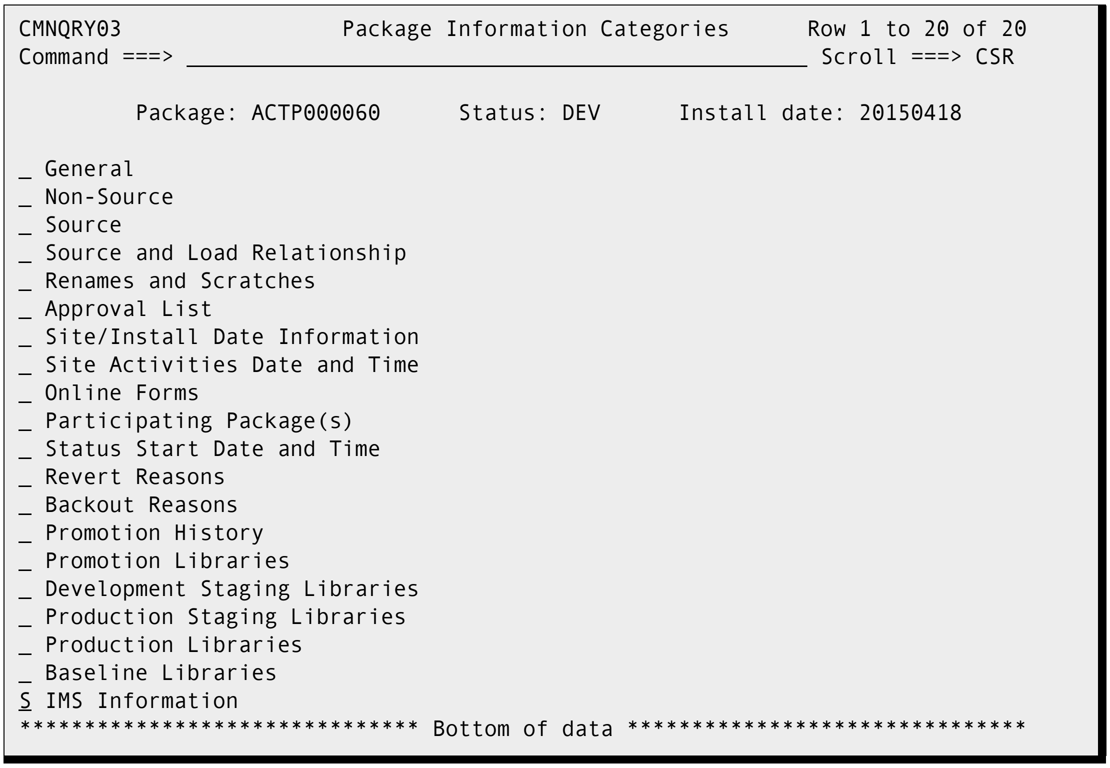
Much of the information presented in this chapter can also be obtained via the QP (Query Package) function, as shown in the following panels:



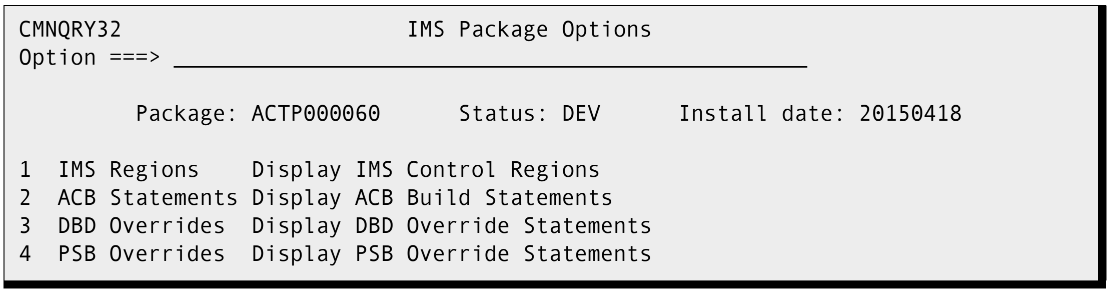
```
CMNLIST0                    Package List Parameters
Command ===>

Package . . . . . . . . . . ACTP*____    (Full name or pattern; blank for list,
                                          or '*' for all packages)
Package status  . . . . . . ___          (Dev, Frz, Dfz, Apr, Rej, Dis, Ins,
                                          Bas, Bak, Opn, Clo, Tcc or Del)
Creator . . . . . . . . . . _______
Work request  . . . . . . . ___________
Department  . . . . . . . . ____
Package level . . . . . . . _            (1-Simple, 2-Complex,
                                          3-Super,  4-Participating)
Package type  . . . . . . . ________     (Planned or Unplanned)
Package time span . . . . . ____         (Permanent or Temporary)
Install date: from  . . . . _______      (yyyymmdd)
              to  . . . . . _______      (yyyymmdd)
Creation date: from . . . . _______      (yyyymmdd)
               to . . . . . _______      (yyyymmdd)

Enter "/" to select option
  _ Other parameters
```



```
CMNLIST3                      Change Package List                Row 1 to 8 of 8
Command ===> ________________________________________________  Scroll ===> CSR_

   Package    Sta Install  Lvl Type    Work request Dept Promote    Aud Creator
QP ACTP000060 DEV 20150418 SMP PLN/PRM              IDD  00             USER016
__ ACTP000061 BAS 20150324 SMP PLN/PRM 100001000106 IDD  00          00 USER015
__ ACTP000062 DEV 20150630 SMP PLN/PRM 100001000106 IDD  00          12 USER015
__ ACTP000063 BAS 20150324 SMP PLN/PRM 100001000106 IDD  00          00 USER015
__ ACTP000064 DEV 20150630 SMP PLN/PRM 100001000106 IDD  00             USER015
__ ACTP000065 DEV 20150405 SMP PLN/PRM              IDD  00             USER016
__ ACTP000066 DEV 20150405 SMP PLN/PRM              IDD  00             USER016
__ ACTP000067 DEV 20150428 SMP PLN/PRM              IDD  00             USER016
******************************* Bottom of data ********************************
```



```
CMNQRY03                 Package Information Categories      Row 1 to 20 of 20
Command ===> ________________________________________________ Scroll ===> CSR

         Package: ACTP000060      Status: DEV      Install date: 20150418

_ General
_ Non-Source
_ Source
_ Source and Load Relationship
_ Renames and Scratches
_ Approval List
_ Site/Install Date Information
_ Site Activities Date and Time
_ Online Forms
_ Participating Package(s)
_ Status Start Date and Time
_ Revert Reasons
_ Backout Reasons
_ Promotion History
_ Promotion Libraries
_ Development Staging Libraries
_ Production Staging Libraries
_ Production Libraries
_ Baseline Libraries
S IMS Information
******************************* Bottom of data *******************************
```



```
CMNQRY32                      IMS Package Options
Option ===> _________________________________________________

         Package: ACTP000060      Status: DEV      Install date: 20150418

1  IMS Regions    Display IMS Control Regions
2  ACB Statements Display ACB Build Statements
3  DBD Overrides  Display DBD Override Statements
4  PSB Overrides  Display PSB Override Statements
```

IMS Control Region Package Records - PACKAGE IMS_CRGN LIST

This function lists IMS control region information associated with a change package. The desired package name is required in the request message. Each returned record includes the control region ID, associated library names, the ZMF site name where the appropriate IMS subsystem resides, and package-level IMS override settings for building install jobs at staging and for generating IMS ACBs, DBDs, PSBs, and MFS source files at promotion. If no package-level overrides have been defined, application defaults are returned.

The Serena XML service/scope/message tags and attributes for messages to *list* IMS control region records for a package are:

```
<service name="PACKAGE">
<scope name="IMS_CRGN">
<message name="LIST">
```

These tags appear in both requests and replies.

PACKAGE IMS_CRGN LIST — Requests

Serena XML supports three types of IMS control region lists for a package:

- ***Comprehensive List*** — Enter the name of the desired package in the `<package>` tag and omit all other tags to list all IMS control regions defined for the named package.
- ***Selected Site*** — Enter the name of the desired package in the `<package>` tag. Also identify the desired IMS site in either `<imsSiteName>` (using the ZMF remote site name where the IMS subsystem executes) or in `<imsLogicalSite>` (using the ZMF name of the logical change library associated with the corresponding baseline or promotion site). The function returns all IMS control regions for the named package and desired site.
- ***Selected Control Region*** — Enter the name of the desired package in the `<package>` tag and the 4-byte ID of the IMS control region of interest in the `<imsControlRegion>` tag to retrieve control region specifications for the named package and control region.

Example XML — PACKAGE IMS_CRGN LIST Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="IMS_CRGN">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>IMSQ000012</package>
   </request>
  </message>
 </scope>
</service>
```

Data structure details for the `<request>` data element appear in *Exhibit 8-1*.

Exhibit 8-1. PACKAGE IMS_CRGN LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<imsControlRegion>	Optional	0 -1	String (4), variable	IMS control region ID desired. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all IMS control regions for named package.
<imsLogicalSite>	Optional	0 -1	String (8), variable	ZMF baseline or promotion library name corresponding to IMS site name in `<imsSiteName>`. ***NOTE:*** Value may be **BASELINE** (all caps) or the name of any promotion library defined in ZMF. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all IMS change library site names for package.
<imsSiteName>	Optional	0 -1	String (8), variable	ZMF name of remote site where IMS subsystem is running. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all IMS subsystem sites for package.
<package>	Required	1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.

PACKAGE IMS_CRGN LIST — Replies

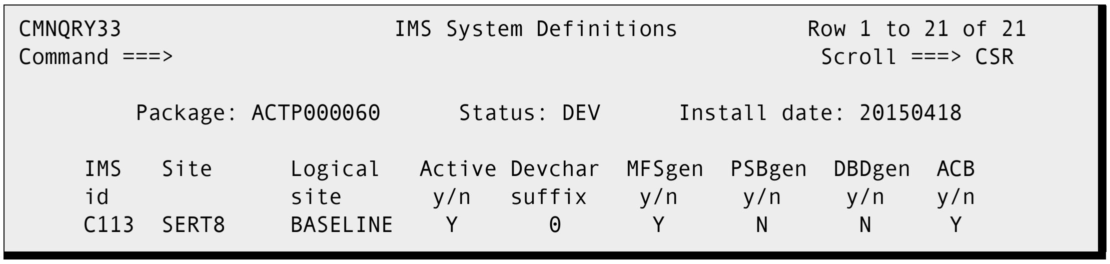
The reply message for a package-level IMS control region list returns zero to many `<result>` data elements. Each `<result>` tag contains information about one IMS control region associated with the package named in the request message.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Example XML — PACKAGE IMS_CRGN LIST Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="IMS_CRGN">
  <message name="LIST">
   <result>
    <package>IMSQ000012</package>
    <applName>IMSQ</applName>
    <packageId>000012</packageId>
    <isDbdAlwaysGenerated>N</isDbdAlwaysGenerated>
    <isPsbAlwaysGenerated>N</isPsbAlwaysGenerated>
    <isMfsAlwaysGenerated>Y</isMfsAlwaysGenerated>
    <isAcbAlwaysCreatedForPcbs>Y</isAcbAlwaysCreatedForPcbs>
    <isImsGlobalActivationEnabled>Y</isImsGlobalActivationEnabled>
    <imsControlRegion>C113</imsControlRegion>
    <imsSiteName>SERT8</imsSiteName>
    <imsLogicalSite>BASELINE</imsLogicalSite>
    <imsDevCharSuffix>0</imsDevCharSuffix>
    <imsResLib>SYS2.IMS910.SDFSRESL</imsResLib>
    <imsMacLib>SYS2.IMS910.SDFSMAC</imsMacLib>
    <imsModStatLib>CMNTP.SERT8.IMSC113.MODSTAT</imsModStatLib>
    <imsGenMacroStageLib>USER24.SETQUERY.WORKLOAD</imsGenMacroStageLib>
    <imsGenMacroComponent>STAGEII</imsGenMacroComponent>
    <imsPsbLib>CMNTP.SERT8.IMSC113.PSBLIB</imsPsbLib>
    <imsDbdLib>CMNTP.SERT8.IMSC113.DBDLIB</imsDbdLib>
    <imsAcbLib>CMNTP.SERT8.IMSC113.ACBLIB</imsAcbLib>
    <imsFmtLib>CMNTP.SERT8.IMSC113.FORMAT</imsFmtLib>
    <imsRefLib>CMNTP.SERT8.IMSC113.REFERAL</imsRefLib>
    <imsBackupModelLib>BACKUPMODEL</imsBackupModelLib>
   </result>
   <response>
    <statusMessage>CMN8600I - The package IMS control region list is complete.</
statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8600</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

If option 1 from above PANEL CMNQRY32 is selected, information similar to that returned via a PACKAGE IMSCRGN LIST request is shown in panel CMNQRY33:



```
CMNQRY33                    IMS System Definitions         Row 1 to 21 of 21
Command ===>                                                Scroll ===> CSR

         Package: ACTP000060      Status: DEV      Install date: 20150418

     IMS    Site      Logical   Active Devchar  MFSgen  PSBgen  DBDgen  ACB
     id               site       y/n   suffix    y/n     y/n     y/n    y/n
     C113   SERT8     BASELINE    Y       0       Y       N       N      Y
```

Data structure details for the `<result>` tag appear in *Exhibit 8-2*.

Exhibit 8-2. PACKAGE IMS_CRGN LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), fixed	ZMF application name. Same as first 4 bytes of package name.
<imsAcbLib>	Optional	0 - 1	String (44), variable	Name of dataset containing all prebuilt ACBs used by IMS control region in `<imsControlRegion>`.
<imsBackupModelLib>	Optional	0 - 1	String (25), variable	High-level node pattern for backups of IMS system libraries during ZMF promotion and install.
<imsControlRegion>	Optional	0 - 1	String (4), variable	SSID of IMS control region.
<imsDbdLib>	Optional	0 - 1	String (44), variable	Name of dataset containing all DBDs defined to IMS.
<imsDevCharSuffix>	Optional	0 - 1	String (1)	1-byte suffix appended to name of IMS module that generates MFS source code. Sets 3270 or SLU2 device characteristics.
<imsFmtLib>	Optional	0 - 1	String (44), variable	Data set with all DIF/DOF and MID/MOD control blocks used by control region in `<imsControlRegion>`.
<imsGenMacroComponent>	Optional	0 - 1	String (8), variable	Name of member containing source code that generated the IMS control region, databases, programs, & terminals. Member of `<imsGenMacroStageLib>`.
<imsGenMacroStageLib>	Optional	0 - 1	String (44), variable	Name of IMS system generation dataset containing the system generation member name.
<imsLogicalSite>	Optional	0 - 1	String (8), variable	IMS logical site.

Exhibit 8-2. PACKAGE IMS_CRGN LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<imsMacLib>	Optional	0 - 1	String (44), variable	Name of dataset with all IMS macros for system, PSB, DBD, ACB, & MFS generation.
<imsModStatLib>	Optional	0 - 1	String (44), variable	Name of sequential dataset containing active library information for MODBLKS, IMSACB, and FORMAT.
<imsPsbLib>	Optional	0 - 1	String (44), variable	Name of dataset containing all PSBs defined to IMS.
<imsRefLib>	Optional	0 - 1	String (44), variable	Name of referral library for MFS generation. Contains intermediate text block stored between steps in generation process.
<imsResLib>	Optional	0 - 1	String (44), variable	Name of APF-authorized IMS system library.
<imsSiteName>	Optional	0 - 1	String (8), variable	ZMF ID for IMS site.
<isAcbAlwaysCreatedForPcbs>	Optional	0 - 1	String (1)	**Y** = Yes, ACB always generated for PSBs. **N** = No, generate ACB for PSBs only if needed. ***NOTE:*** ACB is always generated for DBDs.
<isDbdAlwaysGenerated>	Optional	0 -1	String (1)	**Y** = Yes, DBD always generated. **N** = No, generate DBD only if override specified.
<isImsGlobalActivationEnabled>	Optional	0 -1	String (1)	**Y** = Yes, global IMS package activation enabled **N** = No, global IMS package activation disabled
<isMfsAlwaysGenerated>	Optional	0 - 1	String (1)	**Y** = Yes, MFS always generated **N** = No, generate MFS only if `<imsDevCharSuffix>` value differs from IMS setup.
<isPsbAlwaysGenerated>	Optional	0 -1	String (1)	**Y** = Yes, PSB always generated. **N** = No, generate PSB only if override specified.
<package>	Optional	1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), fixed	ZMF package ID number. Same as last 6 bytes of package name.

Package IMS ACB List - PACKAGE IMS_ACB LIST

This function lists the IMS access method control block (ACB) build statements needed by a particular change package. The desired package name is required in the request message. Each returned record includes generation specifications for one ACB build statement associated with a specific IMS-enabled promotion site and a specific ACB source component. If no ACB build statements are defined for the package, no results are returned.

The Serena XML service/scope/message tags and attributes for messages to *list* IMS ACB build statements for a package are:

```
<service name="PACKAGE">
<scope name="IMS_ACB">
<message name="LIST">
```

These tags appear in both requests and replies.

PACKAGE IMS_ACB LIST — Request

Serena XML supports four types of IMS ACB record lists for a package:

- ***Comprehensive List*** — Enter the name of the desired package in the `<package>` tag and omit all other tags to list all IMS ACB records defined for the named package.
- ***Selected Site*** — Enter the name of the desired package in the `<package>` tag. Also identify the desired IMS site in either the `<imsSiteName>` tag (using the ZMF remote site name where the IMS subsystem executes) or in the `<imsLogicalSite>` tag (using the ZMF name of the logical change library associated with the corresponding baseline or promotion site). The function returns all IMS ACB records for the named package and desired site.
- ***Selected Control Region*** — Enter the name of the desired package in the `<package>` tag and the 4-byte ID of the IMS control region of interest in the `<imsSiteId>` tag to retrieve IMS ACB records for the named package and control region.
- ***Selected Source Component*** — Enter the name of the desired package in the `<package>` tag. Also enter the name of the desired ACB source component for the package from the relevant DBD library or PSB library in the `<component>` tag. If known, enter the source component library type in the `<componentType>` tag. The function returns a list of ACB records, including package-specific ACB target component names, associated with the identified ACB source component.

Example XML — PACKAGE IMS_ACB LIST Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="IMS_ACB">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
```

```
    <package>IMSQ000012</package>
   </request>
  </message>
 </scope>
</service>
```

Data structure details for the `<request>` data element appear in *Exhibit 8-3*.

Exhibit 8-3. PACKAGE IMS_ACB LIST <request>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<component>	Optional	0 -1	String (256), variable	Name of ACB source component in DBD library or PSB library used when generating a target ACB component. NOTE: Typically 8 bytes max.
<componentType>	Optional	0 -1	String (3), variable	Library type for ACB source component named in `<component>` tag.
<imsLogicalSite>	Optional	0 -1	String (8), variable	ZMF change library name corresponding to ZMF site name in `<imsSiteName>` tag. ***NOTE:*** Value may be **BASELINE** or the nickname of any promotion library defined in ZMF.
<imsSiteId>	Optional	0 -1	String (4), variable	ZMF ID of IMS control region desired. ***NOTE:*** Every IMS control region is associated with a site.
<imsSiteName>	Optional	0 -1	String (8), variable	ZMF name of site where IMS subsystem executes.
<package>	Required	1	String (10), variable	Fixed-format ZMF package name. ***NOTE:*** May be masked using asterisk (*) wildcard character.
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.

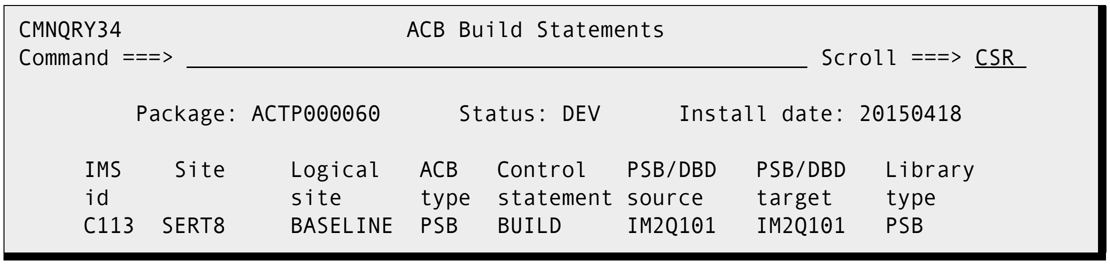
PACKAGE IMS_ACB LIST — Reply

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` contains generation specifications for the IMS ACB build statements needed by a package. The standard `<response>` data element follows to indicate the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it follows the final `<result>` data structure in the reply, the `<response>` tag also serves as and end-of-list marker.

Example XML — PACKAGE IMS_ACB LIST Reply

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="IMS_ACB">
  <message name="LIST">
   <result>
    <package>IMSQ000012</package>
    <applName>IMSQ</applName>
    <packageId>000012</packageId>
    <isAcbGenerated>N</isAcbGenerated>
    <acbGenStatementType>B</acbGenStatementType>
    <acbStatementType>P</acbStatementType>
    <imsSiteId>C113</imsSiteId>
    <imsSiteName>SERT8</imsSiteName>
    <imsLogicalSite>BASELINE</imsLogicalSite>
    <targetComponent>IM2Q101</targetComponent>
    <targetComponentType>PSL</targetComponentType>
    <component>IM2Q101</component>
    <componentType>PSB</componentType>
   </result>
   <response>
    <statusMessage>CMN8600I - The package ACB Statement list is complete.</
statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8600</statusReasonCode>
   </response>
  </message>
 </service>
```

If option 2 from above PANEL CMNQRY32 is selected, information similar to that returned via a PACKAGE IMSACB LIST request is shown in panel CMNQRY34 below:



```
CMNQRY34                     ACB Build Statements
Command ===> ______________________________________________ Scroll ===> CSR

         Package: ACTP000060      Status: DEV      Install date: 20150418

     IMS    Site     Logical   ACB   Control   PSB/DBD   PSB/DBD   Library
     id              site      type  statement source    target    type
     C113  SERT8     BASELINE  PSB   BUILD     IM2Q101   IM2Q101   PSB
```

Data structure details for the `<result>` tag appear in *Exhibit 8-4*.

Exhibit 8-4. PACKAGE IMS_ACB LIST <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<acbGenStatementType>	Optional	0 - 1	String (1)	Code for type of ACB generation statement. Values: **B** = ACB Build Statement **D** = ACB Delete Statement
<acbStatementType>	Optional	0 - 1	String (1)	Code for type of ACB statement. Values: **P** = ACB PSB Statement **D** = ACB DBD Statement
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<component>	Optional	0 -1	String (256), variable	Name of IMS source component for package ACB generation. Component located in IMS PSB library or DBD library defined in control region identified in `<imsSiteId>`. NOTE: Typically 8 bytes max.
<componentType>	Optional	0 -1	String (3), variable	Library type for component named in `<component>` tag.
<imsLogicalSite>	Optional	0 -1	String (8), variable	ZMF change library name corresponding to site name in `<imsSiteName>` tag. ***NOTE:*** Value may be **BASELINE** or the nickname of any promotion library defined in ZMF.
<imsSiteId>	Optional	0 -1	String (4), variable	ZMF ID of IMS control region associated with `<imsSiteName>`.
<imsSiteName>	Optional	0 -1	String (8), variable	ZMF name of site where IMS subsystem executes.
<isAcbGenerated>	Optional	0 - 1	String (1)	**Y** = Yes, ACB always generated for both PSBs & DBDs **N** = No, generate ACB for PSBs only if needed. (ACB for DBDs always generated.)
<package>	Required	1	String (10), variable	Fixed-format ZMF package name.

Exhibit 8-4. PACKAGE IMS_ACB LIST <result> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<targetComponent>	Optional	0 -1	String (256), variable	Name of ACB target component generated at package promotion. NOTE: Typically 8 bytes max.
<targetComponentType>	Optional	0 -1	String (3), variable	Library type for ACB target component named in `<targetComponent>` tag.

IMS DBD Package Overrides - IMSOVRD PKG_DBD LIST

This function lists package-level overrides to the IMS database description (DBD) records associated with a specific component in the IMS DBD library. Values returned include the DBD source component, IMS control statement type, the original DBD control statement contents, and the override content for that DBD statement. If no DBD overrides are defined for a package, no results are returned.

The Serena XML service/scope/message tags and attributes for messages to *list* IMS DBD overrides for a package are:

```
<service name="IMSOVRD">
<scope name="PKG_DBD">
<message name="LIST">
```

These tags appear in both requests and replies.

IMSOVRD PKG_DBD LIST — Requests

Serena XML supports five types of DBD override lists for a package:

- ***Comprehensive List*** — Enter the name of the desired package in the `<package>` tag and omit all other tags to list all IMS DBD overrides defined for the named package.
- ***Selected Site*** — Enter the name of the desired package in the `<package>` tag. Also identify the desired IMS site in either the `<imsSiteName>` tag (using the ZMF remote site name where the IMS subsystem executes) or in the `<imsLogicalSite>` tag (using the ZMF nickname of the change library associated with the corresponding baseline or promotion site). The function returns all IMS DBD override statements for the named package and desired site.
- ***Selected Control Region*** — Enter the name of the desired package in the `<package>` tag and the 4-byte ID of the IMS control region of interest in the `<imsSiteId>` tag to retrieve IMS DBD overrides for the named package and control region.

- ***Selected Source Component*** — Enter the name of the desired package in the `<package>` tag. Also enter the name of the desired DBD library source component to override in the `<component>` tag. If known, enter the source component library type in the `<componentType>` tag. The function returns a list of DBD control statement overrides for the named package and source component.
- ***Selected Control Statement Type*** — Enter the name of the desired package in the `<package>` tag and the IMS keyword for the desired control statement type in the `<controlStatement>` tag. The function returns all DBD overrides of the desired type.

Example XML — IMSOVRD PKG_DBD LIST Request

```
<?xml version="1.0"?>
<service name="IMSOVRD">
 <scope name="PKG_DBD">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>IMSQ000012</package>
   </request>
  </message>
 </scope>
</service>
```

Data structure details for the `<request>` data element appear in *Exhibit 8-5*. Note that the identical `<request>` syntax is used with both DBD and PSB override lists.

Exhibit 8-5. IMSOVRD PKG_DBD LIST <request>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<component>	Optional	0 -1	String (256), variable	Name of source component in IMS DBD (or PSB) library. ***NOTE:*** Use asterisk (*) wildcard or omit tag include all IMS source components referenced by a package.

Exhibit 8-5. IMSOVRD PKG_DBD LIST <request> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<componentType>	Optional	0 -1	String (3), variable	Library type for component named in `<component>` tag. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all IMS component library types.
<controlStatement>	Optional	0 -1	String (8), variable	IMS keyword for type of IMS control statement being overridden. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all statement types.
<imsLogicalSite>	Optional	0 -1	String (8), variable	ZMF change library nickname for site in `<imsSiteName>` tag. ***NOTE:*** Value may be **BASELINE** or the name of any promotion library defined in ZMF. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all change libraries for named package.
<imsSiteId>	Optional	0 -1	String (4), variable	IMS control region ID associated with site in `<imsSiteName>`. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all IMS control regions for package.
<imsSiteName>	Optional	0 -1	String (8), variable	ZMF name of site where IMS subsystem executes. ***NOTE:*** Use asterisk (*) wildcard or omit tag to list all ZMF sites for package.
<package>	Required	1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.

IMSOVRD PKG_DBD LIST — Replies

The reply message for a package-level IMS DBD override list returns zero to many `<result>` data elements. Each `<result>` contains package-level override information for one IMS database description (DBD) control statement associated with one IMS source component in an associated DBD library. If no DBD overrides are defined for the package, no `<result>` tags are returned in the reply.

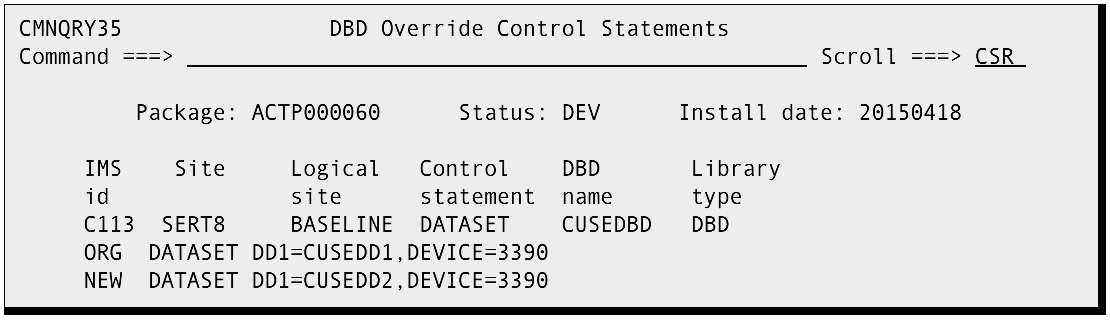
The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of

00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Example XML — IMSOVRD PKG_DBD LIST Reply

```
<?xml version="1.0"?>
<service name="IMSOVRD">
 <scope name="PKG_DBD">
  <message name="LIST">
   <result>
    <package>IMSQ000012</package>
    <applName>IMSQ</applName>
    <packageId>000012</packageId>
    <imsSiteId>C113</imsSiteId>
    <imsSiteName>SERT8</imsSiteName>
    <imsLogicalSite>BASELINE</imsLogicalSite>
    <component>CUSEDBD</component>
    <componentType>DBD</componentType>
    <controlStatement>DATASET</controlStatement>
    <overrideStatement>DATASET DD1=CUSEDD2,DEVICE=3390</overrideStatement>
    <originalStatement>DATASET DD1=CUSEDD1,DEVICE=3390</originalStatement>
   </result>
   <response>
    <statusMessage>CMN8600I - The package DBD overrides list is complete.</
statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8600</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Some of the same information is presented in panel CMNQRY35:



```
CMNQRY35                DBD Override Control Statements
Command ===> ________________________________________________ Scroll ===> CSR

        Package: ACTP000060      Status: DEV      Install date: 20150418

     IMS    Site     Logical   Control    DBD       Library
     id              site      statement  name      type
     C113  SERT8     BASELINE  DATASET    CUSEDBD   DBD
     ORG  DATASET DD1=CUSEDD1,DEVICE=3390
     NEW  DATASET DD1=CUSEDD2,DEVICE=3390
```

Data structure details for the `<result>` tag appear in *Exhibit 8-6*. Note that the identical `<result>` syntax is used with both DBD and PSB override lists.

Exhibit 8-6. IMSOVRD PKG_DBD LIST <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), variable	ZMF application name. Same as first 4 bytes of package name.
<component>	Optional	0 -1	String (256), variable	Name of IMS source component to override in IMS DBD (or PSB) library. NOTE: Typically 8 bytes max.
<componentType>	Optional	0 -1	String (3), variable	Library type for component named in `<component>` tag.
<controlStatement>	Optional	0 -1	String (8), variable	IMS keyword for type of IMS control statement being overridden.
<imsLogicalSite>	Optional	0 -1	String (8), variable	ZMF change library nickname for site in `<imsSiteName>` tag. ***NOTE:*** Value may be **BASELINE** or the name of any promotion library defined in ZMF.
<imsSiteId>	Optional	0 -1	String (4), variable	IMS control region associated with site in `<imsSiteName>`.
<imsSiteName>	Optional	0 -1	String (8), variable	ZMF name of site where IMS subsystem executes.
<originalStatement>	Optional	0 -1	String (64), variable	Contents of the original DBD (or PSB) control statement.
<overrideStatement>	Optional	0 -1	String (64), variable	Contents of the DBD (or PSB) override statement for package.
<package>	Optional	0 -1	String (10), variable	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name.

IMS PSB Package Overrides - IMSOVRD PKG_PSB LIST

This function lists package-level overrides to the IMS program specification block (PSB) control statements associated with a specific source component in the IMS PDB library. Values returned include the PSB source component, IMS control statement type, the original PSB control statement contents, and the override content for that PSB statement. If no PSB overrides are defined for a package, no results are returned.

The Serena XML service/scope/message tags and attributes for messages to *list* IMS PSB overrides for a package are:

```
<service name="IMSOVRD">
<scope name="PKG_PSB">
<message name="LIST">
```

These tags appear in both requests and replies.

IMSOVRD PKG_PSB LIST — Requests

Serena XML supports five types of PSB override lists for a package:

- ***Comprehensive List*** — Enter the name of the desired package in the `<package>` tag and omit all other tags to list all IMS PSB overrides defined for the named package.
- ***Selected Site*** — Enter the name of the desired package in the `<package>` tag. Also identify the desired IMS site in either the `<imsSiteName>` tag (using the ZMF remote site name where the IMS subsystem executes) or in the `<imsLogicalSite>` tag (using the ZMF nickname of the change library associated with the corresponding baseline or promotion site). The function returns all IMS PSB override statements for the named package and desired site.
- ***Selected Control Region*** — Enter the name of the desired package in the `<package>` tag and the 4-byte ID of the IMS control region of interest in the `<imsSiteId>` tag to retrieve IMS PSB overrides for the named package and control region.
- ***Selected Source Component*** — Enter the name of the desired package in the `<package>` tag. Also enter the name of the desired PSB library source component to override in the `<component>` tag. If known, enter the source component library type in the `<componentType>` tag. The function returns a list of PSB control statement overrides for the named package and source component.
- ***Selected Control Statement Type*** — Enter the name of the desired package in the `<package>` tag and the IMS keyword for the desired control statement type in the `<controlStatement>` tag. The function returns all PSB overrides of the desired type.

The Serena XML syntax to request a list of package-level IMS PSB overrides is identical to that for DBD overrides. Data structure details for the `<request>` data element appear in *Exhibit 8-5* of the previous section.

Example XML — IMSOVRD PKG_PSB LIST Request

```
<?xml version="1.0"?>
<service name="IMSOVRD">
 <scope name="PKG_PSB">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>IMSQ000012</package>
   </request>
  </message>
 </scope>
</service>
```

IMSOVRD PKG_PSB LIST — Replies

The reply message for a package-level IMS PSB override list returns zero to many `<result>` data elements. Each `<result>` contains package-level override information for one IMS program specification block (PSB) control statement associated with one IMS source component in an associated PSB library. If no PSB overrides are defined for the package, no `<result>` tags are returned in the reply.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

The Serena XML reply syntax for a package-level IMS PSB override list is identical to that for a package-level DBD override list. Details for the `<result>` tag appear in *Exhibit 8-6* of the previous section.

Example XML — IMSOVRD PKG_PSB LIST Reply

```
<?xml version="1.0"?>
<service name="IMSOVRD">
 <scope name="PKG_PSB">
  <message name="LIST">
   <result>
    <package>IMSQ000012</package>
    <applName>IMSQ</applName>
    <packageId>000012</packageId>
    <imsSiteId>C113</imsSiteId>
    <imsSiteName>SERT8</imsSiteName>
    <imsLogicalSite>BASELINE</imsLogicalSite>
    <component>IM2Q101</component>
    <componentType>PSB</componentType>
    <controlStatement>PSBGEN</controlStatement>
    <overrideStatement>PSBGEN PSBNAME=IM2Q101,LANG=COBOL,CMPAT=YES</
overrideStatement>
    <originalStatement>PSBGEN PSBNAME=IM2Q101,LANG=ASSEM,CMPAT=YES</
originalStatement>
   </result>
   <response>
    <statusMessage>CMN8600I - The package PSB overrides list is complete.</
statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8600</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

IMS DBD Application Overrides - IMSOVRD APL_DBD LIST

This function lists application-level overrides to the IMS database description (DBD) records in a specific source component in the IMS DBD library. Values returned include the DBD source component, IMS control statement type, the original DBD control statement contents, and the override content for that DBD statement. If no DBD overrides are defined for the application, no results are returned.

To retrieve APPLication level overrides and control region information via the ChangeMan ZMF started task, you can go to the application administration panels. In order to access these menus, you will need UPDATE authority to the APPLication's security entity, as well as local admin authority.

IMSOVRD APL_DBD LIST

The Serena XML service/scope/message tags and attributes for messages to *list* IMS DBD overrides for an application are:

```
<service name="IMSOVRD">
<scope name="APL_DBD">
<message name="LIST">
```

These tags appear in both requests and replies.

IMSOVRD APL_DBD LIST — Requests

Serena XML supports five types of DBD override lists for an application:

- ***Comprehensive List*** — Enter the name of the desired application in the `<applName>` tag and omit all other tags to list all IMS DBD overrides defined for that application.
- ***Selected Site*** — Enter the name of the desired application in the `<applName>` tag. Also identify the desired IMS site in either the `<imsSiteName>` tag (using the ZMF remote site name where the IMS subsystem executes) or in the `<imsLogicalSite>` tag (using the ZMF nickname of the change library associated with the corresponding baseline or promotion site). The function returns all IMS DBD override statements for the named application and site.
- ***Selected Control Region*** — Enter the name of the desired application in the `<applName>` tag and the 4-byte ID of the IMS control region of interest in the `<imsSiteId>` tag. The function returns IMS DBD overrides for the named application and control region.
- ***Selected Source Component*** — Enter the name of the desired application in the `<applName>` tag. Also enter the name of the desired DBD library source component to override in the `<component>` tag. If known, enter the source component library type in the `<componentType>` tag. The function returns a list of DBD control statement overrides for the named application and source component.
- ***Selected Control Statement Type*** — Enter the name of the desired application in the `<applName>` tag and the IMS keyword for the desired control statement type in the `<controlStatement>` tag. The function returns all DBD overrides of the desired type.

Example XML — IMSOVRD APL_DBD LIST Request

```
<?xml version="1.0"?>
<service name="IMSOVRD">
 <scope name="APL_DBD">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <applName>IMSQ</applName>
   </request>
  </message>
 </scope>
</service>
```

Data structure details for the `<request>` data element appear in *Exhibit 8-7*. Note that the identical `<request>` syntax is used with both DBD and PSB record lists.

Exhibit 8-7. IMSOVRD APL_DBD LIST <request>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	1	String (4), variable	ZMF application name. ***NOTE:*** OK to omit trailing blanks.
<component>	Optional	0 -1	String (256), variable	Name of source component to override in IMS DBD (or PSB) library. ***NOTE:*** Use asterisk (*) wildcard or omit tag include all IMS source components referenced by an application. NOTE: Typically 8 bytes max.
<componentType>	Optional	0 -1	String (3), variable	Library type for component named in `<component>` tag. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all IMS component library types.
<controlStatement>	Optional	0 -1	String (8), variable	IMS keyword for type of IMS control statement being overridden. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all statement types.
<imsLogicalSite>	Optional	0 -1	String (8), variable	ZMF change library nickname for site in `<imsSiteName>` tag. ***NOTE:*** Value may be **BASELINE** or the name of any promotion library defined in ZMF. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all change libraries for named application.

Exhibit 8-7. IMSOVRD APL_DBD LIST <request> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<imsSiteId>	Optional	0 -1	String (4), variable	IMS control region ID associated with site in `<imsSiteName>`. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all IMS control regions for application.
<imsSiteName>	Optional	0 -1	String (8), variable	ZMF name of site where IMS subsystem executes. ***NOTE:*** Use asterisk (*) wildcard or omit tag to list all ZMF sites for application.

IMSOVRD APL_DBD LIST — Replies

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` contains application-level override information for one IMS database description (DBD) control statement in one IMS source component located in the IMS DBD library.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Example XML — IMSOVRD APL_DBD LIST Reply

```
<?xml version="1.0"?>
<service name="IMSOVRD">
 <scope name="APL_DBD">
  <message name="LIST">
   <result>
    <applName>IMSQ</applName>
    <imsSiteId>C113</imsSiteId>
    <imsSiteName>SERT8</imsSiteName>
    <imsLogicalSite>BASELINE</imsLogicalSite>
    <component>CUSEDBD</component>
    <componentType>DBD</componentType>
    <controlStatement>DATASET</controlStatement>
    <overrideStatement>DATASET DD1=CUSEDD2,DEVICE=3390</overrideStatement>
    <originalStatement>DATASET DD1=CUSEDD1,DEVICE=3390</originalStatement>
   </result>
   <response>
    <statusMessage>CMN8600I - The IMS DBD Override list is complete.</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8600</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Data structure details for the `<result>` tag appear in *Exhibit 8-8*. Note that the identical `<result>` syntax is used with both DBD and PSB override lists.

Exhibit 8-8. IMSOVRD APL_DBD LIST <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), variable	ZMF application name.
<component>	Optional	0 -1	String (256), variable	Name of IMS source component to override in IMS DBD (or PSB) library. NOTE: Typically 8 bytes max.
<componentType>	Optional	0 -1	String (3), variable	Library type for component named in `<component>` tag.
<controlStatement>	Optional	0 -1	String (8), variable	IMS keyword for type of IMS control statement being overridden.
<imsLogicalSite>	Optional	0 -1	String (8), variable	ZMF change library nickname for site in `<imsSiteName>` tag. ***NOTE:*** Value may be **BASELINE** or the name of any promotion library defined in ZMF.
<imsSiteId>	Optional	0 -1	String (4), variable	IMS control region associated with site in `<imsSiteName>`.
<imsSiteName>	Optional	0 -1	String (8), variable	ZMF name of site where IMS subsystem executes.
<originalStatement>	Optional	0 -1	String (64), variable	Contents of the original DBD (or PSB) control statement.
<overrideStatement>	Optional	0 -1	String (64), variable	Contents of the DBD (or PSB) override statement for package.

IMS PSB Application Overrides - IMSOVRD APL_PSB LIST

This function lists application-level overrides to the IMS program specification block (PSB) control statements associated with a specific source component in the IMS PSB library. Values returned include the PSB source component, IMS control statement type, the original PSB control statement contents, and the override content for that PSB statement. If no PSB overrides are defined for the application, no results are returned.

The Serena XML service/scope/message tags and attributes for messages to *list* IMS PSB overrides for an application are:

```
<service name="IMSOVRD">
<scope name="APL_PSB">
<message name="LIST">
```

These tags appear in both requests and replies.

IMSOVRD APL_PSB LIST — Requests

Serena XML supports five types of PSB override lists for an application:

- ***Comprehensive List*** — Enter the name of the desired application in the `<applName>` tag and omit all other tags to list all IMS PSB overrides defined for the application.
- ***Selected Site*** — Enter the name of the desired application in the `<applName>` tag. Also identify the desired IMS site in either the `<imsSiteName>` tag (using the ZMF remote site name where the IMS subsystem executes) or in the `<imsLogicalSite>` tag (using the ZMF nickname of the change library associated with the corresponding baseline or promotion site). The function returns all IMS PSB override statements for the named application and site.
- ***Selected Control Region*** — Enter the name of the desired application in the `<applName>` tag and the 4-byte ID of the IMS control region of interest in the `<imsSiteId>` tag. The function returns IMS PSB control statement overrides for the named application and control region.
- ***Selected Source Component*** — Enter the name of the desired application in the `<applName>` tag. Also enter the name of the desired PSB library source component to override in the `<component>` tag. If known, enter the source component library type in the `<componentType>` tag. The function returns a list of PSB control statement overrides for the named application and source component.
- ***Selected Control Statement Type*** — Enter the name of the desired application in the `<applName>` tag and the IMS keyword for the desired control statement type in the `<controlStatement>` tag. The function returns all PSB overrides of the desired type.

The Serena XML syntax for a request to list application-level IMS PSB overrides is identical to that for DBD overrides. Data structure details for the `<request>` data element appear in *Exhibit 8-7* of the previous section.

IMSOVRD APL_PSB LIST — Replies

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` contains application-level overrides for one IMS program specification block (PSB) control statement in a specific PSB library source component. If no PSB overrides are defined for an application, no `<result>` tags are returned in the reply.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

The Serena XML reply syntax for an application-level IMS PSB override list is identical to that for application-level DBD overrides. Details for the `<result>` tag appear in *Exhibit 8-8* of the previous section.

IMS DBD Global Overrides - IMSOVRD GBL_DBD LIST

To retrieve GLOBAL overrides and control region information via the ChangeMan ZMF started task, you can go to the GLOBAL administration panels. In order to access these menus, you will need GLOBAL admin authority.

IMSOVRD GBL_DBD LIST

This function lists ChangeMan ZMF global overrides to the IMS database description (DBD) records in a specific source component in the IMS DBD library. Values returned include the DBD source component, IMS control statement type, the original DBD control statement contents, and the override content for that DBD statement. If no DBD overrides are defined for at the global level, no results are returned.

The Serena XML service/scope/message tags and attributes for messages to *list* global IMS DBD overrides are:

```
<service name="IMSOVRD">
<scope name="GBL_DBD">
<message name="LIST">
```

These tags appear in both requests and replies.

IMSOVRD GBL_DBD LIST — Requests

Serena XML supports five types of DBD override lists at the global level:

- ***Comprehensive List*** — Submit an empty `<request>` data element (that is, one that contains no subtags) in the XML request message. The `<request>` tag itself is required in the message to distinguish a request from a reply. All globally defined DBD overrides are returned in the reply message.

Note

XML syntax allows both a long form and a short form for empty tags. An empty `<request>` tag can therefore be coded in one of two ways.

Long form:

```
<request>
</request>
```

Equivalent short form:

```
<request/>
```

- ***Selected Site*** — Identify the desired IMS site in either the `<imsSiteName>` tag (using the ZMF remote site name where the IMS subsystem executes) or in the `<imsLogicalSite>` tag (using the ZMF nickname of the change library associated with the corresponding baseline or promotion site). The function returns all globally defined IMS DBD override statements for the named site.

- ***Selected Control Region*** — Enter the 4-byte ID of the IMS control region of interest in the `<imsSiteId>` tag. The function returns all global IMS DBD overrides for the named control region.
- ***Selected Source Component*** — Enter the name of the desired DBD library source component to override in the `<component>` tag. If known, enter the source component library type in the `<componentType>` tag. The function returns a list of global DBD control statement overrides for the named source component.
- ***Selected Control Statement Type*** — Enter the IMS keyword for the desired control statement type in the `<controlStatement>` tag. The function returns all global DBD overrides of the desired type.

Data structure details for the `<request>` data element appear in *Exhibit 8-9*. Note that the identical `<request>` syntax is used with both DBD and PSB override lists.

Exhibit 8-9. IMSOVRD GBL_DBD, GBL_PSB LIST <request>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<component>	Optional	0 -1	String (256), variable	Name of source component to override in IMS DBD (or PSB) library. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all IMS source components with global ZMF overrides. NOTE: Typically 8 bytes max.
<componentType>	Optional	0 -1	String (3), variable	Library type for component named in `<component>` tag. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all IMS component library types.
<controlStatement>	Optional	0 -1	String (8), variable	IMS keyword for type of IMS control statement being overridden. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all statement types.
<imsSiteId>	Optional	0 -1	String (4), variable	IMS control region ID associated with site in `<imsSiteName>`. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all IMS control regions.
<imsSiteName>	Optional	0 -1	String (8), variable	ZMF name of site where IMS subsystem executes. ***NOTE:*** Use asterisk (*) wildcard or omit tag to list all ZMF sites.

Global IMS DBD Override List — Replies

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` contains global override information for one IMS database description (DBD) control statement in one IMS source component located in the IMS DBD library.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Data structure details for the `<result>` tag appear in *Exhibit 8-10*. Note that the identical `<result>` syntax is used with both DBD and PSB override lists.

Exhibit 8-10. Global IMS DBD and PSB Override List <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<component>	Optional	0 -1	String (256), variable	Name of IMS component in baseline library. NOTE: Typically 8 bytes max.
<componentType>	Optional	0 -1	String (3), variable	Library type for component named in `<component>` tag.
<controlStatement>	Optional	0 -1	String (8), variable	The type of DBD (or PSB) control statement that has the IMS override.
<imsSiteId>	Optional	0 -1	String (4), variable	Name of IMS control region as specified by system programmer when IMS system was generated.
<imsSiteName>	Optional	0 -1	String (8), variable	ZMF name of site where IMS subsystem is running. May be **D**, **P**, or **DP** site.
<originalStatement>	Optional	0 -1	String (64), variable	Contents of the original DBD (or PSB) control statement.
<overrideStatement>	Optional	0 -1	String (64), variable	Contents of the DBD (or PSB) override statement.

IMS PSB Global Overrides - IMSOVRD GBL_PSB LIST

This function lists ChangeMan ZMF global overrides to the IMS program specification block (PSB) control statements associated with a specific source component in the IMS PSB library. Values returned include the PSB source component, IMS control statement type, the original PSB control statement contents, and the override content for that PSB statement. If no global PSB overrides are defined, no results are returned.

The Serena XML service/scope/message tags and attributes for messages to *list* global IMS PSB overrides are:

```
<service name="IMSOVRD">
<scope name="GBL_PSB">
<message name="LIST">
```

These tags appear in both requests and replies.

IMSOVRD GBL_PSB LIST — Requests

Serena XML supports five types of PSB override lists at the global level:

- ***Comprehensive List*** — Submit an empty `<request>` data element (that is, one that contains no subtags) in the XML request message. The `<request>` tag itself is required in the message to distinguish a request from a reply. All globally defined PSB overrides are returned in the reply message.

Note

XML syntax allows both a long form and a short form for empty tags. An empty `<request>` tag can therefore be coded in one of two ways.

Long form:

```
<request>
</request>
```

Equivalent short form:

```
<request/>
```

- ***Selected Site*** — Identify the desired IMS site in either the `<imsSiteName>` tag (using the ZMF remote site name where the IMS subsystem executes) or in the `<imsLogicalSite>` tag (using the ZMF nickname of the change library associated with the corresponding baseline or promotion site). The function returns all globally defined IMS PSB override statements for the named site.
- ***Selected Control Region*** — Enter the 4-byte ID of the IMS control region of interest in the `<imsSiteId>` tag. The function returns all global IMS PSB overrides for the named control region.
- ***Selected Source Component*** — Enter the name of the desired PSB library source component to override in the `<component>` tag. If known, enter the source component library type in the `<componentType>` tag. The function returns a list of global PSB control statement overrides for the named source component.
- ***Selected Control Statement Type*** — Enter the IMS keyword for the desired control statement type in the `<controlStatement>` tag. The function returns all global PSB overrides of the desired type.

The Serena XML syntax for a request to list global IMS PSB records is identical to that for DBD overrides. Data structure details for the `<request>` data element appear in *Exhibit 8-9* of the previous section.

IMSOVRD GBL_PSB LIST — Replies

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` contains global override information for one IMS program specification block (PSB) control statement in one IMS source component located in the IMS PSB library.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

The Serena XML syntax for a message to list global IMS PSB records is identical to that for DBD overrides. Details for the `<result>` tag appear in *Exhibit 8-10* of the previous section.

IMS Control Region Application Defaults - IMSCRGN APL LIST

This function lists IMS control region information associated with an application. The desired application name is required in the request message. Each returned record includes the control region ID, associated library names, the ZMF site name where the appropriate IMS subsystem resides, and application-level IMS override settings for building install jobs at staging and for generating IMS ACBs, DBDs, PSBs, and MFS source files at promotion.

The Serena XML service/scope/message tags and attributes for messages to *list* IMS control region records for an application are:

```
<service name="IMSCRGN">
<scope name="APL">
<message name="LIST">
```

These tags appear in both requests and replies.

IMSCRGN APL LIST — Requests

Serena XML supports three types of IMS control region lists for an application:

- ***Comprehensive List*** — Enter the name of the desired application in the `<applName>` tag and omit all other tags to list all IMS control regions defined for the named application.
- ***Selected Site*** — Enter the name of the desired application in the `<applName>` tag. Also identify the desired IMS site in either `<imsSiteName>` (using the ZMF remote site name where the IMS subsystem executes) or in `<imsLogicalSite>` (using the ZMF name of the logical change library associated with the corresponding baseline or promotion site). The function returns all IMS control regions for the named application and desired site.
- ***Selected Control Region*** — Enter the name of the desired application in the `<applName>` tag and the 4-byte ID of the IMS control region of interest in the `<imsControlRegion>` tag to retrieve control region specifications for the named application and control region.

Example XML — IMSCRGN APL LIST Request

```
<?xml version="1.0"?>
<service name="IMSCRGN">
 <scope name="APL">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
```

```
    <applName>IMSQ</applName>
  </request>
  </message>
 </scope>
</service>
```

Data structure details for the `<request>` data element appear in *Exhibit 8-11*.

Exhibit 8-11. IMSCRGN APL LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	0 -1	String (4), variable	ZMF application name. ***NOTE:*** OK to omit trailing blanks.
<imsControlRegion>	Optional	0 -1	String (4), variable	IMS control region ID. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all IMS control regions for named application.
<imsLogicalSite>	Optional	0 -1	String (8), variable	ZMF baseline or promotion library name corresponding to IMS site name in `<imsSiteName>`. ***NOTE:*** Value may be **BASELINE** (all caps) or the name of any promotion library defined in ZMF. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all IMS change library site names for application.
<imsSiteName>	Optional	0 -1	String (8), variable	ZMF name of remote site where IMS subsystem is running. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all IMS subsystem sites for application.

IMSCRGN APL LIST — Replies

The reply message for an application-level IMS control region list returns zero to many `<result>` data elements. The `<result>` subtags are similar to those of the PACKAGE IMS_CRGN LIST reply; the only difference is that the `<applName>`, `<package>`, and `<packageId>` tags are omitted in the IMSCRGN APL LIST reply. Refer to *Exhibit 8-2* for data structure details.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

IMS Control Region Global Defaults - IMSCRGN GBL LIST

This function lists global IMS control region information. Each returned record includes the control region ID, associated library names, the ZMF site name where the appropriate IMS subsystem resides, and global-level IMS override settings for building install jobs at staging and for generating IMS ACBs, DBDs, PSBs, and MFS source files at promotion.

The Serena XML service/scope/message tags and attributes for messages to *list* global IMS control region records are:

```
<service name="IMSCRGN">
<scope name="GBL">
<message name="LIST">
```

These tags appear in both requests and replies.

IMSCRGN GBL LIST — Requests

Serena XML supports three types of IMS control region lists at the global level:

- ***Comprehensive List*** — Submit an empty `<request>` data element (that is, one that contains no subtags) in the XML request message. The `<request>` tag itself is required in the message to distinguish a request from a reply. All globally-defined IMS control regions are returned in the reply message.

Note

XML syntax allows both a long form and a short form for empty tags. An empty `<request>` tag can therefore be coded in one of two ways.

Long form:

```
<request>
</request>
```

Equivalent short form:

```
<request/>
```

- ***Selected Site*** — Identify the desired IMS site in the `<imsSiteName>` tag (using the ZMF remote site name where the IMS subsystem executes). The function returns all globally-defined IMS control regions for the named site.
- ***Selected Control Region*** — Enter the 4-byte ID of the IMS control region of interest in the `<imsControlRegion>` tag. The function returns all global control region specifications for the named control region.

Data structure details for the `<request>` data element appear in *Exhibit 8-12*.

Exhibit 8-12. IMSCRGN GBL LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<imsControlRegion>	Optional	0 -1	String (4), variable	IMS control region ID. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all IMS control regions.
<imsSiteName>	Optional	0 -1	String (8), variable	ZMF name of remote site where IMS subsystem is running. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all IMS subsystem sites.

IMSCRGN GBL LIST — Replies

The reply message for a global-level IMS control region list returns zero to many `<result>` data elements. The `<result>` subtags are similar to those of the PACKAGE IMS_CRGN LIST reply; the only difference is that the `<applName>, <imsLogicalSite>, <package>,` and `<packageId>` tags are omitted in the IMSCRGN GBL LIST reply. Refer to *Exhibit 8-2* for data structure details.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

DB2 Development and Administration

Serena XML supports the following DB2 database tasks for general use:

- *DB2 Active Libraries for Application - DB2ADMIN APL_ACTV LIST*
- *DB2 Logical Subsystems for Application - DB2ADMIN APL_LOGL LIST*
- *DB2 Global Physical Subsystems - DB2ADMIN GBL_PHYS LIST*
- *DB2 Global Logical Subsystems - DB2ADMIN GBL_LOGL LIST*

DB2 Active Libraries for Application - DB2ADMIN APL_ACTV LIST

This function lists two kinds of DB2 “active” libraries for an application:

- ***Bind Active Libraries*** — Promotion and production libraries defined in the DB2 Option to execute DB2 binds at promote, demote, install, and backout.
- ***Stored Procedure Active Libraries*** — Promotion and production libraries defined in the DB2 Option to invoke special processing after changes to stored procedures, triggers, and user-defined functions.

Values returned for these libraries include the physical dataset name, the DB2 logical subsystem ID, and the DB2 entity type. If no DB2 active libraries are defined for an application, no results are returned.

The Serena XML service/scope/message tags and attributes for messages to *list* active DB2 libraries for an application are:

```
<service name=”DB2ADMIN”>
<scope name=”APL_ACTV”>
<message name=”LIST”>
```

These tags appear in both requests and replies.

To display the same information using ChangeMan ZMF, if you have the required access, you can go to the Selectable Options panel (CMNGBSOP).

DB2ADMIN APL_ACTV LIST — Requests

Serena XML supports two kinds of DB2 active library list requests for an application:

- ***Comprehensive List*** — Enter the name of the desired application in the `<applName>` tag to list all DB2 active libraries for that application.
- ***Single Logical Subsystem*** — Name the desired application in the `<applName>` tag and the DB2 logical subsystem ID of interest in the `<db2LogicalName>` tag. Information is returned for the named DB2 subsystem only.

Example XML — DB2ADMIN APL_ACTV LIST Request

```
<?xml version="1.0"?>
<service name="DB2ADMIN">
 <scope name="APL_ACTV">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <applName>CISQ</applName>
   </request>
  </message>
 </scope>
</service>
```

Data structure details for the `<request>` tag appear in *Exhibit 8-13*.

Exhibit 8-13. DB2ADMIN APL_ACTV LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	1	String (4), variable	ZMF application name. ***NOTE:*** OK to omit trailing blanks.
<db2LogicalName>	Optional	0 - 1	String (8), variable	DB2 logical subsystem ID of interest. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all DB2 subsystems for named application.

DB2ADMIN APL_ACTV LIST — Replies

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` describes one active DB2 library associated with the application named in the request. If no active DB2 libraries are defined for the application using the DB2 Option of ChangeMan ZMF, no `<result>` tags are returned in the reply message.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Example XML — DB2ADMIN APL_ACTV LIST Reply

```
<?xml version="1.0"?>
<service name="DB2ADMIN">
 <scope name="APL_ACTV">
  <message name="LIST">
   <result>
    <applName>CISQ</applName>
    <db2LogicalName>PROM810</db2LogicalName>
    <db2Lib>CMNTP.SERT8.PROM.CISQ.C001AUT.DBR</db2Lib>
    <db2LibType>B</db2LibType>
   </result>
   <result>
    <applName>CISQ</applName>
    <db2LogicalName>PROM810</db2LogicalName>
    <db2Lib>CMNTP.SERT8.PROM.CISQ.C001AUT.DBB</db2Lib>
    <db2LibType>B</db2LibType>
   </result>
   <result>
    <applName>CISQ</applName>
    <db2LogicalName>PROM810</db2LogicalName>
    <db2Lib>CMNTP.SERT8.PROM.CISQ.C001AQA.DBR</db2Lib>
    <db2LibType>B</db2LibType>
   </result>
   <result>
    <applName>CISQ</applName>
    <db2LogicalName>PROM810</db2LogicalName>
    <db2Lib>CMNTP.SERT8.PROM.CISQ.C001AQA.DBB</db2Lib>
    <db2LibType>B</db2LibType>
   </result>
   <response>
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Data structure details for the `<result>` tag appear in *Exhibit 8-14*.

Exhibit 8-14. DB2ADMIN APL_ACTV LIST <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	1	String (4), variable	ZMF name of application.
<db2Lib>	Optional	0 - 1	String (44), variable	Fully qualified dataset name of DB2 active library.

<db2LibType>	Optional	0 -1	String (1)	Code for DB2 entity type. Values:
				B = DBB bind control
				S = Stored procedure load/REXX

<db2LogicalName>	Optional	0 -1	String (8), variable	DB2 logical subsystem ID (or partition) defined for the active library.

DB2 Logical Subsystems for Application - DB2ADMIN APL_LOGL LIST

This function lists specifications for the DB2 logical subsystems defined for an application. An application name is required in the request. Returned values include DB2 logical and physical subsystem IDs and description, the site where the DB2 subsystem runs, and application-level configuration settings. DB2 subsystems must be properly configured in both DB2 and ChangeMan ZMF prior to running this service, or unexpected results may occur.

The Serena XML service/scope/message tags and attributes for messages to *list* DB2 logical subsystems for an application are:

```
<service name=”DB2ADMIN”>
<scope name=”APL_LOGL”>
<message name=”LIST”>
```

These tags appear in both requests and replies.

DB2ADMIN APL_LOGL LIST — Requests

Serena XML supports the following options for listing DB2 logical subsystem at the application level:

- ***Comprehensive List*** — Name the desired application in the `<applName>` tag to list all logical subsystems defined for the application.
- ***Selected Site List*** — Name the desired application in the `<applName>` tag and the DB2 site of interest in the `<db2SiteName>` tag to list all DB2 logical subsystems for the named application at the selected site.
- ***Selected Physical Subsystem List*** — Name the desired application in the `<applName>` tag. In the `<db2SubSystemId>` tag, identify the DB2 physical subsystem where the parameters and templates of the logical subsystem(s) will be used. All logical subsystems defined for that physical subsystem in the named application are returned in the reply.
- ***Single Logical Subsystem*** — Name the desired application in the `<applName>` tag. Identify the DB2 logical subsystem of interest in the `<db2LogicalName>` tag. Application-level specifications for the requested subsystem are returned in the reply.

Example XML — DB2ADMIN APL_LOGL LIST Request

```
<?xml version="1.0"?>
<service name="DB2ADMIN">
 <scope name="APL_LOGL">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <applName>CISQ</applName>
   </request>
  </message>
 </scope>
</service>
```

Data structure details for the `<request>` tag appear in *Exhibit 8-15*.

Exhibit 8-15. DB2ADMIN APL_LOGL LIST <request>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	1	String (4), variable	ZMF application name. ***NOTE:*** OK to omit trailing blanks.
<db2LogicalName>	Optional	0 - 1	String (8), variable	DB2 logical subsystem ID (or partition) of interest. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all DB2 logical subsystems for application.

Exhibit 8-15. DB2ADMIN APL_LOGL LIST <request> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<db2SiteName>	Optional	0 - 1	String (8), variable	ZMF name of remote site where DB2 physical subsystem runs. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all sites.
<db2SubSystemId>	Optional	0 - 1	String (4), variable	DB2 physical subsystem ID of interest. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all DB2 physical subsystems for application.

DB2ADMIN APL_LOGL LIST — Replies

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` contains application-level specifications for one DB2 logical subsystem. If no DB2 logical subsystems are defined for the application, no `<result>` tags are returned in the reply message.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Example XML — DB2ADMIN APL_LOGL LIST Reply

```
<?xml version="1.0"?>
<service name="DB2ADMIN">
 <scope name="APL_LOGL">
  <message name="LIST">
   <result>
    <applName>CISQ</applName>
    <db2SubSystemId>C105</db2SubSystemId>
    <db2LogicalName>PROD</db2LogicalName>
    <db2SiteName>SERT8</db2SiteName>
    <bindOwnerInsert>PROD</bindOwnerInsert>
    <packageTargetPattern>???7</packageTargetPattern>
    <bindQualifierTargetPattern>?????T</bindQualifierTargetPattern>
    <logicalSubSysBindFailFlag>Y</logicalSubSysBindFailFlag>
    <recycleStoredProcs>Y</recycleStoredProcs>
    <keepTriggerSequence>Y</keepTriggerSequence>
   </result>
   <result>
    <applName>CISQ</applName>
    <db2SubSystemId>C105</db2SubSystemId>
    <db2LogicalName>PROM810</db2LogicalName>
    <db2SiteName>SERT8</db2SiteName>
    <logicalSubSysBindFailFlag>Y</logicalSubSysBindFailFlag>
    <recycleStoredProcs>Y</recycleStoredProcs>
    <keepTriggerSequence>Y</keepTriggerSequence>
   </result>
   <response>
```

```
    <statusMessage>CMN8700I - LIST service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Data structure details for the `<result>` tag appear in *Exhibit 8-16*.

Exhibit 8-16. Application-Level DB2 Logical Subsystem List <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	1	String (4), variable	ZMF name of application.
<bindOwnerInsert>	Optional	0 - 1	String (128), variable	TSO ID of default bind owner to be inserted into DB2 templates.
<bindOwnerSourcePattern>	Optional	0 - 1	String (128), variable	Pattern or template for DB2 source bind owner.
<bindOwnerTargetPattern>	Optional	0 - 1	String (128), variable	Pattern or template for DB2 target bind owner.
<bindQualifierInsert>	Optional	0 - 1	String (128), variable	Default high-level qualifier for DB2 datasets in bind plan commands, bind package commands, and create trigger definitions.
<bindQualifierSourcePattern>	Optional	0 - 1	String (128), variable	Pattern or template for high-level qualifier name for DB2 source subsystem bind commands.
<bindQualifierTargetPattern>	Optional	0 - 1	String (128), variable	Pattern or template for high-level qualifier name for DB2 target subsystem bind commands.
<db2LogicalName>	Optional	0 - 1	String (8), variable	DB2 logical subsystem ID (or partition) returned.
<db2SiteName>	Optional	0 - 1	String (8), variable	ZMF name of remote site where DB2 physical subsystem runs.
<db2SubsystemID>	Optional	0 - 1	String (4), variable	DB2 physical subsystem ID where logical subsystem templates run.
<keepTriggerSequence>	Optional	0 - 1	String (1)	Y = Yes, rebuild all triggers to preserve trigger sequence for table/event if a trigger changes. N = No, don't preserve trigger sequence if change occurs; instead move changed trigger to end.
<locationSourcePattern>	Optional	0 - 1	String (128), variable	Pattern or template for location ID for DB2 source subsystem.

Exhibit 8-16. Application-Level DB2 Logical Subsystem List <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<locationTargetPattern>	Optional	0 - 1	String (128), variable	Pattern or template for location ID for DB2 target subsystem.
<logicalSubSysBindFailFlag>	Optional	0 - 1	String (1)	Y = Yes, bind failure is significant; stop promote/demote processing if DB2 bind fails. N = No, bind failure not significant; continue promote/demote process.
<logicalSubSystemComment>	Optional	0 - 1	String (30), variable	Free-format text description of DB2 logical subsystem (or partition).
<packageSourcePattern>	Optional	0 - 1	String (128), variable	Pattern or template for bind package name for DB2 source subsystem.
<packageTargetPattern>	Optional	0 - 1	String (128), variable	Pattern or template for bind package name for DB2 target subsystem.
<planNameSourcePattern>	Optional	0 - 1	String (24), variable	Pattern or template for bind plan name for DB2 source subsystem.
<planNameTargetPattern>	Optional	0 - 1	String (24), variable	Pattern or template for bind plan name for DB2 target subsystem.
<recycleStoredProcs>	Optional	0 - 1	String (1)	Y = Yes, automatically refresh stored procedures & external user-defined functions in DB2 if change occurs. N = No, don't automatically refresh stored procedures & user-defined functions in DB2.
<schemaSourcePattern>	Optional	0 - 1	String (128), variable	Pattern or template for explicit schema in procedure-name, function-name, or trigger-name in CREATE command for DB2 source subsystem.
<schemaTargetPattern>	Optional	0 - 1	String (128), variable	Pattern or template for explicit schema in procedure-name, function-name, or trigger-name in CREATE command for DB2 target subsystem.
<wlmEnvMask>	Optional	0 - 1	String (54), variable	Default mask for WLM address space containing stored procedures.
<wlmEnvSourcePattern>	Optional	0 - 1	String (54), variable	Pattern or template for name of WLM address space containing stored procedures for source DB2 subsystem.
<wlmEnvTargetPattern>	Optional	0 - 1	String (54), variable	Pattern or template for name of WLM address space containing stored procedures for target DB2 subsystem.

DB2 Global Physical Subsystems - DB2ADMIN GBL_PHYS LIST

This function lists the DB2 physical subsystems defined at the global level for a given ChangeMan ZMF started task. The request message may contain an empty `<request>`

data element or a selection of optional subtags. Returned values include the physical subsystem ID, the logical subsystem ID, the fully qualified dataset name of the associated DB2 system load library, and predefined job cards for data binding in the DB2 production environment. Physical DB2 subsystems must be defined both in DB2 and in the DB2 Option for ChangeMan ZMF prior to running the DB2 physical subsystems list service.

The Serena XML service/scope/message tags and attributes for messages to *list* globally defined DB2 physical subsystems are:

```
<service name=”DB2ADMIN”>
<scope name=”GBL_PHYS”>
<message name=”LIST”>
```

These tags appear in both requests and replies.

DB2ADMIN GBL_PHYS LIST — Requests

Serena XML supports three options for DB2 physical subsystem lists at the global level:

- ***Comprehensive List*** — Submit an empty `<request>` data element (that is, one that contains no subtags) in the XML request message. The `<request>` tag itself is required in the message to distinguish a request from a reply. All DB2 physical subsystems defined at the global level for the ChangeMan ZMF started task are returned in the reply message.

Note

XML syntax allows both a long form and a short form for empty tags. An empty `<request>` tag can therefore be coded in one of two ways.

Long form:

```
<request>
</request>
```

Equivalent short form:

```
<request/>
```

- ***Selected Site List*** — Name a desired site in the `<db2SiteName>` tag to request a list of all DB2 physical subsystems that reside at that site.
- ***Single Physical Subsystem*** — Name the DB2 physical subsystem of interest in the `<db2SubSystemId>` tag to list global specifications for the named site.

Example XML — DB2ADMIN GBL_PHYS LIST Request

```
<?xml version="1.0"?>
<service name="DB2ADMIN">
 <scope name="GBL_PHYS">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
   </request>
  </message>
 </scope>
</service>
```

Data structure details for the `<request>` tag appear in *Exhibit 8-17*.

Exhibit 8-17. DB2ADMIN GBL_PHYS LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<db2SiteName>	Optional	0 - 1	String (8), variable	ZMF site name where DB2 subsystem resides. ***NOTE:*** Use asterisk (*) wildcard or omit tag to list subsystems at all sites.
<db2SubsystemID>	Optional	0 - 1	String (4), variable	DB2 physical subsystem ID of interest. ***NOTE:*** Use asterisk (*) wildcard or omit tag to list all physical subsystems.

DB2ADMIN GBL_PHYS LIST — Replies

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` contains global specifications for one DB2 physical subsystem known to the ChangeMan ZMF started task that received the XML request message. If no DB2 subsystems are defined using the DB2 Option of ChangeMan ZMF, no `<result>` tags appear in the reply message.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Example XML — DB2ADMIN GBL_PHYS LIST Reply

```
<?xml version="1.0"?>
<service name="DB2ADMIN">
 <scope name="GBL_PHYS">
  <message name="LIST">
   <result>
    <db2SubSystemId>C101</db2SubSystemId>
    <db2SiteName>SERT8</db2SiteName>
    <db2SystemLoadLib>SYS2.DB2810.SDSNLOAD</db2SystemLoadLib>
 <db2ProdBindJobCards>//USER35A JOB ,ACCOUNT INFORMATION</db2ProdBindJobCards>
    <db2ProdBindJobCards>//*</db2ProdBindJobCards>
    <db2ProdBindJobCards>//*</db2ProdBindJobCards>
    <db2ProdBindJobCards>//*</db2ProdBindJobCards>
 <db2ProdBindJobCard01>//USER35A JOB ,ACCOUNT INFORMATION</db2ProdBindJobCard01>
    <db2ProdBindJobCard02>//*</db2ProdBindJobCard02>
    <db2ProdBindJobCard03>//*</db2ProdBindJobCard03>
    <db2ProdBindJobCard04>//*</db2ProdBindJobCard04>
   </result>
   <result>
    <db2SubSystemId>C105</db2SubSystemId>
    <db2SiteName>SERT8</db2SiteName>
    <db2SystemLoadLib>SYS2.DB2810.SDSNLOAD</db2SystemLoadLib>
    <db2ProdBindJobCards>//SERT8   JOB (AMW,000),&apos;C105&apos;,MSGCLASS=Y,</
db2ProdBindJobCards>
    <db2ProdBindJobCards>//             TIME=(,10),USER=USER24,NOTIFY=USER24</
db2ProdBindJobCards>
    <db2ProdBindJobCards>//*</db2ProdBindJobCards>
    <db2ProdBindJobCards>//*</db2ProdBindJobCards>
    <db2ProdBindJobCard01>//SERT8   JOB (AMW,000),&apos;C105&apos;,MSGCLASS=Y,</
db2ProdBindJobCard01>
    <db2ProdBindJobCard02>//             TIME=(,10),USER=USER24,NOTIFY=USER24</
db2ProdBindJobCard02>
    <db2ProdBindJobCard03>//*</db2ProdBindJobCard03>
    <db2ProdBindJobCard04>//*</db2ProdBindJobCard04>
   </result>
   <response>
  <statusMessage>CMN8700I - DB2 Admin service completed</statusMessage>
   <statusReturnCode>00</statusReturnCode>
   <statusReasonCode>8700</statusReasonCode>
  </response>
  </message>
 </scope>
</service>
```

Data structure details for the `<result>` tag appear in *Exhibit 8-18*.

Exhibit 8-18. DB2ADMIN GBL_PHYS LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<db2ProdBindJobCard01>	Optional	0 - 1	String (72), variable	First of up to four default JCL job cards used in DB2 production bind jobs for this subsystem.
<db2ProdBindJobCard02>	Optional	0 - 1	String (72), variable	Second of up to four default JCL job cards used in DB2 production bind jobs for this subsystem.
<db2ProdBindJobCard03>	Optional	0 - 1	String (72), variable	Third of up to four default JCL job cards used in DB2 production bind jobs for this subsystem.
<db2ProdBindJobCard04>	Optional	0 - 1	String (72), variable	Fourth of up to four default JCL job cards used in DB2 production bind jobs for this subsystem.
<db2ProdBindJobCards>	Optional	0 - 1	String (72), variable	Use tags db2ProdBindJobCards01 - 04 instead.
<db2SiteName>	Optional	0 - 1	String (8), variable	ZMF remote site nickname associated with DB2 subsystem.

Exhibit 8-18. DB2ADMIN GBL_PHYS LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<db2SubsystemID>	Optional	0 - 1	String (4), variable	DB2 physical subsystem ID assigned by ZMF administrator.
<db2SystemLoadLib>	Optional	0 - 1	String (44), variable	Physical data set name of DB2 load library for this subsystem.

DB2 Global Logical Subsystems - DB2ADMIN GBL_LOGL LIST

This function lists the global defaults for DB2 logical subsystem settings for a given ChangeMan ZMF started task. The request message may contain an empty `<request>` data element or a selection of optional subtags. Returned values include DB2 logical and physical subsystem IDs and description, the site where the DB2 subsystem runs, and global configuration settings. DB2 subsystems must be properly configured in both DB2 and ChangeMan ZMF prior to running this service, or unexpected results may occur.

The Serena XML service/scope/message tags and attributes for messages to *list* globally defined DB2 logical subsystems are:

```
<service name=”DB2ADMIN”>
<scope name=”GBL_LOGL”>
<message name=”LIST”>
```

These tags appear in both requests and replies.

DB2ADMIN GBL_LOGL LIST — Requests

Serena XML supports the following options for listing DB2 logical subsystems defined at the global level:

- ***Comprehensive List*** — Submit an empty `<request>` data element (that is, one that contains no subtags) in the XML request message. The `<request>` tag itself is required in the message to distinguish a request from a reply. All DB2 physical subsystems defined at the global level for the ChangeMan ZMF started task are returned in the reply message.

Note

XML syntax allows both a long form and a short form for empty tags. An empty `<request>` tag can therefore be coded in one of two ways.

Long form:

```
  <request>
  </request>
```

Equivalent short form:

```
  <request/>
```

- ***Selected Site List*** — Name the site of interest in the `<db2SiteName>` tag to list all DB2 logical subsystems for selected site.
- ***Selected Physical Subsystem List*** — In the `<db2SubSystemId>` tag, identify the DB2 physical subsystem where the parameters and templates of the logical subsystem(s) will be used. All logical subsystems defined for that physical subsystem are returned.
- ***Single Logical Subsystem*** — Name the DB2 logical subsystem of interest in the `<db2LogicalName>` tag. Global default specifications for the requested subsystem are returned in the reply.

Example XML — DB2ADMIN GBL_LOGL LIST Request

```
<?xml version="1.0"?>
<service name="DB2ADMIN">
 <scope name="GBL_LOGL">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
   </request>
  </message>
 </scope>
</service>
```

Data structure details for the `<request>` tag appear in *Exhibit 8-19*.

Exhibit 8-19. DB2ADMIN GBL_LOGL LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<db2LogicalName>	Optional	0 - 1	String (8), variable	DB2 logical subsystem ID (or partition) of interest. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all DB2 logical subsystems for application.
<db2SiteName>	Optional	0 - 1	String (8), variable	ZMF name of remote site where DB2 physical subsystem runs. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all sites.
<db2SubSystemId>	Optional	0 - 1	String (4), variable	DB2 physical subsystem ID of interest. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all DB2 physical subsystems for application.

DB2ADMIN GBL_LOGL LIST — Replies

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` tag contains global specifications for one DB2 logical subsystem associated with the ChangeMan ZMF started task that received the XML request message.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Example XML — DB2ADMIN GBL_LOGL LIST Reply

```
<?xml version="1.0"?>
<service name="DB2ADMIN">
 <scope name="GBL_LOGL">
  <message name="LIST">
   <result>
    <db2SubSystemId>C105</db2SubSystemId>
    <db2LogicalName>PROD</db2LogicalName>
    <db2SiteName>SERT8</db2SiteName>
    <bindOwnerInsert>PROD</bindOwnerInsert>
    <packageTargetPattern>???7</packageTargetPattern>
    <bindQualifierTargetPattern>?????T</bindQualifierTargetPattern>
    <logicalSubSysBindFailFlag>Y</logicalSubSysBindFailFlag>
    <recycleStoredProcs>Y</recycleStoredProcs>
    <keepTriggerSequence>Y</keepTriggerSequence>
   </result>
   <result>
    <db2SubSystemId>C105</db2SubSystemId>
    <db2LogicalName>PROM810</db2LogicalName>
    <db2SiteName>SERT8</db2SiteName>
    <logicalSubSysBindFailFlag>Y</logicalSubSysBindFailFlag>
    <recycleStoredProcs>Y</recycleStoredProcs>
    <keepTriggerSequence>Y</keepTriggerSequence>
   </result>
   <response>
    <statusMessage>CMN8700I - DB2 Admin service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Data structure details for the `<result>` tag appear in *Exhibit 8-20*.

Exhibit 8-20. DB2ADMIN GBL_LOGL LIST<result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<bindOwnerInsert>	Optional	0 - 1	String (128), variable	TSO ID of default bind owner to be inserted into DB2 templates.
<bindOwnerSourcePattern>	Optional	0 - 1	String (128), variable	Pattern or template for DB2 source bind owner.
<bindOwnerTargetPattern>	Optional	0 - 1	String (128), variable	Pattern or template for DB2 target bind owner.
<bindQualifierInsert>	Optional	0 - 1	String (128), variable	Default high-level qualifier for DB2 datasets in bind plan commands, bind package commands, and create trigger definitions.
<bindQualifierSourcePattern>	Optional	0 - 1	String (128), variable	Pattern or template for high-level qualifier name for DB2 source subsystem bind commands.
<bindQualifierTargetPattern>	Optional	0 - 1	String (128), variable	Pattern or template for high-level qualifier name for DB2 target subsystem bind commands.
<db2LogicalName>	Optional	0 - 1	String (8), variable	DB2 logical subsystem ID (or partition) returned.
<db2SiteName>	Optional	0 - 1	String (8), variable	ZMF name of remote site where DB2 physical subsystem runs.
<db2SubsystemID>	Optional	0 - 1	String (4), variable	DB2 physical subsystem ID where logical subsystem templates run.
<keepTriggerSequence>	Optional	0 - 1	String (1)	Y = Yes, rebuild all triggers to preserve trigger sequence for table/event if a trigger changes. N = No, don't preserve trigger sequence if change occurs; instead move changed trigger to end.
<locationSourcePattern>	Optional	0 - 1	String (128), variable	Pattern or template for location ID for DB2 source subsystem.
<locationTargetPattern>	Optional	0 - 1	String (128), variable	Pattern or template for location ID for DB2 target subsystem.
<logicalSubSysBindFailFlag>	Optional	0 - 1	String (1)	Y = Yes, bind failure is significant; stop promote/demote processing if DB2 bind fails. N = No, bind failure not significant; continue promote/demote process.
<logicalSubSystemComment>	Optional	0 - 1	String (30), variable	Free-format text description of DB2 logical subsystem (or partition).
<packageSourcePattern>	Optional	0 - 1	String (128), variable	Pattern or template for bind package name for DB2 source subsystem.

Exhibit 8-20. DB2ADMIN GBL_LOGL LIST<result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<packageTargetPattern>	Optional	0 - 1	String (128), variable	Pattern or template for bind package name for DB2 target subsystem.
<planNameSourcePattern>	Optional	0 - 1	String (24), variable	Pattern or template for bind plan name for DB2 source subsystem.
<planNameTargetPattern>	Optional	0 - 1	String (24), variable	Pattern or template for bind plan name for DB2 target subsystem.
<recycleStoredProcs>	Optional	0 - 1	String (1)	Y = Yes, automatically refresh stored procedures & external user-defined functions in DB2 if change occurs. N = No, don't automatically refresh stored procedures & user-defined functions in DB2.
<schemaSourcePattern>	Optional	0 - 1	String (128), variable	Pattern or template for explicit schema in procedure-name, function-name, or trigger-name in CREATE command for DB2 source subsystem.
<schemaTargetPattern>	Optional	0 - 1	String (128), variable	Pattern or template for explicit schema in procedure-name, function-name, or trigger-name in CREATE command for DB2 target subsystem.
<wlmEnvMask>	Optional	0 - 1	String (54), variable	Default mask for WLM address space containing stored procedures.
<wlmEnvSourcePattern>	Optional	0 - 1	String (54), variable	Pattern or template for name of WLM address space containing stored procedures for source DB2 subsystem.
<wlmEnvTargetPattern>	Optional	0 - 1	String (54), variable	Pattern or template for name of WLM address space containing stored procedures for target DB2 subsystem.

ONLINE FORMS MANAGEMENT 9

Serena XML supports forms-based package workflow management for customers who have installed the Online Forms Manager option of ChangeMan ZMF. Serena XML helps these customers automate form processing tasks. It also facilitates data interchange between ChangeMan ZMF online forms and external workflow management software.

Support is provided for two user task groups:

- ***Online Forms Lifecycle Tasks*** — Work with online forms associated with a package in motion. Typical commands include *submit*, *approve*, *reject*, and *comment*.
- ***Forms Information Management*** — Retrieve online form definitions and contents. The typical command is *list*.

The XML syntax that identifies online form functions appears in the name attribute of either the `<service>` tag or the `<scope>` tag, as follows:

```
<service name="FORMS">
<scope name="FORMS">
```

ONLINE FORMS LIFECYCLE TASKS

Online forms participate in the package management lifecycle. The following lifecycle tasks for forms are supported for general use:

- *Unfreeze Online Forms - PACKAGE FORMS UNFREEZE*
- *Refreeze Online Forms - PACKAGE FORMS REFREEZE*
- *Submit a Form for Approval - FORMS PKG SUBMIT*
- *Approve a Form - FORMS PKG APPROVE*
- *Reject a Form - FORMS PKG REJECT*
- *Add Comments to a Form - FORMS PKG COMMENT*

Unfreeze Online Forms - PACKAGE FORMS UNFREEZE

Serena XML lets you unfreeze online forms in a frozen package for online completion, approval, or other changes. Options exist to collectively unfreeze all online forms in a package or to selectively unfreeze only those forms named in the request.

The Serena XML service/scope/message tags for a package-level *unfreeze* message for online forms are:

```
<service name="PACKAGE">
<scope name="FORMS">
<message name="UNFREEZE">
```

These tags appear in both requests and replies.

Note

The forms unfreeze request takes the value "package" in the `name` attribute of the `<service>` tag because it is executed by the low-level package service. The forms-specific scope of the service is shown in the `name` attribute of the `<scope>` tag, which takes the value "`forms`".

PACKAGE FORMS UNFREEZE — Requests

The forms unfreeze function requires a package name in the `<package>` tag as input. It assumes a full unfreeze is requested unless you specify otherwise by supplying one or more form numbers. Select the desired unfreeze option as follows:

- ***Full Unfreeze*** — Omit the `<formNumber>` and `<listcount>` tags to unfreeze all online forms in a package. This is the default.
- ***Selective Unfreeze*** — Supply one or more form numbers in the `<formNumber>` tag to selectively unfreeze the identified forms.

The following example shows how you might code a selective forms unfreeze request. Data structure details for the `<request>` tag appear in *Exhibit 9-1*.

Example XML — PACKAGE FORMS UNFREEZE Request

```
<?xml version="1.0"?>
<service name="PACKAGE">
 <scope name="FORMS">
  <message name="UNFREEZE">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <package>ACTP000009</package>
    <listCount>0001</listCount>
    <formNumber>010</formNumber>
   </request>
  </message>
 </scope>
</service>
```

Exhibit 9-1. PACKAGE FORMS UNFREEZE <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<formNumber>	Optional	1	String (3), variable	ZMF form ID of desired online form.
<listCount>	Optional	0 - 1	Integer (3), variable	Number of forms named in request. Must match number of `<forms>` tags. Value range: 1 - 800. ***NOTE:*** Required for selective unfreeze or refreeze. ***NOTE:*** If used, `<forms>` tag is also required.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.

PACKAGE FORMS UNFREEZE — Replies

The reply for a forms unfreeze message does not return a `<result>` data structure. It does, however, return a standard `<response>` data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Refreeze Online Forms - PACKAGE FORMS REFREEZE

This function refreezes previously unfrozen online forms in package, preventing further change. Options exist to collectively refreeze all online forms in a package or to selectively refreeze only those forms named in the request.

The Serena XML service/scope/message tags for a package-level *refreeze* message for online forms are:

```
<service name="PACKAGE">
<scope name="FORMS">
<message name="REFREEZE">
```

These tags appear in both requests and replies.

Note

The forms refreeze request takes the value "package" in the `name` attribute of the `<service>` tag because it is executed by the low-level package service. The forms-specific scope of the service is shown in the `name` attribute of the `<scope>` tag, which takes the value "`forms`".

PACKAGE FORMS REFREEZE — Requests

As with unfreeze requests, Serena XML supports two types of package-level refreeze requests for custom forms:

Select the desired unfreeze option as follows:

- ***Full Refreeze*** — Omit the `<formNumber>` and `<listcount>` tags to unfreeze all online forms in a package. This is the default.
- ***Selective Refreeze*** — Supply one or more form numbers in the `<formNumber>` tag (s) to selectively unfreeze the identified forms.

The `<request>` tag data structure for a forms *refreeze* request is identical to that for an forms *unfreeze* request. See *Exhibit 9-1* for details.

PACKAGE FORMS REFREEZE — Replies

The reply for a forms refreeze message does not return a `<result>` data structure. It does, however, return a standard `<response>` data structure to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Submit a Form for Approval - FORMS PKG SUBMIT

Once an online form has been completed interactively in ISPF, Serena XML can automate its submission for approval. This function assumes that approvers, notifications, and variables associated with a specific form have previously been defined using online forms maintenance.

The Serena XML service/scope/message tags for a message to *submit* an online form for approve are:

```
<service name="FORMS">
<scope name="PKG">
<message name="SUBMIT">
```

These tags appear in both requests and replies.

FORMS PKG SUBMIT — Requests

Only one form may be submitted for approval at a time using Serena XML. Both the form number and the package where the desired instance of the form resides are required in the request. Data structure details for the `<request>` tag appear in *Exhibit 9-2*.

Exhibit 9-2. FORMS PKG SUBMIT <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<formNumber>	Required	1	String (3), variable	ZMF form ID of desired online form.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name where desired instance of form resides.
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.

FORMS PKG SUBMIT — Replies

The reply for a forms submit message does not return a `<result>` data structure. It does, however, return a standard `<response>` data structure to indicate the success or failure of the submit request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Approve a Form - FORMS PKG APPROVE

This function approves the action requested by a previously submitted online form. If approval of the form in the interactive ChangeMan ZMF environment would normally submit a job stream for execution, that same job stream is submitted for execution upon approval in Serena XML.

The Serena XML service/scope/message tags for a message to *approve* an online form are:

```
<service name="FORMS">
<scope name="PKG">
<message name="APPROVE">
```

These tags appear in both requests and replies.

FORMS PKG APPROVE — Requests

Only one form may be approved at a time using Serena XML. Both the form number and the package where the desired instance of the form resides are required in the request. Data structure details for the `<request>` tag appear in *Exhibit 9-3*.

Exhibit 9-3. Form Approval <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 -1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<formNumber>	Required	1	String (3), variable	ZMF form ID of desired online form.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name where desired instance of form resides.
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.

FORMS PKG APPROVE — Replies

No `<result>` tags are returned in the Serena XML reply message for a form approval request. However, the reply message does return a standard `<response>` data element to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Reject a Form - FORMS PKG REJECT

This function rejects the action requested by a previously submitted online form. The form may not be open for concurrent use at the time it is rejected. Reject reasons are required.

To add reject reasons separately as comments, without actually rejecting the form request, use the online forms comment service. (See *"Add Comments to a Form - FORMS PKG COMMENT"*.)

The Serena XML service/scope/message tags for an online form *reject* message are:

```
<service name="FORMS">
<scope name="PKG">
<message name="REJECT">
```

These tags appear in both requests and replies.

FORMS PKG REJECT — Requests

Requests to reject a form require a package name, form number, and at least one reason for the rejection. Only one form may be rejected at a time. Data structure details for the `<request>` tag appear in *Exhibit 9-4*.

Exhibit 9-4. FORMS PKG REJECT <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<formNumber>	Required	1	String (3), variable	ZMF form ID of desired online form.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name where desired instance of form resides.
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<rejectReason01>	Required	1	String (72), variable	Text of reason for rejecting action requested by form. First of up to ten lines of text.
<rejectReason02>	Optional	0 - 1	String (72), variable	Text of reason for rejecting action requested by form. Second of up to ten lines of text.

Exhibit 9-4. FORMS PKG REJECT <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<rejectReason03>	Optional	0 - 1	String (72), variable	Text of reason for rejecting action requested by form. Third of up to ten lines of text.
<rejectReason04>	Optional	0 - 1	String (72), variable	Text of reason for rejecting action requested by form. Fourth of up to ten lines of text.
<rejectReason05>	Optional	0 - 1	String (72), variable	Text of reason for rejecting action requested by form. Fifth of up to ten lines of text.
<rejectReason06>	Optional	0 - 1	String (72), variable	Text of reason for rejecting action requested by form. Sixth of up to ten lines of text.
<rejectReason07>	Optional	0 - 1	String (72), variable	Text of reason for rejecting action requested by form. Seventh of up to ten lines of text.
<rejectReason08>	Optional	0 - 1	String (72), variable	Text of reason for rejecting action requested by form. Eighth of up to ten lines of text.
<rejectReason09>	Optional	0 - 1	String (72), variable	Text of reason for rejecting action requested by form. Ninth of up to ten lines of text.
<rejectReason10>	Optional	0 - 1	String (72), variable	Text of reason for rejecting action requested by form. Tenth of up to ten lines of text.

FORMS PKG REJECT — Replies

No `<result>` tags are returned in the Serena XML reply message for an online form rejection. However, the reply message does return a standard `<response>` data element to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Add Comments to a Form - FORMS PKG COMMENT

This function allows you to add free-format text reasons for rejecting the action requested by an online form without committing to a reject decision. The rejection action itself requires a separate service. (See *"Reject a Form - FORMS PKG REJECT"*.)

The Serena XML service/scope/message tags for an online form *comment* message are:

```
<service name="FORMS">
<scope name="PKG">
<message name="COMMENT">
```

These tags appear in both requests and replies.

FORMS PKG COMMENT — Requests

The request message for this function requires a package name, a form number, and at least one comment, which is assumed to be a reason for rejecting the requested action but need not be. Up to ten lines of comment text (reject reasons) are supported.

Data structure details for the `<request>` tag appear in *Exhibit 9-5*.

Exhibit 9-5. FORMS PKG COMMENT<request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<formNumber>	Required	1	String (3), variable	ZMF form ID of desired online form.
<package>	Required	1	String (10), fixed	Fixed-format ZMF package name where desired instance of form resides.
<packageId>	Optional	0 - 1	Integer (6), variable	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.
<rejectReason01>	Required	1	String (72), variable	Text of reason for rejecting action requested by form. First of up to ten lines of text.
<rejectReason02>	Optional	0 - 1	String (72), variable	Text of reason for rejecting action requested by form. Second of up to ten lines of text.
<rejectReason03>	Optional	0 - 1	String (72), variable	Text of reason for rejecting action requested by form. Third of up to ten lines of text.
<rejectReason04>	Optional	0 - 1	String (72), variable	Text of reason for rejecting action requested by form. Fourth of up to ten lines of text.
<rejectReason05>	Optional	0 - 1	String (72), variable	Text of reason for rejecting action requested by form. Fifth of up to ten lines of text.
<rejectReason06>	Optional	0 - 1	String (72), variable	Text of reason for rejecting action requested by form. Sixth of up to ten lines of text.

Exhibit 9-5. FORMS PKG COMMENT<request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<rejectReason07>	Optional	0 - 1	String (72), variable	Text of reason for rejecting action requested by form. Seventh of up to ten lines of text.
<rejectReason08>	Optional	0 - 1	String (72), variable	Text of reason for rejecting action requested by form. Eighth of up to ten lines of text.
<rejectReason09>	Optional	0 - 1	String (72), variable	Text of reason for rejecting action requested by form. Ninth of up to ten lines of text.
<rejectReason10>	Optional	0 - 1	String (72), variable	Text of reason for rejecting action requested by form. Tenth of up to ten lines of text.

FORMS PKG COMMENT — Replies

No `<result>` tags are returned in the Serena XML reply message for an online form comment. However, the reply message does return a standard `<response>` data element to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

FORMS INFORMATION MANAGEMENT

Forms information management tasks retrieve online form definitions and contents. Serena XML supports the following tasks for general use:

- *List Global Online Forms - FORMS GBL LIST*
- *List Package Online Forms - FORMS PKG LIST*
- *List Package Online Form Details - FORMS PKG DETAIL*

List Global Online Forms - FORMS GBL LIST

The global forms list function retrieves global form definitions previously created using online forms maintenance. Included in the returned definition are a global description of the form, approver and notification information, and a variable list defined by the customer during forms maintenance.

The Serena XML service/scope/message tags for a message to *list* the global form definitions are:

```
<service name="FORMS">
<scope name="GBL">
<message name="LIST">
```

These tags appear in both requests and replies.

FORMS GBL LIST — Requests

The global online forms list supports two kinds of requests:

- ***Complete Forms List*** — Use the "match-all" (asterisk) wildcard in the `<formNumber>` tag to request information about all globally defined online forms.
- ***Selective Forms List*** — Enter the form number in the `<formNumber>` tag to request the global specification for the named online form.

Data structure details for the `<request>` tag appear in *Exhibit 9-6*.

Exhibit 9-6. FORMS GBL LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<formNumber>	Optional	1	String (3), variable	ZMF form ID of desired online form. ***NOTE:*** Asterisk (*) wildcard requests all forms.

FORMS GBL LIST — Replies

The reply message for a global online forms list returns zero to many `<result>` data elements. Each `<result>` tag contains the global definition for one form identified by form number. Note that it includes two complex subtags with subordinate tags of their own: `<notification>` and `<varList>`.

The standard `<response>` data element follows the last `<result>` to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last tag returned, the `<response>` tag also serves as an end-of-list marker.

Data structure details for the `<result>` tag appear in *Exhibit 9-7*.

Exhibit 9-7. FORMS GBL LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<allowAccess>	Optional	0 - 1	String (1)	**Y** = Yes, allow form users to edit form instances in package. **N** = No user edits allowed.
<formDesc>	Optional	0 - 1	String (255), variable	Global description of form.

Exhibit 9-7. FORMS GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<formNumber>	Optional	1	String (3), variable	ZMF form ID of returned online form.
<leadTime>	Optional	0 - 1	Integer (3), variable	Lead time required to act on the form request, in days.
<notification>	Optional	1 - 10	Complex	Approver notification list. Complex tag. See *Exhibit 9-8*.
<submitOnApproval>	Optional	0 - 1	String (1)	**Y** = Yes, automatically submit job stream on form approval. **N** = No, don't automatically submit job stream on approval.
<varList>	Optional	1 - 200	Complex	User-defined variable list. Complex tag. See *Exhibit 9-9*

<notification> Subtag

The `<notification>` data element lists users to notify and a notification method to use when a form is submitted for action. This is a complex, repeatable. Data structure details for the `<notification>` subtag appear in *Exhibit 9-8*.

Exhibit 9-8. <notification> Subtag

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<notifierTypeDesc>	Optional	0 - 1	Integer (8), variable	ZMF notification method used. Values: •MVSSEND •EMAIL •SERNET •BATCH
<userList>	Optional	0 - 1	String (44), variable	Delimited list of TSO user IDs or email addresses to notify by the method shown in `<notifierType>` when form is submitted for approval.

<varList> Subtag

The `<varList>` data element defines the content of the form. Each instance of this complex tag defines an online form variable presented in the form's ISPF panel. This is a complex subtag that is repeatable to accommodate multiple variables.

Data structure details for the `<varList>` subtag appear in *Exhibit 9-9*.

Exhibit 9-9. <varList> Subtag

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<varName>	Optional	0 - 1	String (8), variable	Name of user-defined form variable returned. Corresponds to data field in ISPF panel.
<varLen>	Optional	0 - 1	Integer (4), variable	Maximum length of variable data in bytes. Value range: 1 to 44.
<varFormat>	Optional	0 - 1	String (1)	Code for type of data in variable. Values: **F** = Fixed-length string. **C** = Variable-length character.
<isTableEntry>	Optional	0 - 1	String (1)	**Y** = Yes, data entry OK in field. **N** = No data entry permitted.
<isKeyField>	Optional	0 - 1	String (1)	**Y** = Yes, variable is table key field. **N** = No, not table key field.

List Package Online Forms - FORMS PKG LIST

The package forms list function retrieves package forms associated with a package.

The Serena XML service/scope/message tags for a message to *list* package forms are:

```
<service name="FORMS">
<scope name="PKG">
<message name="LIST">
```

These tags appear in both requests and replies.

FORMS PKG LIST — Requests

The package forms list request may be customized using the form status tags.

Note

All yes/no status tags default to the value "Y" if no tag in the group has an explicitly assigned value. But if *any* flag tag in the group is explicitly assigned a value, all other tags in the group change their default values to "N."

Data structure details for the `<request>` tag appear in *Exhibit 9-10*.

Exhibit 9-10. FORMS PKG LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), variable	ZMF application name. Same as first 4 bytes of package name. ***NOTE:*** OK to omit trailing blanks. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`.
<filterActiveStatus	Optional	0 - 1	String (1)	**Y** = Active status. **N** = Not active status.
<filterApprovedStatus	Optional	0 - 1	String (1)	**Y** = Approved status. **N** = Not approved status.
<filterFrozenStatus	Optional	0 - 1	String (1)	**Y** = Frozen status. **N** = Not frozen status.
<filterInactiveStatus	Optional	0 - 1	String (1)	**Y** = Inactive status. **N** = Not inactive status.
<filterRejectedStatus	Optional	0 - 1	String (1)	**Y** = Rejected status. **N** = Not rejected status.
<filterSubmittedStatus	Optional	0 - 1	String (1)	**Y** = Submitted for approval status. **N** = Not submitted for approval status.
<filterUnfrozenStatus	Optional	0 - 1	String (1)	**Y** = Unfrozen status. **N** = Not unfrozen status.
<formNumber>	Optional	0 - 1	String (3), variable	ZMF form ID of desired online form.
<package>	Required	1	String (10)	Fixed-format ZMF package name where desired instance of form resides.
<packageId>	Optional	0 - 1	Integer (6)	ZMF package ID number. Same as last 6 bytes of package name. ***NOTE:*** Leading zeroes required. ***NOTE:*** Not recommended. Use `<package>` instead of `<applName>` & `<packageId>`.

FORMS PKG LIST — Replies

The reply message for a package online form list returns zero to many `<result>` data elements.

The standard `<response>` data element follows the last `<result>` to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last tag returned, the `<response>` tag also serves as an end-of-list marker.

Data structure details for the `<result>` tag appear in *Exhibit 9-11*.

Exhibit 9-11. FORMS PKG LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4)), variable	ZMF application name.
<approver>	Optional	0 - 1	String (8)), variable	TSO user ID of approver.
<formDesc>	Optional	0 - 1	String (40), variable	Form description.
<formName>	Optional	0 - 1	String (8), variable	Form name.
<formNumber>	Optional	0 - 1	String (3), variable	Form number.
<formStatus>	Optional	0 - 1	String (1)	Form status: **0** = Active **1** = Approved **4** = Frozen **5** = Inactive **9** = Rejected **B** = Submitted for approval **C** = Unfrozen
<isAccessAllowed>	Required	1	String (1)	**Y** = Yes, access allowed. **N** = No, access not allowed.
<isApprovalSubmitted>	Required	1	String (1)	**Y** = Yes, approval was submitted. **N** = No, approval was not submitted.
<lastUser>	Optional	0 - 1	String (8) variable	TSO user ID of last user to update form.
<package>	Optional	0 - 1	String (10)	Fixed-format ZMF package name.
<packageId>	Optional	0 - 1	Integer (6)	ZMF package ID number. Same as last 6 bytes of package name.

<rejectReasons>
  <rejectReason01>
   .
   .
   .
  <rejectReason10>
	Optional	0 - 10	String (72), variable	Reject reason text, zero to ten lines.

<user>	Optional	0 - 1	String (8) variable	TSO user ID of user who associated the form with the package.

List Package Online Form Details - FORMS PKG DETAIL

The package form detail list function retrieves a package form definition previously created using online forms maintenance.

The Serena XML service/scope/message tags for a message to *list* a package form definition are:

```
<service name="FORMS">
<scope name="PKG">
<message name="DETAIL">
```

These tags appear in both requests and replies.

FORMS PKG DETAIL— Requests

The package form detail list request is identical to the FORMS PKG LIST request. Refer to *Exhibit 9-10* for the `<request>` tag data structure details.

FORMS PKG DETAIL — Replies

The reply message for a package form detail list returns zero to many `<result>` data elements.

The standard `<response>` data element follows the last `<result>` to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last tag returned, the `<response>` tag also serves as an end-of-list marker.

Data structure details for the `<result>` tag appear in *Exhibit 9-12*.

Exhibit 9-12. FORMS PKG DETAIL <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<is KeyField>	Optional	0 - 1	String (1)	**Y** = Yes, key field. **N** = No, not key field.
<is TableEntry>	Optional	0 - 1	String (1)	**Y** = Yes, table entry. **N** = No, not table entry.
<varDataLen>	Optional	0 - 1	Integer (4)	Length of variable data.
<varFormat>	Optional	0 - 1	String (1)	Variable format: **C** = Character **F** = Fixed
<varName>	Optional	0 - 1	String (8), variable	Variable name.
<variablePortionLen>	Optional	1 - 10	String (5), variable	Length of this portion

CHANGEMAN ZMF ADMINISTRATION TASKS 10

Serena XML provides a broad range of ChangeMan ZMF administration functions for general use. These functions support administration tasks at both the global and application levels.

ChangeMan ZMF administration tasks include the following:

- ***Change Library Administration*** — Tasks that retrieve information about change libraries, including the baseline library, promotion libraries, and production libraries.
- ***Site Administration*** — Tasks that retrieve information about ChangeMan ZMF remote sites and the site installation calendar.
- ***Developer Environment Administration*** — Tasks that retrieve information about languages, library types, build procedures, and general development environment parameters at both the global and application levels.
- ***Approver and Notification Administration*** — Tasks that retrieve information about approvers and approver notification lists, or that notify users of an event.

CHANGE LIBRARY ADMINISTRATION

Serena XML supports the following change library administration tasks for general use:

- *List Baseline Library Datasets - BASELIB SERVICE LIST*
- *List Promotion Library Datasets - PROMLIB LIBRARY LIST*
- *List Promotion Site Configuration Records - PROMLIB SITE LIST*
- *List Production Library Datasets - PRODLIB SERVICE LIST*

The syntax that identifies these functions appears in the `name` attribute of the `<service>` tag, as follows:

```
<service name="BASELIB">
<service name="PROMLIB>
<service name="PRODLIB">
```

List Baseline Library Datasets - BASELIB SERVICE LIST

The Serena XML function to list baseline library datasets returns the fully qualified dataset names for the current baseline library and up to nine physical back-level libraries associated with a named application. All baseline libraries at all defined sites are included in the scope of this function. If no baseline libraries are defined, no results are returned in the reply message.

The Serena XML service/scope/message tags and attributes for messages to *list* baseline library datasets are:

```
<service name=”BASELIB”>
<scope name=”SERVICE”>
<message name=”LIST”>
```

These tags appear in both requests and replies.

BASELIB SERVICE LIST — Requests

Serena XML can request all baseline library datasets for an application. Alternatively, you can restrict the list of datasets returned to those for a particular library type or a particular site.

Example XML — BASELIB SERVICE LIST Request

```
<?xml version="1.0"?>
<service name="BASELIB">
 <scope name="SERVICE">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <applName>IMSQ</applName>
   </request>
  </message>
 </scope>
</service>
```

Data structure details for the `<request>` data element appear in *Exhibit 10-1*.

Exhibit 10-1. BASELIB SERVICE LIST<request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	1	String (4), variable	ZMF application name for which baseline datasets are requested. ***NOTE:*** OK to omit trailing blanks.
<libType>	Optional	0 - 1	String (3), variable	Library type for which baseline datasets are requested. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all library types.
<siteName>	Optional	0 - 1	String (8), variable	ZMF name of site where baseline libraries reside. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all sites.

BASELIB SERVICE LIST — Replies

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` tag lists baseline library datasets for one library type at a single site. All results apply to the application named in the request message.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Example XML — BASELIB SERVICE LIST Reply

```
<?xml version="1.0"?>
<service name="BASELIB">
 <scope name="SERVICE">
  <message name="LIST">
   <result>
    <applName>IMSQ</applName>
    <libType>SRC</libType>
    <installInProdOption>2</installInProdOption>
    <baseLibStorageMeans>7</baseLibStorageMeans>
    <baseLibLevel>010</baseLibLevel>
    <baseLibs>CMNTP.SERT8.BASE.IMSQ.SRC</baseLibs>
    <baseLibs>CMNTP.SERT8.BASE.IMSQ.SRC.DELTA</baseLibs>
    <baseLibs></baseLibs>
    <baseLibs></baseLibs>
    <baseLibs></baseLibs>
    <baseLibs></baseLibs>
    <baseLibs></baseLibs>
    <baseLibs></baseLibs>
    <baseLibs></baseLibs>
    <baseLibs></baseLibs>
    <baseLibName00>CMNTP.SERT8.BASE.IMSQ.SRC</baseLibName00>
    <baseLibName01>CMNTP.SERT8.BASE.IMSQ.SRC.DELTA</baseLibName01>
    <baseLibName02></baseLibName02>
    <baseLibName03></baseLibName03>
    <baseLibName04></baseLibName04>
    <baseLibName05></baseLibName05>
    <baseLibName06></baseLibName06>
    <baseLibName07></baseLibName07>
    <baseLibName08></baseLibName08>
    <baseLibName09></baseLibName09>
   </result>
.
.
.
   <response>
    <statusMessage>CMN8700I - Baseline library service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
 </service>
```

Data structure details for the `<result>` tag appear in *Exhibit 10-2*.

Exhibit 10-2. BASELIB SERVICE LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	1	String (4), variable	ZMF application name.
<baseLibLevel>	Optional	0 - 1	String (3), variable	Number of baseline library levels maintained.
<baseLibName00>	Optional	0 - 1	String (44), variable	Dataset name for baseline library level 0.
<baseLibName01>	Optional	0 - 1	String (44), variable	Dataset name for baseline library level -1. ***NOTE:*** This dataset also includes all back-level baseline ripples if `<baseLibStorageMeans>` is 3 or 7.
<baseLibName02>	Optional	0 - 1	String (44), variable	Dataset name for baseline library level -2. ***NOTE:*** Returned only if value in `<baseLibStorageMeans>` is *not* 3 or 7.
<baseLibName03>	Optional	0 - 1	String (44), variable	Dataset name for baseline library level -3. ***NOTE:*** Returned only if value in `<baseLibStorageMeans>` is *not* 3 or 7.
<baseLibName04>	Optional	0 - 1	String (44), variable	Dataset name for baseline library level -4. ***NOTE:*** Returned only if value in `<baseLibStorageMeans>` is *not* 3 or 7.
<baseLibName05>	Optional	0 - 1	String (44), variable	Dataset name for baseline library level -5. ***NOTE:*** Returned only if value in `<baseLibStorageMeans>` is *not* 3 or 7.
<baseLibName06>	Optional	0 - 1	String (44), variable	Dataset name for baseline library level -6. ***NOTE:*** Returned only if value in `<baseLibStorageMeans>` is *not* 3 or 7.
<baseLibName07>	Optional	0 - 1	String (44), variable	Dataset name for baseline library level -7. ***NOTE:*** Returned only if value in `<baseLibStorageMeans>` is *not* 3 or 7.
<baseLibName08>	Optional	0 - 1	String (44), variable	Dataset name for baseline library level -8. ***NOTE:*** Returned only if value in `<baseLibStorageMeans>` is *not* 3 or 7.
<baseLibName09>	Optional	0 - 1	String (44), variable	Dataset name for baseline library level -9. ***NOTE:*** Returned only if value in `<baseLibStorageMeans>` is *not* 3 or 7.

Exhibit 10-2. BASELIB SERVICE LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<baseLibs>	Optional	0 - 10	String (44), variable	Local baseline library dsname 0 - 9. ***NOTE:*** The <baseLibs> tags were replaced by the <baseLibNamenn> tags but have been temporarily retained to provide backward compatibility for customers' existing applications (they will be removed in a future release of ChangeMan ZMF). The <baseLibNamenn> tags should be used unless you have an existing custom application that uses the <baseLibs> tags.
<baseLibStorageMeans>	Optional	0 -1	String (1)	Code for baseline library storage method & format. Values: 2 = Delta deck (Obsolete) 3 = Librarian Archie 4 = Librarian 5 = Panvalet 6 = PDS 7 = Stacked reverse delta **H** = HFS (Hierarchical File System)
<installInProdOption>	Optional	0 - 1	String (1)	Code for production install option. Values: 1 = Install if production library exists 2 = Don't install in production library 3 = Install in production library
<libType>	Optional	1	String (3), variable	Library type included in baseline library.
<siteName>	Optional	1	String (8), variable	ZMF name of site where baseline library resides.

List Promotion Library Datasets - PROMLIB LIBRARY LIST

The Serena XML function to list promotion library datasets returns the fully qualified dataset names for the up to three target promotion datasets and the promotion shadow library associated with a named application. All promotion libraries at all defined sites and promotion levels are included in the scope of this function. If no promotion libraries are defined, no results are returned in the reply message.

The Serena XML service/scope/message tags and attributes for messages to *list* promotion library records are:

```
<service name=”PROMLIB”>
<scope name=”LIBRARY”>
<message name=”LIST”>
```

These tags appear in both requests and replies.

PROMLIB LIBRARY LIST — Requests

Serena XML can request an application’s promotion library records for one or all library types in the application and one or all promotion levels and sites where the application resides. Site-specific setup information is omitted.

Example XML — PROMLIB LIBRARY LIST Request

```
<?xml version="1.0"?>
<service name="PROMLIB">
 <scope name="LIBRARY">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <applName>IMSQ</applName>
   </request>
  </message>
 </scope>
</service>
```

Data structure details for the `<request>` data element appear in *Exhibit 10-3*.

Exhibit 10-3. PROMLIB LIBRARY LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	1	String (4), variable	ZMF application name. ***NOTE:*** OK to omit trailing blanks.
<excludeSysLib>	Optional	0 - 1	String (1)	Y = Include only datasets for libraries that exclude syslib processing N = Include only datasets for libraries that allow syslib processing ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all syslib options.
<libType>	Optional	0 - 1	String (3), variable	Library type for which promotion library datasets are requested. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all library types.

Exhibit 10-3. PROMLIB LIBRARY LIST <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<promotionLevel>	Optional	0 - 1	Integer (2), fixed	Numeric relative promotion level for which promotion library datasets are requested. ***NOTE:*** Corresponds to nickname in `<promotionName>`. ***NOTE:*** Leading zeroes required. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all promotion levels.
<promotionName>	Optional	0 - 1	String (8), variable	ZMF nickname for promotion library function & level desired. ***NOTE:*** Corresponds to code in `<promotionLevel>`. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all promotion levels.
<siteName>	Optional	0 - 1	String (8), variable	ZMF name of site where promotion library resides. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all sites for application.

PROMLIB LIBRARY LIST — Replies

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` tag lists all the promotion library datasets defined at a particular promotion level for a particular application, site, and library type. The dataset name for the corresponding shadow library also is returned.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Example XML — PROMLIB LIBRARY LIST Reply

```
<?xml version="1.0"?>
<service name="PROMLIB">
 <scope name="LIBRARY">
  <message name="LIST">
   <result>
    <applName>IMSQ</applName>
    <siteName>SERT8</siteName>
    <promotionName>C001AQA</promotionName>
    <promotionLevel>20</promotionLevel>
    <libType>CPY</libType>
    <shadowLib>CMNTP.SERT8.PROM.IMSQ.C001AQA.CPY</shadowLib>
    <promoLib01>CMNTP.SERT8.PROM.IMSQ.C001AQA.CPY</promoLib01>
    <excludeSysLib>Y</excludeSysLib>
   </result>
   <result>
    <applName>IMSQ</applName>
    <siteName>SERT8</siteName>
    <promotionName>C001AQA</promotionName>
    <promotionLevel>20</promotionLevel>
    <libType>DBB</libType>
    <shadowLib>CMNTP.SERT8.PROM.IMSQ.C001AQA.DBB</shadowLib>
    <promoLib01>CMNTP.SERT8.PROM.IMSQ.C001AQA.DBB</promoLib01>
    <excludeSysLib>Y</excludeSysLib>
   </result>
.
.
.
   <response>
    <statusMessage>CMN8700I - Promo Library service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Data structure details for the `<result>` tag appear in *Exhibit 10-4*.

Exhibit 10-4. PROMLIB LIBRARY LIST<result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	1	String (4), variable	ZMF application name.
<excludeSysLib>	Optional	0 - 1	String (1)	Y = Exclude promotion library from system processing for JCL file tailoring & audit N = Don't exclude library from system processing for JCL file tailoring & audit

<hfsLib>	Optional	0 - 1	String (1)	Does promotion library reside in the z/OS Unix Hierarchical File System?
				Y = Yes, HFS promotion library
				N = No, not HFS library

<libType>	Optional	1	String (3), variable	Library type included in promotion library.
<promoLib01>	Optional	0 -1	String (44), variable	Fully qualified dataset name of first target promotion library.
<promoLib02>	Optional	0 - 1	String (44), variable	Fully qualified dataset name of second target promotion library.
<promoLib03>	Optional	0 - 1	String (44), variable	Fully qualified dataset name of third target promotion library.

Exhibit 10-4. PROMLIB LIBRARY LIST<result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<promotionLevel>	Optional	0 - 1	Integer (2), fixed	Numeric relative promotion level of promotion library. Corresponds to nickname in `<promotionName>`.
<promotionName>	Optional	0 - 1	String (8), variable	ZMF nickname corresponding to code in `<promotionLevel>`.
<shadowLib>	Optional	0 -1	String (44), variable	Fully qualified dataset name of local promotion shadow library. ***NOTE:*** Returned only for remote promotion libraries.
<siteName>	Required	1	String (8), variable	ZMF name of site where promotion library resides.

List Promotion Site Configuration Records - PROMLIB SITE LIST

This function lists promotion site configuration records for a named application. All defined promotion sites and promotion levels are included in the scope of this function. If no promotion sites are defined, no results are returned in the reply message.

The Serena XML service/scope/message tags and attributes for messages to *list* promotion site configuration records are:

```
<service name=”PROMLIB”>
<scope name=”SITE”>
<message name=”LIST”>
```

These tags appear in both requests and replies.

PROMLIB SITE LIST — Requests

Serena XML can request an application's promotion site configuration records for one or all promotion levels defined for an application. Records can be requested for an individual named site or for all sites.

Data structure details for the `<request>` data element appear in *Exhibit 10-5*.

Exhibit 10-5. PROMLIB SITE LIST<request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	1	String (4), variable	ZMF application name for which promotion site records are requested. ***NOTE:*** OK to omit trailing blanks.
<promotionLevel>	Optional	0 - 1	Integer (2), fixed	Numeric relative promotion level for which site records are requested. ***NOTE:*** Corresponds to value in `<promotionName>`. ***NOTE:*** Leading zeroes required. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all promotion levels.
<promotionName>	Optional	0 - 1	String (8), variable	ZMF nickname for promotion level for which site records are requested. ***NOTE:*** Corresponds to value in `<promotionLevel>`. ***NOTE:*** Use asterisk (*) wildcard or omit to include all promotion levels.
<siteName>	Optional	0 - 1	String (8), variable	ZMF name of promotion site desired. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all sites for application.

PROMLIB SITE LIST — Replies

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` tag contains site configuration information for one promotion library at one site and level associated with the application named in the request.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Data structure details for the `<result>` tag appear in *Exhibit 10-6*.

Exhibit 10-6. PROMLIB SITE LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	1	String (4), variable	ZMF application name.
<forcePriorSiteDemotion>	Optional	0 - 1	String (1)	Y = Force demotion from prior promotion level before promoting to this library. N = Don't force demotion from prior promotion level before promoting to this library.
<localInternalReaderClass>	Optional	0 - 1	String (1)	Output class for JCL promote & demote jobs submitted to internal reader on the local LPAR where ZMF runs.
<promoBuildProc>	Optional	0 -1	String (8), variable	Name of promotion build procedure defined for this application, site, and promotion level.
<promoEntity>	Optional	0 -1	String (8), variable	TSO ID of security entity that promotes application packages to this site and promotion level.
<promotionLevel>	Optional	0 - 1	Integer (2), fixed	Numeric relative promotion level listed promotion library.
<promotionName>	Optional	0 - 1	String (8), variable	ZMF nickname for promotion level in `<promotionLevel>`.
<siteInternalReaderClass>	Optional	0 - 1	String (1)	Output class for JCL promote & demote jobs submitted to internal reader at this remote site.
<siteName>	Optional	1	String (8), variable	ZMF name of promotion site listed.

List Production Library Datasets - PRODLIB SERVICE LIST

This function returns a list of fully qualified dataset names for all production libraries used with a named application. All defined production sites and library types are included in the scope of this function. If no production libraries or sites are defined, no results are returned in the reply message.

The Serena XML service/scope/message tags and attributes for messages to *list* production library records are:

```
<service name="PRODLIB">
<scope name="SERVICE">
<message name="LIST">
```

These tags appear in both requests and replies.

PRODLIB SERVICE LIST — Requests

Serena XML can request production library dataset names for one or all production sites and for one or all library types associated with a named application. Data structure details for the `<request>` data element appear in *Exhibit 10-7*.

Exhibit 10-7. Production Library Dataset List <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	1	String (4), variable	ZMF application name for which production datasets are requested. ***NOTE:*** OK to omit trailing blanks.
<libType>	Optional	0 - 1	String (3), variable	Library type for which production datasets are requested. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all library types.
<siteName>	Optional	0 - 1	String (8), variable	ZMF name of site where production library resides. ***NOTE:*** Use asterisk (*) wildcard or omit tag to include all production sites.

PRODLIB SERVICE LIST — Replies

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` tag contains the fully qualified dataset name of a production library which allows a specific library type at a specific production site. Also returned are the dataset names for associated temporary and backup libraries.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Data structure details for the `<result>` tag appear in *Exhibit 10-8*.

Exhibit 10-8. PRODLIB SERVICE LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	1	String (4), variable	ZMF application name supported by the production library.
<backupLib>	Optional	0 - 1	String (44), variable	Fully qualified dataset name of the backup library for this production library.

<hfsLib>	Optional	0 - 1	String (1)	Does promotion library reside in the z/OS Unix Hierarchical File System?
				Y = Yes, HFS promotion library
				N = No, not HFS library

Exhibit 10-8. PRODLIB SERVICE LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<libType>	Optional	0 - 1	String (3), variable	Library type supported by this production library.
<prodLib>	Optional	0 - 1	String (44), variable	Full qualified dataset name for this production library.
<siteName>	Optional	0 - 1	String (8), variable	ZMF name of site where this production library resides.
<tempLib>	Optional	0 - 1	String (44), variable	Fully qualified dataset name of the temporary library used with this production library.

SITE ADMINISTRATION

Serena XML supports the following remote site administration tasks for general use:

- *List Globally Defined Remote Sites - SITE GBL LIST*
- *List Remote Sites for Application - SITE APPL LIST*
- *List Install Calendar for Site - CALENDAR SERVICE LIST*

The syntax that identifies these functions appears in the `name` attribute of the `<service>` tag, as follows:

```
<service name="SITE">
<service name="CALENDAR">
```

The site-oriented scope of the installation calendar service is implicit rather than explicit.

List Globally Defined Remote Sites - SITE GBL LIST

This function lists all globally defined remote sites known to the current instance of ChangeMan ZMF. Also returned are global specifications for the z/OS logical unit and ChangeMan ZMF subsystem ID associated with that site, the staging library model for the site, the delay library, and site settings for peg dates, file transfer methods, and job cards. If no sites have been defined, no `<result>` tags are returned in the reply message.

The Serena XML service/scope/message tags and attributes for messages to *list* globally defined remote sites are:

```
<service name="SITE">
<scope name="GBL">
<message name="LIST">
```

These tags appear in both requests and replies.

SITE GBL LIST — Requests

This Serena XML function accepts a single, optional tag in the `<request>` date element of the request message. That tag, `<siteName>`, is used when global site specifications are requested for a single site. To request a list of specifications for all sites, enter a “match-all” wildcard character in this tag or omit it altogether.

If you omit the `<siteName>` tag from the request message, you must still include an empty `<request>` tag in your request message. The `<request>` tag is required to identify the message as a request rather than a reply.

Note

XML syntax allows both a long form and a short form for empty tags. An empty `<request>` tag can therefore be coded in one of two ways.

Long form:

```
<request>
</request>
```

Equivalent short form:

```
<request/>
```

Data structure details for the `<request>` data element appear in *Exhibit 10-9*.

Exhibit 10-9. SITE GBL LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<siteName>	Optional	0 - 1	String (8), variable	ZMF name of remote site. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all sites.

SITE GBL LIST — Replies

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` tag contains global specifications for one remote site known to the queried instance of ChangeMan ZMF.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Data structure details for the `<result>` tag appear in *Exhibit 10-10*.

Exhibit 10-10. SITE GBL LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<cmnSubSystemId>	Optional	0 - 1	String (1)	Subsystem ID of ZMF started task on remote site.
<defaultUnitName>	Optional	0 - 1	String (8), variable	Default DASD unit name assigned to site partition on remote site.

Exhibit 10-10. SITE GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<defaultVolume>	Optional	0 - 1	String (6), variable	Default DASD volume assigned to site partition on remote site.
<delayLib>	Optional	0 - 1	String (255), variable	Fully qualified dataset name of temporary library that receives ZMF data transfers to site.
<displayOrderNo>	Optional	0 - 1	Integer (undefined)	Display order number. This numeric value dictates the default order in which a list of items is displayed.
<gregorianPegDate>	Optional	0 - 1	Date (8), yyyymmdd	Peg date expressed as Gregorian date. ***NOTE:*** Must be equivalent to value in `<julianPegDate>` tag.
<jesNodeName>	Optional	0 - 1	String (8), variable	Site JES node name.
<jobCard01>	Optional	0 - 1	String (72), variable	First of up to 4 default job cards for application installation at this site.
<jobCard02>	Optional	0 - 1	String (72), variable	Second of up to 4 default job cards for application installation at this site.
<jobCard03>	Optional	0 - 1	String (72), variable	Third of up to 4 default job cards for application installation at this site.
<jobCard04>	Optional	0 - 1	String (72), variable	Fourth of up to 4 default job cards for application installation at this site.
<julianPegDate>	Optional	0 - 1	String (7), yyyyddd	Peg date expressed in Julian day format. ***NOTE:*** Must be equivalent to value in `<gregorianPegDate>` tag.
<logicalUnitName>	Optional	0 - 1	String (8), variable	z/OS logical unit name associated with ZMF remote site.
<serparmDsn>	Optional	0 - 1	String (44)	SERPARM dataset name.
<siteName>	Optional	0 - 1	String (8), variable	ZMF name of remote site.
<siteServerIpAddress>	Optional	0 - 1	String (32), variable	Site server IP Address.
<siteServerPortID>	Optional	0 - 1	String (5), variable	Site server port id.
<siteStageLibModel>	Optional	0 - 1	String (32), variable	Name of staging library on local ZMF server to use as model for creation & setup of staging library at this remote site.
<siteTimeDifference>	Optional	0 - 1	String (5), ±hhss	Signed integer representing site time zone offset from Universal Time (UT) in hours & seconds.

Exhibit 10-10. SITE GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<xferVehicle>	Optional	0 - 1	String (1)	Code for mainframe-to-mainframe file transfer tool. Values: 1 = IEBCOPY 2 = Other (such as XCOM, BDT, or CONNECT:Direct) ***NOTE:*** Value of 1 (IEBCOPY) is valid only if DASD shared between development site & remote site.

List Remote Sites for Application - SITE APPL LIST

This function lists one or all remote sites defined for a named application in the current instance of ChangeMan ZMF. Also returned are the default job cards for installing application-specific packages at the listed site. If no sites have been defined for the named application, no `<result>` tags are returned in the reply message.

The Serena XML service/scope/message tags and attributes for messages to *list* remote sites for an application are:

```
<service name=”SITE”>
<scope name=”APL”>
<message name=”LIST”>
```

These tags appear in both requests and replies.

SITE APPL LIST — Requests

Serena XML lets you list one or all sites defined for an application. Only the application name is required as input to the request. Specify a site name to restrict results to application-specific information about a single site.

Data structure details for the `<request>` data element appear in *Exhibit 10-11*.

Exhibit 10-11. SITE APPL LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	1	String (4), variable	ZMF name of desired application. ***NOTE:*** OK to omit trailing blanks.
<siteName>	Optional	0 - 1	String (8), variable	ZMF name of desired site.

SITE APPL LIST — Replies

The reply message listing an application’s remote sites returns zero to many `<result>` data elements. Each `<result>` tag contains the name of one site defined for that application.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Data structure details for the `<result>` tag appear in *Exhibit 10-12*.

Exhibit 10-12. SITE APPL LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	1	String (4), variable	ZMF application name.
<displayOrderNo>	Optional	0 - 1	Integer (undefined)	Display order number. This numeric value dictates the default order in which a list of items is displayed.
<jobCard01>	Optional	0 - 1	String (72), variable	First of up to 4 default job cards for application installation at this site.
<jobCard02>	Optional	0 - 1	String (72), variable	Second of up to 4 default job cards for application installation at this site.
<jobCard03>	Optional	0 - 1	String (72), variable	Third of up to 4 default job cards for application installation at this site.
<jobCard04>	Optional	0 - 1	String (72), variable	Fourth of up to 4 default job cards for application installation at this site.
<jobCards>	Optional	0 - 4	String (72), variable	Use tags jobCard01 - joCard04 instead.
<siteName>	Required	1	String (8), variable	ZMF name of one site associated with named application.

List Install Calendar for Site - CALENDAR SERVICE LIST

This function lists the package installation schedule for one or all sites defined to ChangeMan ZMF. If no installs are scheduled for the next 364 days, no results are returned.

Note

Systemwide calendar and scheduler settings are managed via global parameter settings. See *List Global Parameters - PARMS GBL LIST* for more information about global calendar & scheduler settings.

The Serena XML service/scope/message tags and attributes for messages to *list* the installation calendar for one or more sites are:

```
<service name="CALENDAR">
<scope name="SERVICE">
<message name="LIST">
```

These tags appear in both requests and replies.

CALENDAR SERVICE LIST — Requests

Serena XML supports two types of installation calender *list* requests:

- ***Site Calendar*** — Request the install calendar for one site by entering the site name in the `<siteName>` tag of the `<request>` data element. The number of package installs for that site are returned by date for the next 364 days.
- ***Composite Calendar*** — Request a composite install calendar for all sites by entering a match-all (asterisk) wildcard in the `<siteName>` tag or omitting the tag altogether. The total number of package installs for the organization are returned by date for the next 364 days. No site breakout is included.

Data structure details for the `<request>` data element appear in *Exhibit 10-13*.

Exhibit 10-13. CALENDAR SERVICE LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<siteName>	Optional	0 - 1	String (8), variable	ZMF site name. ***NOTE:*** Use asterisk (*) wildcard character or omit tag to request schedule for all sites, with no site breakdown.

CALENDAR SERVICE LIST — Replies

The reply message for this function returns one `<result>` data element, which contains up to 364 scheduling calendar records. Each record totals the planned number of package install jobs for the site named in the request message, or for the entire organization if no site was named.

The standard `<response>` data element follows the `<result>` tag in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Data structure details for the `<result>` tag appear in *Exhibit 10-14*.

Exhibit 10-14. CALENDAR SERVICE LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<calendarEntry>	Optional	0 - 364	Complex	See *Exhibit 10-15*.

The `<result>` data element contains a complex subtag, `<calendarEntry>`, which itself contains several subtags. Data structure details appear in *Exhibit 10-15*.

Exhibit 10-15. <calendarEntry> Subtag Data Structure

Seq	Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
1	<calendarDate>	Required	1	Date, yyyymmdd	Date for install jobs to run.
2	<numberOfDays>	Required	1	Integer (4), variable	Number of days
3	<scheduledInstallCount>	Required	1	Integer (3), variable	Number of package install jobs scheduled for date.
4	<maximumInstallCount>	Required	1	String (3), variable	Maximum number of install jobs set for date by ZMF administrator. ***NOTE:*** An unlimited number of packages may be installed if this value = 'UNL'.

Developer Environment Administration

For administrators of the ChangeMan ZMF developer environment, Serena XML supports the following tasks for general use:

- *List Global Library Types - LIBTYPE GBL LIST*
- *List Application Library Types - LIBTYPE APL LIST*
- *List Global Language Parsers - LANGUAGE GBL LIST*
- *List Application Language Parsers - LANGUAGE APL LIST*
- *List Global Build Procedures - PROCS GBL LIST*
- *List Application Build Procedures - PROCS APL LIST*
- *List Global Parameters - PARMS GBL LIST*
- *Parameters Application List - PARMS APL LIST*
- *List Global Reason Codes - REASONS SERVICE LIST*

The syntax that identifies these functions appears primarily in the `name` attribute of the `<service>` tag, as follows:

```
<service name="LIBTYPE">
<service name="LANGUAGE">
<service name="PROCS">
<service name="PARMS">
<service name="REASONS">
```

In addition, the `name` attribute of the `<scope>` tag is one of the following:

```
<service name="GBL">
<service name="APL">
```

List Global Library Types - LIBTYPE GBL LIST

LIBTYPE GBL LIST

This Serena XML function requests a list of all globally defined library types known to this instance of ChangeMan ZMF. Also returned with the library type is its “like-library” type, library type description, library-level options for staging versions and DBMS support, and dataset configuration settings.

The Serena XML service/scope/message tags and attributes for messages to *list* the globally library types are:

```
<service name="LIBTYPE">
<scope name="GBL">
<message name="LIST">
```

These tags appear in both requests and replies.

LIBTYPE GBL LIST — Request

This function supports four kinds of library type list requests:

- ***All Library Types*** — Enter a “match-all” (asterisk) wild card in `<libType>` or omit this tag altogether to retrieve all globally defined library types and their specifications.
- ***All “Like-Library” Library Types*** — Lists all physical library types assigned to the “like-library” category identified by code in the `<likeType>` tag of your request.
- ***All DB2 Library Types*** — To list all DB2 library types, enter “Y” in the `<isDb2LibType>` tag of your request.
- ***ChangeMan ZMF Settings for a Named Library Type*** — Lists the “like-library” type, library type description, library-level support options, and dataset configuration for the library type named in the `<libType>` tag of your request.

If you omit the `<libType>` tag from your request, you must still submit an empty `<request>` data element to identify the message as a request rather than a reply.

Note

XML syntax allows both a long form and a short form for empty tags. An empty `<request>` tag can therefore be coded in one of two ways.

Long form:

```
<request>
</request>
```

Equivalent short form:

```
<request/>
```

Data structure details for the `<request>` data element appear in *Exhibit 10-16*.

Exhibit 10-16. LIBTYPE GBL LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies

<isDb2LibType>	Optional	0 - 1	String (1)	Y = Include only DB2 libraries.
				N = Omit all DB2 libraries.
				NOTE: Omit tag or use asterisk (*) wildcard to request both DB2 and non-DB2 library types.

<libType>	Optional	0 - 1	String (3), variable	Name of physical library type. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all library types.
<likeType>	Optional	0 - 1	String (1)	Code for “like-library” type assigned to library type name. Values: 1 = Like Copy Library 2 = Like Load Library 3 = Like Other Library 4 = Like PDS Library 5 = Like Source Library 6 = Like Ncal Library 7 = Like Object Library

Exhibit 10-16. LIBTYPE GBL LIST <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<userFunction>	Optional	0 - 1	String (1), variable	User function related to the service request. This modifies the behavior of the service. Values: 1 - Browse 2 - Checkout 3 - Create 4 - Delete 5 - Edit 6 - Edit and stage 7 - Recompile 8 - Relink 9 - Stage A - Update B - Checkin C - Build D - Browse listing E - Compare F - Scan G - Scratch/rename

LIBTYPE GBL LIST — Reply

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` tag contains the name and specifications for one globally defined library type.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Data structure details for the `<result>` tag appear in *Exhibit 10-17*.

Exhibit 10-17. LIBTYPE GBL LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<apsDevLib>	Optional	0 - 1	String (44), variable	Fully qualified dataset name of APS development library associated with library type named in `<libType>`. ***NOTE:*** Required if `<isApsLibType>` value is Y.
<apsEntity>	Optional	0 - 1	String (8), variable	Name of APS security entity used to access APS development library. ***NOTE:*** Required if `<isApsLibType>` value is Y.
<bindControlSubType>	Optional	0 - 1	String (1)	**Y** = Yes, DB2 bind ctrl plan subtype **N** = Not DB2 bind ctrl plan subtype ***NOTE:*** Required if `<isDb2LibType>` value is Y.

Exhibit 10-17. LIBTYPE GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<blockSize>	Optional	0 - 1	String (6), variable	Size of DASD block in bytes to use with staging library allocations for library type named in `<libType>`.
<chkOutActivityFile>	Optional	0 - 1	String (1)	**Y** = Yes, copy component to activity file at checkout. **N** = Don't make activity file copy.
<chkOutComponentGenDesc>	Optional	0 - 1	String (1)	**Y** = Yes, copy component general description to staging change description at check-out. **N** = No, leave component change description blank in staging at check-out.
<db2SqlTerminationChar>	Optional	0 - 1	String (1)	DB2 SQL sentence termination character.
<dbrmSubType>	Optional	0 - 1	String (1)	**Y** = Yes, DBRM subtype **N** = Not DBRM subtype
<ddlSqlSubType>	Optional	0 - 1	String (1)	**Y** = Yes, DB2 DDL/SQL subtype **N** = Not DB2 DDL/SQL subtype ***NOTE:*** Required if `<isDb2LibType>` value is Y.
<deferStageLibCreation>	Optional	0 - 1	String (1)	**Y** = Yes, defer allocation of library type in staging library until first component check-out to library type in `<libType>`. **N** = No, don't defer library type creation, even if empty.
<dirBlocks>	Optional	0 - 1	String (6), variable	Blocks allocated in staging library to directory for this library type. ***NOTE:*** Size of block defined in `<blockSize>` tag.
<displayOrderNo>	Optional	0 - 1	Integer (undefined)	Display order number. This numeric value dictates the default order in which a list of items is displayed.
<eAttr>	Optional	0 - 1	String (1)	Extended attribute option. Values: **N** = Dataset cannot have extended attributes or reside in EAS. **O** = Dataset can have extended attributes and reside in EAS. **blank** = Default based on data type.

Exhibit 10-17. LIBTYPE GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<imsEntity>	Optional	0 - 1	String (1)	IMS entity class for library type, if applicable. Values: **1** = PSB source **2** = DBD source 3 = MFS source 4 = PSB target 5 = DBD target 6 = FMT target 7 = REF target ***NOTE:*** Required if `<isImsLibType>` value is Y.
<includeUtilityInfo>	Optional	0 - 1	String (1)	**Y** = Yes, track scratch/rename utility activity for library type in `<libType>`. **N** = No, omit scratch/rename activity.
<isApsLibType>	Optional	0 - 1	String (1)	**Y** = Yes, library type is APS **N** = Not APS
<isDb2LibType>	Optional	0 - 1	String (1)	**Y** = Yes, library type is DB2 **N** = Not DB2
<isHfsLibType>	Optional	0 - 1	String (1)	**Y** = Yes, this is HFS library type **N** = No, not HFS library type
<isImsLibType>	Optional	0 - 1	String (1)	**Y** = Yes, library type is IMS **N** = Not IMS
<isPdsLibType>	Optional	0 - 1	String (1)	**Y** = Yes, library type is PDS **N** = Not PDSE
<isPdseLibType>	Optional	0 - 1	String (1)	**Y** = Yes, library type is PDSE **N** = Not PDSE
<isPdseObject>	Optional	0 - 1	String (1)	**Y** = Yes, library type is PDSE Object **N** = Not PDSE Object
<isSsvAllowed>	Optional	0 - 1	String (1)	**Y** = Yes, SSV allowed **N** = No, SSV is not allowed
<isSsvEnforced/>	Optional	0 - 1	String (1)	**Y** = Yes, SSV enforced **N** = No, SSV is not enforced
<isSysManaged/>	Optional	0 - 1	String (1)	**Y** = Yes, System Managed **N** = No, not System Managed
<libType>	Optional	0 - 1	String (3), variable	Name of physical library type.
<libTypeDesc>	Optional	0 - 1	String (44), variable	Global description of library type.
<librarySequenceNo>	Optional	0 - 1	Integer (1), fixed	Syslib Concatenation Seq. No.

Exhibit 10-17. LIBTYPE GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<likeType>	Optional	0 - 1	String (1)	Code for "like-library" type assigned to library type name. Values: 1 = Like Copy Library 2 = Like Load Library 3 = Like Other Library 4 = Like PDS Library 5 = Like Source Library 6 = Like Ncal Library 7 = Like Object Library
<packagBindControlSubType>	Optional	0 - 1	String (1)	**Y** = Yes, DB2 pkg bind control subtype **N** = Not DB2 pkg bind control subtype ***NOTE:*** Required if `<isDb2LibType>` value is Y.
<primarySpace>	Optional	0 - 1	String (8), variable	Minimum DASD space allocation in staging library for this library type. ***NOTE:*** Given in units defined by `<spaceType>` tag.
<recordFormat>	Required	1	String(3), variable	Code for logical record format. Values: **F** = Fixed **FA** = Fixed ASA **FM** = Fixed Machine **FB** = Fixed Block **FBA** = Fixed Block ASA **FBM** = Fixed Block Machine **FBS** = Fixed Block Standard **FS** = Fixed Standard **FSA** = Fixed Standard ASA **FSM** = Fixed Standard Machine **V** = Variable **VA** = Variable ASA **VM** = Variable Machine **VB** = Variable Block **VBA** = Variable Block ASA **VBM** = Variable Block Machine **VS** = Variable Spanned **VSA** = Variable Spanned ASA **VSM** =Variable Spanned Machine **U** = Undefined **UA** = Undefined ASA **UM** = Undefined Machine **UB** = Undefined Block **UBA** = Undefined Block ASA **UBM** = Undefined Block Machine **US** = Undefined Spanned **USA** = Undefined Spanned ASA **USM** =Undefined Spanned Machine
<recordLength>	Optional	0 - 1	String (6), variable	Length of data set record in bytes.

Exhibit 10-17. LIBTYPE GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<secondarySpace>	Optional	0 - 1	String (8), variable	DASD allocation for extents in staging library for this library type. ***NOTE:*** Given in units defined by `<spaceType>` tag.

<spaceType>	Optional	0 - 1	String (3), variable	DASD space allocation unit in staging library for this library type. Values:
				Blk = Blocks
				Cyl = Cylinders
				Trk = Tracks

<sqlStoredProcDefinition>	Optional	0 - 1	String (1)	**Y** = Yes, DB2 SQL stored procedure definition subtype **N** = Not DB2 SQL stored procedure ***NOTE:*** Required if `<isDb2LibType>` value is Y.
<ssvOption>	Optional	0 - 1	String (8), variable	Global option to save staging versions for library type named in `<libType>`. **YES** = Save staging versions. **NO** = Don't save staging versions. **OPT** = Optional; save staging versions enabled by application.
<storedProcSubtype>	Optional	0 - 1	String (1)	**Y** = Yes, DB2 stored procedure subtype **N** = Not DB2 stored procedure subtype ***NOTE:*** Required if `<isDb2LibType>` value is Y.
<targetActivityFile>	Optional	0 - 1	String (3), variable	Library type for check-out activity file associated with the library type named in `<libType>`.
<targetLoadLibtype>	Optional	0 - 1	String (3), variable	The "like-load" target library type associated with source library type in `<libType>`. ***NOTE:*** Required if library type is like-source, i.e. `<likeType>` value = *5*.
<triggerSubType>	Optional	0 - 1	String (1)	**Y** = Yes, DB2 trigger subtype **N** = Not DB2 trigger subtype ***NOTE:*** Required if `<isDb2LibType>` value is Y.
<unitName>	Optional	0 - 1	String (8), variable	Logical unit name for DASD volume assigned to library type.
<volume>	Optional	0 - 1	String (6), variable	DASD reference volume serial ID.

List Application Library Types - LIBTYPE APL LIST

This function requests a list of all library types defined for a named application. Also returned with the library type is its "like-library" type, library type description, library-level options for staging versions and DBMS support, and dataset configuration settings. Values not overridden at the application level are not returned in the results of this function. The global defaults used in place of such overrides are outside the scope of this function.

The Serena XML service/scope/message tags and attributes for messages to *list* the library type records for an application are:

```
<service name="LIBTYPE">
<scope name="APL">
<message name="LIST">
```

These tags appear in both requests and replies.

LIBTYPE APL LIST — Requests

This Serena XML function requests a list of library types defined for use with a named application. The application name is required in the request. The function supports four kinds of requests:

- ***All Library Types*** — Name the application for which library types and their specifications are desired in the `<applName>` tag. Enter a "match-all" (asterisk) wild card in `<libType>` or omit this tag altogether. This retrieves a list of all library types defined for the application.
- ***All "Like-Library" Library Types*** — Name the application for which library types and their specifications are desired in the `<applName>` tag. Enter a code corresponding to the "like-library" type of interest in the <likeType> tag. This requests a list of all physical library types assigned to the "like-library" category identified in the `<likeType>` tag and defined for the named application.
- ***All DB2 Library Types*** — Name the application for which library types and their specifications are desired in the `<applName>` tag. Enter "Y" in the `<isDb2LibType>` tag. This requests all DB2 library types and specifications defined for the named application.
- ***ChangeMan ZMF Settings for a Named Library Type*** — Name the application for which library type specifications are desired in the `<applName>` tag. Name the particular library type of interest in `<libType>`. The function lists the "like-library" type, library type description, library-level support options, and dataset configuration for the named library type as they are defined for the named application.

Data structure details for the `<request>` data element appear in *Exhibit 10-18*.

Exhibit 10-18. LIBTYPE APL LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	1	String (4), variable	ZMF application name. ***NOTE:*** OK to omit trailing blanks.

Exhibit 10-18. LIBTYPE APL LIST <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies

<isDb2LibType>	Optional	0 - 1	String (1)	Y = Include only DB2 libraries.
				N = Omit all DB2 libraries.
				NOTE: Omit tag or use asterisk (*) wildcard to request both DB2 and non-DB2 library types.

<libType>	Optional	0 - 1	String (3), variable	Name of desired library type. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all library types.
<likeLibType>	Optional	0 - 1	String (1)	Code for "like-library" type assigned to library type name. Values: 1 = Like Copy Library 2 = Like Load Library 3 = Like Other Library 4 = Like PDS Library 5 = Like Source Library 6 = Like Ncal Library 7 = Like Object Library
<userFunction>	Optional	0 - 1	String (1), variable	User function related to the service request. This modifies the behavior of the service. Values: 1 - Browse 2 - Checkout 3 - Create 4 - Delete 5 - Edit 6 - Edit and stage 7 - Recompile 8 - Relink 9 - Stage A - Update B - Checkin C - Build D - Browse listing E - Compare F - Scan G - Scratch/rename

LIBTYPE APL LIST — Replies

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` tag contains information about one library type defined for the named application.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Data structure details for the `<result>` tag appear in *Exhibit 10-19*.

Exhibit 10-19. LIBTYPE APL LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), variable	ZMF application name. ***NOTE:*** OK to omit trailing blanks.
<apsDevLib>	Optional	0 - 1	String (44), variable	Fully qualified dataset name of APS development library associated with library type named in `<libType>`. ***NOTE:*** Required if `<isApsLibType>` value is Y.
<apsEntity>	Optional	0 - 1	String (8), variable	Name of APS security entity used to access APS development library. ***NOTE:*** Required if `<isApsLibType>` value is Y.
<blockSize>	Optional	0 - 1	String (6), variable	Size of DASD block in bytes to use with staging library allocations for library type named in `<libType>`.
<chkOutActivityFile>	Optional	0 - 1	String (1)	**Y** = Yes, copy component to activity file at checkout. **N** = Don't make activity file copy.
<chkOutComponentGenDesc>	Optional	0 - 1	String (1)	**Y** = Yes, copy component general description to staging change description at check-out. **N** = No, leave component change description blank in staging at check-out.
<db2SqlTerminationChar>	Optional	0 - 1	String (1)	DB2 SQL sentence termination character.
<dbrmSubType>	Optional	0 - 1	String (1)	**Y** = Yes, DBRM subtype **N** = Not DBRM subtype
<ddlSqlSubType>	Optional	0 - 1	String (1)	**Y** = Yes, DB2 DDL/SQL subtype **N** = Not DB2 DDL/SQL subtype ***NOTE:*** Required if `<isDb2LibType>` value is Y.
<deferStageLibCreation>	Optional	0 - 1	String (1)	**Y** = Yes, defer allocation of library type in staging library until first component check-out to library type in `<libType>`. **N** = No, don't defer library type creation, even if empty.
<dirBlocks>	Optional	0 - 1	String (6), variable	Blocks allocated in staging library to directory for this library type. ***NOTE:*** Size of block defined in `<blockSize>` tag.

Exhibit 10-19. LIBTYPE APL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<displayOrderNo>	Optional	0 - 1	Integer (undefined)	Display order number. This numeric value dictates the default order in which a list of items is displayed.
<eAttr>	Optional	0 - 1	String (1)	Extended attribute option. Values: **N** = Dataset cannot have extended attributes or reside in EAS. **O** = Dataset can have extended attributes and reside in EAS. **blank** = Default based on data type.
<imsEntity>	Optional	0 - 1	String (1)	IMS entity class for library type, if applicable. Values: **1** = PSB source **2** = DBD source **3** = MFS source **4** = PSB target **5** = DBD target **6** = FMT target **7** = REF target ***NOTE:*** Required if `<isImsLibType>` value is Y.
<includeUtilityInfo>	Optional	0 - 1	String (1)	**Y** = Yes, track scratch/rename utility activity for library type in `<libType>`. **N** = No, omit scratch/rename activity.
<includeUtilityInfo>	Optional	0 - 1	String (1)	**Y** = Include utility comp info. **N** = Do not include utility comp info.
<isApsLibType>	Optional	0 - 1	String (1)	**Y** = Yes, library type is APS **N** = Not APS
<isDb2LibType>	Optional	0 - 1	String (1)	**Y** = Yes, library type is DB2 **N** = Not DB2
<isHfsLibType>	Optional	0 - 1	String (1)	**Y** = Yes, this is HFS library type **N** = No, not HFS library type
<isImsLibType>	Optional	0 - 1	String (1)	**Y** = Yes, library type is IMS **N** = Not IMS
<isPdsLibType>	Optional	0 - 1	String (1)	**Y** = Yes, library type is PDS **N** = Not PDS
<isPdseLibType>	Optional	0 - 1	String (1)	**Y** = Yes, library type is PDSE **N** = Not PDSE
<isPdseObject>	Optional	0 - 1	String (1)	**Y** = PDSE Object lib **N** = Not PDSE Object lib
<isSsvAllowed>	Optional	0 - 1	String (1)	**Y** = SSV allowed **N** = SSV not allowed

Exhibit 10-19. LIBTYPE APL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<isSsvEnforced>	Optional	0 - 1	String (1)	**Y** = SSV enforced **N** = SSV not enforced
<isSysManaged>	Optional	0 - 1	String (1)	**Y** = System Managed **N** = Not System Managed
<libType>	Optional	0 - 1	String (3), variable	Name of physical library type.
<libTypeDesc>	Optional	0 - 1	String (44), variable	Global description of library type.
<librarySequenceNo>	Optional	0 - 1	String (3)	SYSLIB concatenation sequence number.
<likeType>	Optional	0 - 1	String (1)	Code for “like-library” type assigned to library type name. Values: 1 = Like Copy Library 2 = Like Load Library 3 = Like Other Library 4 = Like PDS Library 5 = Like Source Library 6 = Like Ncal Library 7 = Like Object Library
<packagBindControlSubType>	Optional	0 - 1	String (1)	**Y** = Yes, DB2 pkg bind control subtype **N** = Not DB2 pkg bind control subtype ***NOTE:*** Required if `<isDb2LibType>` value is Y.
<planBindControlSubType>	Optional	0 - 1	String (1)	**Y** = Yes, DB2 bind ctrl plan subtype **N** = Not DB2 bind ctrl plan subtype ***NOTE:*** Required if `<isDb2LibType>` value is Y.
<primarySpace>	Optional	0 - 1	String (8), variable	Minimum DASD space allocation in staging library for this library type. ***NOTE:*** Given in units defined by `<spaceType>` tag.

Exhibit 10-19. LIBTYPE APL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<recordFormat>	Optional	0 - 2	String(3), variable	Code for logical record format. Values: **F** = Fixed **FA** = Fixed ASA **FM** = Fixed Machine **FB** = Fixed Block **FBA** = Fixed Block ASA **FBM** = Fixed Block Machine **FBS** = Fixed Block Standard **FS** = Fixed Standard **FSA** = Fixed Standard ASA **FSM** = Fixed Standard Machine **V** = Variable **VA** = Variable ASA **VM** = Variable Machine **VB** = Variable Block **VBA** = Variable Block ASA **VBM** = Variable Block Machine **VS** = Variable Spanned **VSA** = Variable Spanned ASA **VSM** =Variable Spanned Machine **U** = Undefined **UA** = Undefined ASA **UM** = Undefined Machine **UB** = Undefined Block **UBA** = Undefined Block ASA **UBM** = Undefined Block Machine **US** = Undefined Spanned **USA** = Undefined Spanned ASA **USM** =Undefined Spanned Machine
<recordLength>	Optional	0 - 1	String (6), variable	Length of data set record in bytes.
<secondarySpace>	Optional	0 - 1	String (8), variable	DASD allocation for extents in staging library for this library type. ***NOTE:*** Given in units defined by `<spaceType>` tag.

<spaceType>	Optional	0 - 1	String (3), variable	DASD space allocation unit in staging library for this library type. Values:
				Blk = Blocks
				Cyl = Cylinders
				Trk = Tracks

<sqlStoredProcDefinition>	Optional	0 - 1	String (1)	**Y** = Yes, DB2 SQL stored procedure definition subtype **N** = Not DB2 SQL stored procedure ***NOTE:*** Required if `<isDb2LibType>` value is Y.

Exhibit 10-19. LIBTYPE APL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<ssvOption>	Optional	0 - 1	String (8), variable	Application-level option to save staging versions for library type in `<libType>`. **Always** = Always save staging version **Prompt** = Prompt to save staging version **None** = No staging version option set; take global default
<storedProcSubtype>	Optional	0 - 1	String (1)	**Y** = Yes, DB2 stored procedure subtype **N** = Not DB2 stored procedure subtype ***NOTE:*** Required if `<isDb2LibType>` value is Y.
<targetActivityFile>	Optional	0 - 1	String (3), variable	Library type for check-out activity file associated with the library type named in `<libType>`.
<targetLoadLibtype>	Optional	0 - 1	String (3), variable	The "like-load" target library type associated with source library type in `<libType>`. ***NOTE:*** Required if library type is like-source, i.e. `<likeType>` value = *5*.
<triggerSubType>	Optional	0 - 1	String (1)	**Y** = Yes, DB2 trigger subtype **N** = Not DB2 trigger subtype ***NOTE:*** Required if `<isDb2LibType>` value is Y.
<unitName>	Optional	0 - 1	String (8), variable	Logical unit name for DASD volume assigned to library type.
<volume>	Optional	0 - 1	String (6), variable	DASD reference volume serial ID.

List Global Language Parsers - LANGUAGE GBL LIST

This function lists the parsers assigned to each globally defined programming language in ChangeMan ZMF. If no languages have been defined at the global level, no results are returned by this function.

The Serena XML service/scope/message tags and attributes for messages to *list* globally defined programming language parsers are:

```
<service name="LANGUAGE">
<scope name="GBL">
<message name="LIST">
```

These tags appear in both requests and replies.

LANGUAGE GBL LIST — Requests

The request message for this function identifies one or all globally defined programming languages whose parsers are of interest. To list all globally defined languages and their parsers, use a “match-all” (asterisk) wildcard character in the `<language>` tag, or omit this tag altogether and submit an empty `<request>`. Enter a specific language name to retrieve the parser for that language.

Note

XML syntax allows both a long form and a short form for empty tags. An empty `<request>` tag can therefore be coded in one of two ways.

Long form:

```
<request>
</request>
```

Equivalent short form:

```
<request/>
```

Data structure details for the `<request>` data element appear in *Exhibit 10-20*.

Exhibit 10-20. LANGUAGE GBL LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<language>	Optional	1	String (8), variable	Programming language desired. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request parsers for all languages.

LANGUAGE GBL LIST — Replies

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` tag contains one globally defined language and parser combination.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Data structure details for the `<result>` tag appear in *Exhibit 10-21*.

Exhibit 10-21. LANGUAGE GBL LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<displayOrderNo>	Optional	0 - 1	Integer (undefined)	Display order number. This numeric value dictates the default order in which a list of items is displayed.
<language>	Optional	0 - 1	String (8), variable	Name of programming language.

Exhibit 10-21. LANGUAGE GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<parser>	Optional	0 - 1	String (8), variable	Name of globally defined default language parsing routine.

List Application Language Parsers - LANGUAGE APL LIST

This function lists application-level parser overrides for the programming languages used in a named application. If no language parser is defined at the application level, no results are returned. Global parser defaults for languages without an application-level override are outside the scope of this function.

The Serena XML service/scope/message tags and attributes for messages to *list* the programming language parsers for an application are:

```
<service name="LANGUAGE">
<scope name="APL">
<message name="LIST">
```

These tags appear in both requests and replies.

LANGUAGE APL LIST — Requests

The request message for this function requires the name of the desired application in the `<applName>` tag. It additionally identifies one or all programming languages whose parsers are of interest. To list all languages with application overrides for their parsers, use a "match-all" (asterisk) wildcard character in the `<language>` tag, or omit this tag altogether. Enter a specific language name to retrieve the parser override for that language.

Data structure details for the `<request>` data element appear in *Exhibit 10-22*.

Exhibit 10-22. LANGUAGE APL LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	1	String (4), variable	ZMF application name desired. ***NOTE:*** OK to omit trailing blanks.
<language>	Optional	0 - 1	String (8), variable	Programming language desired. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all languages.

LANGUAGE APL LIST — Replies

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` tag contains information about one language and parser combination defined to override global defaults for the named application.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Data structure details for the `<result>` tag appear in *Exhibit 10-23*.

Exhibit 10-23. LANGUAGE APL LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), variable	ZMF application name.
<displayOrderNo>	Optional	0 - 1	Integer (undefined)	Display order number. This numeric value dictates the default order in which a list of items is displayed.
<language>	Optional	0 - 1	String (8), variable	Name of programming language with application-level parser override.
<parser>	Optional	0 - 1	String (8), variable	Name of language parsing routine defined for application.

List Global Build Procedures - PROCS GBL LIST

This function lists globally defined build procedures, their descriptions, and their associated programming languages and parsers. If no build procedures are defined at the global level, no results are returned.

The Serena XML service/scope/message tags and attributes for messages to *list* globally defined build procedures are:

```
<service name="PROCS">
<scope name="GBL">
<message name="LIST">
```

These tags appear in both requests and replies.

PROCS GBL LIST — Requests

Request messages for the global build procedure list take one of three forms:

- ***All Build Procedures*** — To request all globally defined build procedures, enter a “match-all” (asterisk) wildcard character in both the `<procName>` and `<language>` tags. Alternatively, omit both tags and submit and empty `<request>` data structure.
- ***Build Procedures Defined for Named Language*** — Name the language of interest in the `<language>` tag to request all build procedures that support that language.
- ***Language Defined for Named Build Procedure*** — Name the build procedure of interest in the `<procName>` tag to request its description, language, and parser definition.

Note

XML syntax allows both a long form and a short form for empty tags. An empty `<request>` tag can therefore be coded in one of two ways.

Long form:

```
<request>
</request>
```

Equivalent short form:

```
<request/>
```

Data structure details for the `<request>` data element appear in *Exhibit 10-24*.

Exhibit 10-24. PROCS GBL LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<language>	Optional	0 - 1	String (8), variable	Programming language desired. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all languages.
<procName>	Optional	0 - 1	String (8), variable	Name of build procedure desired. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all build procedures.

PROCS GBL LIST — Replies

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` tag contains information about one globally defined build procedure.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Data structure details for the `<result>` tag appear in *Exhibit 10-25*.

Exhibit 10-25. PROCS GBL LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<displayOrderNo>	Optional	0 - 1	String (1)	This numeric value dictates the default order in which a list of items is displayed.
<language>	Optional	0 - 1	String (8), variable	Name of programming language supported by build procedure in `<procName>`.
<parser>	Optional	0 - 1	String (8), variable	Name of language parsing routine defined for build procedure in `<procName>`.
<procDesc>	Optional	0 - 1	String (44), variable	Global description of build procedure named in `<procName>`.

Exhibit 10-25. PROCS GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<procName>	Optional	0 - 1	String (8), variable	Name of globally defined build procedure.

List Application Build Procedures - PROCS APL LIST

This function lists application build procedures, their descriptions, and their associated programming languages and parsers. If no build procedures are defined at the application level, no results are returned. Global build procedure defaults are outside the scope of this function.

The Serena XML service/scope/message tags and attributes for messages to *list* application build procedures are:

```
<service name=”PROCS”>
<scope name=”APL”>
<message name=”LIST”>
```

These tags appear in both requests and replies.

PROCS APL LIST — Requests

Request messages for the application build procedure list take one of three forms:

- ***All Build Procedures*** — To request all build procedures defined for an application, name the desired application in the `<applName>` tag and omit all other tags from the request.
- ***Build Procedures Defined for Named Language*** — Name the application of interest in the `<applName>` tag and the language of interest in the `<language>` tag to request all application build procedures that support that language.
- ***Language Defined for Named Build Procedure*** — Name the application of interest in the `<applName>` tag and the build procedure of interest in the `<procName>` tag to request the description, language, and parser definition for that build procedure.

Data structure details for the `<request>` data element appear in *Exhibit 10-26*.

Exhibit 10-26. PROCS APL LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	1	String (4), variable	ZMF application name desired. ***NOTE:*** OK to omit trailing blanks.
<language>	Optional	0 - 1	String (8), variable	Programming language desired. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all languages.
<procName>	Optional	0 - 1	String (8), variable	Name of build procedure desired. ***NOTE:*** Use asterisk (*) wildcard or omit tag to request all build procedures.

PROCS APL LIST — Replies

The reply message for this function returns zero to many `<result>` data elements. Each `<result>` tag contains information about one application build procedure.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Data structure details for the `<result>` tag appear in *Exhibit 10-27*.

Exhibit 10-27. PROCS APL LIST<result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	0 - 1	String (4), variable	Name of ZMF application listed.
<displayOrderNo>	Optional	0 - 1	String (1)	This numeric value dictates the default order in which a list of items is displayed.
<language>	Optional	0 - 1	String (8), variable	Name of programming language supported by build procedure in `<procName>`.
<parser>	Optional	0 - 1	String (8), variable	Name of language parsing routine defined for build procedure in `<procName>`.
<procDesc>	Optional	0 - 1	String (44), variable	Application-level description of build procedure named in `<procName>`.
<procName>	Optional	0 - 1	String (8), variable	Name of application-level build procedure.

List Global Parameters - PARMS GBL LIST

The global parameter list reports global settings in ChangeMan ZMF that enforce the business rules for software change management at your installation. If you have not customized your global parameters, the default values are reported.

The Serena XML service/scope/message tags and attributes for messages to *list* global parameters are:

```
<service name="PARMS">
<scope name="GBL">
<message name="LIST">
```

These tags appear in both requests and replies.

PARMS GBL LIST — Request

Global parameter list requests take no parameters and contain an empty `<request>` data element. The `<request>` tag is required to identify the message as a request rather than a reply.

Note

XML syntax allows both a long form and a short form for empty tags. An empty `<request>` tag can therefore be coded in one of two ways.

Long form:
```
<request>
</request>
```

Equivalent short form:
```
<request/>
```

PARMS GBL LIST — Reply

The reply message for this function exactly one `<result>` data element containing the global parameter settings for this instance of ChangeMan ZMF.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Data structure details for the `<result>` tag appear in *Exhibit 10-28*.

Exhibit 10-28. PARMS GBL LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<addUsrVarsToPkgLst>	Optional	0 - 1	String (1)	Y = Yes, add user variables to pkg list. N = No, Don't add user variables to pkg list.
<allocRetryCount>	Optional	0 - 1	String (6), variable	The number of times to retry failed allocation attempts. Failed allocation attempts are retried when generating package installation JCL. Value range: 0 to 65535.
<allocRetryWait>	Optional	0 - 1	String (6), variable	The time in seconds to wait between allocation retry attempts. Value range: 0 to 65535.

<allowAssignedProcOverride>	Optional	0 - 1	String (1)	Y = Yes, allow designated PROC override.
				N = Don't allow designated PROC override.

<allowChkOutToDevLib>	Optional	0 - 1	String (1)	Y = Yes, allow checkout to personal development library outside ZMF. N = Don't allow checkout to personal development library outside ZMF.

Exhibit 10-28. PARMS GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<allowCompMultAppl>	Optional	0 - 1	String (1)	Y = Yes, allow component in multiple applications. N = Don't allow component in multiple applications.
<allowGnfLocalUpdate>	Optional	0 - 1	String (1)	Y = Yes, allow application admin to update global notification file. N = No, don't allow application admin to update global notification file.
<allowLinkPackages>	Optional	0 - 1	String (1)	Y = Yes, allow external package linking. N = No, don't allow package linking.
<allowOnlyOneApproval>	Optional	0 - 1	String (1)	Y = Yes, allow install after only one approval. N = No, don't allow install if only one approval received.
<allowStageOverlay>	Optional	0 - 1	String (1)	Y = Yes, allow staged component to be overlaid by later check-in. N = No, don't allow staging overlays.
<allowTempChange>	Optional	0 - 1	String (1)	Y = Yes, allow temp change packages. N = No temp change packages allowed.
<apsConfigLib>	Optional	0 - 1	String (44), variable	Fully qualified dataset name of APS configuration library.
<apsHighLevelNode>	Optional	0 - 1	String (32), variable	Global default high-level node for APS datasets.
<apsRule>	Optional	0 - 1	String (1)	Code for APS processing rule. Values: 0 = No APS processing; standard ZMF. 1 = Explicit Mode. APS operates on physical dataset or component. 2 = Implicit Mode, Level 2. APS operates t level of application (AP), scenario (CN), data structure (DS), program (PG), online express (OX), report mockup (RP), & screen (SC). 3 = Implicit Mode, Level 1. APS list disabled for application (AP) or program (PG) operations; ZMF automatically checks out & stages related components. ***NOTE:*** Default value is 1.

Exhibit 10-28. PARMS GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<auditLevel>	Optional	0 - 1	String (1)	Code for package audit level required prior to freeze. Values: 0 = Audit optional before freeze. 1 = Audit required before freeze, but any return code for audit accepted except ABEND. 2 = Audit required before freeze. Return code for audit is 12 or less; out-of-synch conditions accepted in staging libraries. 3 = Audit required before freeze. Return code for audit is 8 or less. No out-of-synch conditions accepted in stage libraries, but are accepted with respect to baseline libraries. 4 = Audit required before freeze. Return code for audit is 4 or less. No out-of-synch conditions accepted relative to staging or baseline libraries; but staged module may duplicate its baseline counterpart. 5 = Audit required before freeze. Return code for audit is zero. No out-of-synch conditions or baseline module duplicates accepted.

<auditLockPackage>	Optional	0 - 1	String (1)	Lock package during audit. Values:
				A = Always.
				N = Never.
				O = Optional

<autoScratchLoadMbr>	Optional	0 - 1	String (1)	Y = Yes, auto scratch load with source. N = No, do not auto scratch load with source.
<buildInstallJclAtApprove>	Optional	0 - 1	String (1)	Y = Yes, build JCL install job at approval. N = No, don’t build JCL install job at approval.
<businessFromTime>	Optional	0 - 1	Time (4), hhmm	Global start time for hours of business operation, 24-hour format, no seconds.
<businessToTime>	Optional	0 - 1	Time (4), hhmm	Global end time for hours of business operation, 24-hour format, no seconds.

Exhibit 10-28. PARMS GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<chkOutRule>	Optional	0 - 1	String (1)	Code for checkout enforcement rule applied to staging of components that already exist in baseline. Values: 1 = Allow any user to stage components without preserving baseline checkout integrity. 2 = Allow authorized users to stage components without preserving baseline checkout integrity. 3 = Enforce baseline checkout integrity for all users.
<cmnEnvironment>	Optional	0 - 1	String (1)	Code for ZMF environment type. Values: 1 = All (development & production on same LPAR, no remote production) 2 = Development only 3 = Both development & production on same LPAR; remote production sites also supported) 4 = Production only
<cmnSchedulerInterval>	Optional	0 - 1	String (4), variable	ZMF internal scheduler polling interval in minutes.
<cmnVersion>	Optional	0 - 1	String (4), vrmm	Version, release, and modification level of this ZMF instance.
<compBuildStartedProcName>	Optional	0 - 1	String (8), variable	Started procedure name for building component compile, recompile and relink JCL.
<componentAgingPeriod>	Optional	0 - 1	String (6), variable	Component aging period for deletion in days. Value range: 1 to 32,000.
<componentMasterLib>	Optional	0 - 1	String (44), variable	Fully qualified dataset name of library where component master resides. NOTE: Component master must be PDS library, not HFS directory.
<createCmpWorkRecs>	Optional	0 - 1	String (1)	Y = Yes, create component worklist records for a package. N = No, omit component worklist records from package.
<db2SubSystemId>	Optional, DB2 only	0 - 1	String (4), fixed	Global default for DB2 subsystem name. ***NOTE:*** Default value is `'dsn '`.

Exhibit 10-28. PARMS GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<defaultScheduler>	Optional	0 - 1	String (1)	Type of scheduler used. 1 = ChangeMan ZMF. The installation jobs are submitted by the ChangeMan ZMF started task at the scheduled install date and time. 2 = Manual. The installation jobs are submitted as soon as the package approvals are complete. 3 = Other. The installation jobs are inserted into a third party scheduler via a batch job.
<defaultStartedProcName>	Optional	0 - 1	String (8), variable	Default started procedure name for building JCL.
<defaultUnitName>	Optional	0 - 1	String (8), variable	Global default for storage unit name.
<defaultVolume>	Optional	0 - 1	String (6), variable	Global default for storage volume ID.
<delayLib>	Optional	0 - 1	String (44), variable	Fully qualified dataset name of delay library for current ZMF instance.
<disableCalendar>	Optional	0 - 1	String (1)	Y = Yes, disable install calendar. N = No, don’t disable install calendar.
<disallowParallelChkOut>	Optional	0 - 1	String (1)	Y = Yes, prohibit checkout of active components. N = No, don’t prohibit checkout of active components.
<eliminatePersonalLib>	Optional	0 - 1	String (1)	Y = Yes, prohibit personal development library outside ZMF control. N = No, don’t prohibit personal development library outside ZMF.
<emailServerName>	Optional	0 - 1	String (32), variable	Email server name.
<emailServerPortid/>	Optional	0 - 1	String (5), variable	Email server port number.
<enableApprovalOrderProcess>	Optional	0 - 1	String (1)	Y = Yes, turn on hierarchical approvals. N = No, don’t turn on hierarchical approvals.
<enableCompUserVars>	Optional	0 - 1	String (1)	Y = Yes, enable component user variables. N = No, don’t enable component user variables.
<enableDisplayOrder3dSkel>	Optional	0 - 1	String (1)	Y = Yes, enable display order for 3D skeletons. N = No, don’t enable display order for 3D skeletons.

Exhibit 10-28. PARMS GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<enableDisplayOrderApplication>	Optional	0 - 1	String (1)	Y = Yes, enable display order for applications. N = No, don't enable display order for applications.
<enableDisplayOrderDb2Logical>	Optional	0 - 1	String (1)	Y = Yes, enable display order for DB2 logical. N = No, don't enable display order for DB2 logical.
<enableDisplayOrderDb2Physical>	Optional	0 - 1	String (1)	Y = Yes, enable display order for DB2 physical. N = No, don't enable display order for DB2 physical.
<enableDisplayOrderDbdOverride>	Optional	0 - 1	String (1)	Y = Yes, enable display order for DBD overrides. N = No, don't enable display order for DBD overrides.
<enableDisplayOrderImsControlReg>	Optional	0 - 1	String (1)	Y = Yes, enable display order for IMS control regions. N = No, don't enable display order for IMS control regions.
<enableDisplayOrderLanguage>	Optional	0 - 1	String (1)	Y = Yes, enable display order for language. N = No, don't enable display order for language.
<enableDisplayOrderLibtype>	Optional	0 - 1	String (1)	Y = Yes, enable display order for library types. N = No, don't enable display order for library types.
<enableDisplayOrderOnlineForm>	Optional	0 - 1	String (1)	Y = Yes, enable display order for online forms. N = No, don't enable display order for online forms.
<enableDisplayOrderProcedure>	Optional	0 - 1	String (1)	Y = Yes, enable display order for procedures . N = No, don't enable display order for procedures.
<enableDisplayOrderPsbOverride>	Optional	0 - 1	String (1)	Y = Yes, enable display order for PSB overrides. N = No, don't enable display order for PSB overrides.
<enableDisplayOrderReason>	Optional	0 - 1	String (1)	Y = Yes, enable display order for reason codes. N = No, don't enable display order for reason codes.

Exhibit 10-28. PARMS GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<enableDisplayOrderSite>	Optional	0 - 1	String (1)	Y = Yes, enable display order for sites. N = No, don't enable display order for sites.
<enableDisplayOrderXmlReport>	Optional	0 - 1	String (1)	Y = Yes, enable display order for XML reports. N = No, don't enable display order for XML reports.

<enableJes2Spool>	Optional	0 - 1	String (1)	Y = Yes, enable JES2 job entry system.
				N = No, don't use JES2 job entry.

<enableLLamBaseLib>	Optional	0 - 1	String (1)	Y = Yes, generate Librarian LAM
				N = No, don't enable CA Librarian LAM

<enableLibrBaseLib>	Optional	0 - 1	String (1)	Y = Yes, generate Librarian baseline lib
				N = Don't enable CA Librarian baseline

<enableOtherBaseLib>	Optional	0 - 1	String (1)	Y = Yes, generate other (Roscoe) baseline library N = No, don't enable other baseline library

<enablePanBaseLib>	Optional	0 - 1	String (1)	Y = Yes, generate Panvalet baseline lib
				N = No, don't enable CA Panvalet

<enableStackedRevDelta>	Optional	0 - 1	String (1)	Y = Yes, stacked reverse delta baseline N = Don't enable stacked reverse delta
<enableStageEditRecovery>	Optional	0 - 1	String (1)	Y = Yes, enable recovery of staging library edits. N = No, don't recover staging edits.

<enableVsamRlsOption>	Optional	0 - 1	String (1)	Y = Yes, enable VSAM record-level security.
				N = No, don't enableVSAM record-level security.

<eventNotifyCheckpoint>	Optional	0 - 1	String (18), FIXED	Event Notify Checkpoint number.
<forceChkOutToPackage>	Optional	0 - 1	String (1)	Y = Yes, force checkout to one package at a time. N = No, don't restrict checkout to one package at a time.
<forceDept>	Optional	0 - 1	String (1)	Y = Yes, require department number at package create. N = No, department number optional.
<forcePackageAudit>	Optional	0 - 1	String (1)	Y = Yes, require audit of all packages prior to approval. N = No, don't force package audit prior to approval.

Exhibit 10-28. PARMS GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<forceWorkChangeRequest>	Optional	0 - 1	String (1)	Y = Yes, required work order/change request at package create. N = No, work order/change request optional.
<globalNoticeModDate>	Optional	0 - 1	Date, yyyymmdd	Modification date for global notification dataset.
<globalNoticeModTime>	Optional	0 - 1	Time, hhmm	Modification time for global notification dataset, 24-hour format, no seconds.
<globalNoticeModTimeSeconds>	Optional	0 - 1	Integer (2)	Modification time seconds for global notification dataset.
<globalNotificationLib>	Optional	0 - 1	String (44), variable	Global notification dataset.
<hfsTempDirectory>	Optional	0-1	String (64), variable	Name of temporary HFS directory, prefixed by path from installation root, used for temporary changes to HFS components.
<inactiveTimeLimit>	Optional	0 - 1	String (6), variable	Time-out limit for inactive terminals in minutes. Value range: 1 to 999,999.

<includeIspllib>	Optional	0 - 1	String (1)	Y = Yes, include ISPLLIB libraries in validation & audit.
				N = No, omit ISPLLIB.

<includeIspmlib>	Optional	0 - 1	String (1)	Y = Yes, include ISPMLIB libraries in validation & audit.
				N = No, omit ISPMLIB.

<includeIspplib>	Optional	0 - 1	String (1)	Y = Yes, include ISPPLIB libraries in validation & audit.
				N = No, omit ISPPLIB.

<includeIspslib>	Optional	0 - 1	String (1)	Y = Yes, include ISPSLIB libraries in validation & audit.
				N = No, omit ISPSLIB.

<infoApproveRecord>	Optional	0 - 1	String (1)	Y = Yes, approve package automatically. N = No, do not approve package automatically.
<infoBypassAging>	Optional	0 - 1	String (1)	Y = Yes, INFO update bypassed during aging. N = No, INFO update not bypassed during aging.

Exhibit 10-28. PARMS GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<infoRule>	Optional	0 - 1	String (1)	Code for IBM INFO notification & approval rules. Values: 0 = No ZMF-to-INFO communication. 1 = INFO notified of package create. 2 = INFO notified of pkg create/update. 3 = Rule 2, plus: INFO subtask must be attached; enable INFO approval of packages. 4 = Rule 3, plus: planned packages must create INFO change record in advance & use its ID in ZMF work order/change request field. 5 = Rule 4, plus: unplanned packages also have INFO change record.
<infoSoapMember>	Optional	0 - 1	String (8)	INFO System Bus Soap member name.

<infoUsingBus>	Optional	0 - 1	String (1)	Y = Yes, INFO using system Bus.
				N = No, INFO not using system Bus.

<infoUsingVsam>	Optional	0 - 1	String (1)	Y = Yes, use VSAM file for INFO data mapping.
				N = No, don’t use VSAM file for INFO.

<infoVsamCreateTempPkg>	Optional	0 - 1	String (1)	Y = Yes, override ZMF temp package duration with setting in INFO VSAM file, or 1 day if INFO sets no value. N = No, don’t override ZMF temp pkg duration with INFO VSAM setting.
<infoVsamRejectPkg>	Optional	0 - 1	String (1)	Y = Yes, turn on selective INFO approval processing; ignore reject & other messages short of full approval. N = No, don’t turn on selective INF approval processing.
<infoVsamReuseRecord>	Optional	0 - 1	String (1)	Y = Yes, reuse VSAM file INFO record# from deleted packages for new packages. N = No, don’t reuse INFO record#. ***NOTE:*** If Y, deleted packages cannot be undeleted.
<installStartedProcName>	Optional	0 - 1	String (8), variable	Started procedure name for building package install JCL.
<isFullAssistanceCompleted>	Optional	0 - 1	String (1)	Y = Yes, impact analysis full assist tasks completed. N = No, full assist not completed.
<jobCard01>	Optional	0 - 1	String (72), variable	First of up to four global default JCL job cards.
<jobCard02>	Optional	0 - 1	String (72), variable	Second of up to four global default JCL job cards.

Exhibit 10-28. PARMS GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<jobCard03>	Optional	0 - 1	String (72), variable	Third of up to four global default JCL job cards.
<jobCard04>	Optional	0 - 1	String (72), variable	Fourth of up to four global default JCL job cards.
<keepBaselineBySite>	Optional	0 - 1	String (1)	Y = Yes, keep independent baseline libraries for each site. N = No, consolidate all sites under one baseline library.
<licensedLine1>	Optional	0 - 1	String (56), variable	Free-format text identification of ZMF licensee. First of up to two lines.
<licensedLine2>	Optional	0 - 1	String (56), variable	Free-format text identification of ZMF licensee. Second of up to two lines.
<logicalUnitName>	Optional	0 - 1	String (8), variable	Name of logical unit (LU) where this ZMF instance resides.
<maximumSiteCount>	Optional	0 - 1	Integer (3), variable	Deprecated; obsolete.
<modLevel>	Optional	0 - 1	String (2), fixed	ZMF modification level. Same as last two bytes of `<cmnVersion>`.
<nonVioUnitName>	Optional	0 - 1	String (8), variable	Default non-VIO UNIT name.

<notInfoChangeSystem>	Optional	0 - 1	String (1)	Y = Yes, INFO/MAN change management system.
				N = No, not an INFO/MAN change management system.

<numberOfDaysInCal>	Optional	0 - 1	String ()	Calendar number of days.
<onlineFormsIsppLib>	Optional, OFM only	0 - 1	String (44), variable	Fully qualified dataset name of ISPF panel library (ISPPLIB) for online forms.
<onlyMemoDelEmptyPackages>	Optional	0 - 1	String (1)	Y = Yes, enforce memo-delete rules for empty packages. N = No, don't enforce memo-delete for empty packages; allow physical deletion.
<packageAgingPeriod>	Optional	0 - 1	String (6), variable	Package aging period for deletion in days. Value range: 1 to 32,000.
<packageMasterLib>	Optional	0 - 1	String (44), variable	Fully qualified dataset name of library where package master resides.
<packageNumInPcver>	Optional	0 - 1	String (1)	Y = Yes, copy package number to PCVER variable. N = No, don't copy package number.

<pcverOutputOnly>	Optional	0 - 1	String (1)	Y = Yes, set PCVER to output only.
				N = No, PCVER not output only.

Exhibit 10-28. PARMS GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<processPartPackageByInsDate>	Optional	0 - 1	String (1)	Y = Yes, process participating packages by install date. N = No, don't process by install date.
<prohibitApprovalByCreator>	Optional	0 - 1	String (1)	Y = Yes, exclude package creator from package approval list. N = No, don't exclude package creator from package approval list.
<prohibitApprovalByWorker>	Optional	0 - 1	String (1)	Y = Yes, exclude developers who worked on package from package approval list. N = No, don't exclude package workers from package approval list.
<prohibitJobNameIncrement>	Optional	0 - 1	String (1)	Y = Yes, prohibit job name increment. N = No, don't prohibit job name increment.
<promotionRule>	Optional	0 - 1	String (1)	Code for package promotion rule. Values: 0 = Allow promote & demote even if package not frozen. 1 = Require freeze before promote. Require selective component demote, unfreeze, edit, refreeze, & re-promote directly to promotion level of package as a whole. 2 = Require freeze before promote. Require selective component demote, unfreeze, edit, refreeze, & re-promote through all intermediate promotion levels to level of package as a whole. 3 = Require freeze before promote. Require full package demote, then selective component unfreeze, edit, refreeze, & full package promotion through all intermediate promotion levels. 4 = Require freeze before promote. Require full package demote, followed by revert to development status, before editing components.
<promotionStartedProcName>	Optional	0 - 1	String (8), variable	Started procedure name for building package promote and demote JCL.
<release>	Optional	0 - 1	String (1)	ZMF release. Same as second byte of `<cmnVersion>`.

Exhibit 10-28. PARMS GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<releaseManAuditLevel/>	Optional	0 - 1	String (1)	Audit level. This numeric value dictates the Audit level that must be passed before a developing change package can be Frozen. 0 - Audit is recommended but entirely optional. 1 - Audit is required but any return code (except abend) is acceptable. 2 - Audit is required and the return code must not exceed 12, which implies that there are "out-of-synch" situations within the staging libraries. 3 - Audit is required and the return code must not exceed 8, which implies that there are no "out-of-synch" situations within the staging libraries, but are "out-of-synch" situations with respect to the Baseline libraries. 4 - Audit is required and the return code must not exceed 4, which implies that there are no "out-of-synch" situations within the staging or Baseline libraries, but at least one module of a staging library is a "duplicate" of its Baseline counterpart. 5 - Audit is required and the return code must not exceed 0, which implies that there are no "out-of-synch" situations with either implies that there are no "out-of-synch" situations with either the staging or Baseline libraries, and no "duplicates" exist.
<releaseManDsPattern>	Optional, ERO only	0 - 1	String (5), variable	Pattern for release area dataset name. May be any combination of the following dataset qualifier symbols, with L as the final qualifier and P optional: H = High-level qualifier as defined in `<rloHighLevelNode>`. R = Release name defined to ERO. A = Release area name defined for R in ERO. P = Application or project defined for R in ERO. L = Library type defined for R in ERO. ***NOTE:*** If used, `<rloHighLevelNode>` also returned.
<resourceClassLength>	Optional	0 - 1	String (4), variable	Length of `<resourceClassName>` in bytes. ***NOTE:*** Returned only if `<resourceClassName>` used.

Exhibit 10-28. PARMS GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<resourceClassName>	Optional	0 - 1	String (8), variable	Resource class name for RACF or other security package. ***NOTE:*** If used, its length appears in `<resourceClassLength>`.
<resourceClassOwner>	Optional	0 - 1	String (8), variable	Owner of security resource in `<resourceClassName>` as defined in RACF or other security package. ***NOTE:*** Returned optionally if `<resourceClassName>` used.
<rloHighLevelNode>	Optional, ERO only	0 - 1	String (8), variable	Global default high-level node for ERO release area datasets.
<rloHighLevelPath>	Optional, ERO only	0 - 1	String (1024), variable	Global default high-level HFS path for ERO release area files.
<showUserPanels>	Optional	0 - 1	String (1)	Y = Yes, display package user panels. N = No, don't show pkg user panels.
<siteName>	Optional	0 - 1	String (8), variable	ZMF name of site where this ZMF instance resides.
<stageDevHfsModel>	Optional	0 - 1	String (64), variable	HFS file name of model staging development library.
<stageDevLibModel>	Optional	0 - 1	String (32), variable	Dataset name of model staging development library.
<stageLibAgingPeriod>	Optional	0 - 1	String (6), variable	Staging library aging period for deletion in days. Value range: 1 to 32,000.
<stageLimitLevel>	Optional	0 - 1	String (1)	Code for staging restriction level. Values: 1 = Allow all users to stage or check-in to a package from a development library. 2 = Allow users that pass entity check to stage or check-in to a package from a development library. 3 = No users allowed to stage or check-in to a package from a development library.
<stageProdHfsModel>	Optional	0 - 1	String (64), variable	HFS file name of model staging production library.
<stageProdLibModel>	Optional	0 - 1	String (32), variable	Dataset name of model staging production library.

<suppressMsgsForINSJobs>	Optional	0 - 1	String (1)	Y = Yes, suppress INS job messages.
				N = No, display INS job messages.

<useBatchNotifier>	Optional	0 - 1	String (1)	Y = Yes, use batch notification vehicle. N = No, don't use batch notifier.

Exhibit 10-28. PARMS GBL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies

<useCmnScheduler>	Optional	0 - 1	String (1)	Y = Yes, use ZMF install scheduler.
				N = No, don't use ZMF scheduler.

<useEmailNotifier>	Optional	0 - 1	String (1)	Y = Yes, use email notification vehicle. N = No, don't use email notifier.
<useGlobalNotifier>	Optional	0 - 1	String (1)	Y = Yes, use global notification file. N = No, don't use global notification file.

<useIebcopyDelivery>	Optional	0 - 1	String (1)	Y = Yes, use IEBCOPY file transfers.
				N = No, don't use IEBCOPY file xfer.

<useLikeLODinSyslib>	Optional	0 - 1	String (1)	Y = Yes, use LIKE-LOD in syslib.
				N = No, don't use LIKE-LOD in syslib

<useManualScheduler>	Optional	0 - 1	String (1)	Y = Yes, use manual install scheduler. N = No, don't use manual scheduler.

<useMvsSendNotifier>	Optional	0 - 1	String (1)	Y = Yes, use MVS Send notification.
				N = No, don't use MVS Send notifier.

<useOpcEcaScheduler>	Optional	0 - 1	String (1)	Y = Yes, use OPS/ECA scheduler.
				N = No, don't use OPS/ECA scheduler.

<useOtherScheduler>	Optional	0 - 1	String (1)	Y = Yes, use other install scheduler. N = No, don't use other scheduler.
<useSerCompress>	Optional	0 - 1	String (1)	Y = Yes, use SERNET compression on file transfers. N = No, don't use SERNET data compression.

<useSernetNotifier>	Optional	0 - 1	String (1)	Y = Yes, use SERNET notification vehicle.
				N = No, don't use SERNET notification vehicle.

<useZprefixForBatchJobs>	Optional	0 - 1	String (1)	Y = Yes, use ZPREFIX instead of ZUSER for batch job user ID. N = No, use ZUSER default for batch job user ID.
<validateStageVersion>	Optional	0 - 1	String (1)	Y = Yes, validate component staging version at check-in. N = No, don't validate staging version.
<version>	Optional	0 - 1	String (1)	ZMF version. Same as first byte of `<cmnVersion>`.

Parameters Application List - PARMS APL LIST

The application parameter list reports application-level overrides to global settings in ChangeMan ZMF that enforce the business rules for software change management at your installation. If you have not customized your application parameters, the default values are reported.

The Serena XML service/scope/message tags and attributes for messages to *list* application parameters are:

```
<service name="PARMS">
<scope name="APL">
<message name="LIST">
```

These tags appear in both requests and replies.

PARMS APL LIST — Request

The application parameter list service requires an application name in the request message. You can request two types of application parameter list:

- ***Default Parameters Application List*** — Name the desired application in the `<applName>` tag and omit the `<parms>` tag in the request message or leave it blank. This option requests all application parameters for the named application. User must have access permissions for the application to retrieve a result.
- ***Short Application Parameter List*** — Name the desired application in the `<applName>` tag and enter "S" (upper case) in the `<parms>` tag of the request message. This option requests only the application description. A result is returned if the application exists, even if the user does not have application access permissions.

Data structure details for the <request> data element appear in *Exhibit 10-29*.

Exhibit 10-29. PARMS APL LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	1	String (4), variable	ZMF application name desired. ***NOTE:*** OK to omit trailing blanks.
<parms>	Optional	0 - 1	String (1)	Code for type of list requested. Values: S = Short list (description only), all users. Blank = Complete list, authorized application users only. ***NOTE:*** If used, upper case required.

PARMS APL LIST — Reply

The reply message for this function exactly one `<result>` data element containing the requested parameter settings for the named application. If no parameter settings have been customized for this application, default (global) values are returned.

The standard `<response>` data element follows any `<result>` tags in the reply and indicates the success or failure of the list request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. Because it is the final data element in the XML reply message, the `<response>` tag serves as an end-of-list marker.

Data structure details for the `<result>` tag appear in *Exhibit 10-30*.

Exhibit 10-30. PARMS APL LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<allowChkOutToDevLib>	Optional	0 - 1	String (1)	Y = Yes, allow checkout to personal development library outside ZMF. N = Don't allow checkout to personal development library outside ZMF
<allowLinkPackages>	Optional	0 - 1	String (1)	Y = Yes, allow external package linking. N = No, don't allow package linking.
<allowOnlyOneApproval>	Optional	0 - 1	String (1)	Y = Yes, allow install after only one approval. N = No, don't allow install if only one approval received.
<allowStageOverlay>	Optional	0 - 1	String (1)	Y = Yes, allow staged component to be overlaid by later check-in. N = No, don't allow staging overlays.
<allowTempChange>	Optional	0 - 1	String (1)	Y = Yes, allow temp change packages. N = No temp change packages allowed.
<applDesc>	Optional	0 - 1	String (44), variable	Application description.
<applName>	Optional	0 - 1	String (4), variable	ZMF application name.
<apsRule>	Optional	0 - 1	String (1)	Code for APS processing rule. Values: 0 = No APS processing; standard ZMF. 1 = Explicit Mode. APS operates on physical dataset or component. 2 = Implicit Mode, Level 2. APS operates t level of application (AP), scenario (CN), data structure (DS), program (PG), online express (OX), report mockup (RP), & screen (SC). 3 = Implicit Mode, Level 1. APS list disabled for application (AP) or program (PG) operations; ZMF automatically checks out & stages related components. ***NOTE:*** Default value is 1.

Exhibit 10-30. PARMS APL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<auditLevel>	Optional	0 - 1	String (1)	Code for package audit level required prior to freeze. Values: 0 = Audit optional before freeze. 1 = Audit required before freeze, but any return code for audit accepted except ABEND. 2 = Audit required before freeze. Return code for audit is 12 or less; out-of-synch conditions accepted in staging libraries. 3 = Audit required before freeze. Return code for audit is 8 or less. No out-of-synch conditions accepted in stage libraries, but are accepted with respect to baseline libraries. 4 = Audit required before freeze. Return code for audit is 4 or less. No out-of-synch conditions accepted relative to staging or baseline libraries; but staged module may duplicate its baseline counterpart. 5 = Audit required before freeze. Return code for audit is zero. No out-of-synch conditions or baseline module duplicates accepted.

<auditLockPackage>	Optional	0 - 1	String (1)	Lock package during audit. Values:
				A = Always.
				N = Never.
				O = Optional

<buildInstallJclAtApprove>	Optional	0 - 1	String (1)	Y = Yes, build JCL install job at approval. N = No, don't build JCL install job at approval.
<businessFromTime>	Optional	0 - 1	Time (4), hhmm	Start time for business operations & package install for this application, 24-hour format, no seconds.
<businessToTime>	Optional	0 - 1	Time (4), hhmm	End time for business operations & package install for this application, 24-hour format, no seconds.
<chkOutEntity>	Optional	0 - 1	String (8), variable	Security entity in RACF or other security package to which user ID must belong to pass security entity check on staging. ***NOTE:*** Returned if value is 2 in `<chkOutRule>`.

Exhibit 10-30. PARMS APL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<chkOutRule>	Optional	0 - 1	String (1)	Code for checkout enforcement rule applied to staging of components that already exist in baseline. Values: 1 = Allow any user to stage components without preserving baseline checkout integrity. 2 = Allow authorized users to stage components without preserving baseline checkout integrity. 3 = Enforce baseline checkout integrity for all users.
<cmnaudrcEntity>	Optional	0 - 1	String (8), variable	Entity for authority to run CMNAUDRC outside of ChangeMan
<componentAgingPeriod>	Optional	0 - 1	String (6), variable	Component aging period for deletion in days. Value range: 1 to 32,000.
<createCmpWorkRecs>	Optional	0 - 1	String (1)	Y = Yes, create component worklist records for a package. N = No, omit component worklist records from package.
<defaultScheduler>	Optional	0 - 1	String (8), variable	Type of scheduler used. 1 - ChangeMan ZMF. The installation jobs are submitted by the ChangeMan ZMF started task at the scheduled install date and time. 2 - Manual. The installation jobs are submitted as soon as the package approvals. are complete. 3 - Other. The installation jobs are inserted into a third party scheduler via a batch job.
<defaultUnitName>	Optional	0 - 1	String (8), variable	Application default for storage unit name.
<defaultVolume>	Optional	0 - 1	String (6), variable	Application default for storage volume ID.
<disallowParallelChkOut>	Optional	0 - 1	String (1)	Y = Yes, prohibit checkout of active components. N = No, don't prohibit checkout of active components.
<eliminatePersonalLib>	Optional	0 - 1	String (1)	Y = Yes, prohibit personal development library outside ZMF control. N = No, don't prohibit personal development library outside ZMF.
<enableApprovalOrderProcess>	Optional	0 - 1	String (1)	Y = Yes, turn on hierarchical approvals. N = No, don't turn on hierarchical approvals.
<enableDisplayOrderDb2Logical>	Optional	0 - 1	String (1)	Y = Yes, enable display order for DB2 logical N = No, don't enable.

Exhibit 10-30. PARMS APL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<enableDisplayOrderDbdOverride>	Optional	0 - 1	String (1)	Y = Yes, enable display order for DBD overrides. N = No, don't enable.
<enableDisplayOrderImsControlReg>	Optional	0 - 1	String (1)	Y = Yes, enable display order for IMS control regions. N = No, don't enable.
<enableDisplayOrderLanguage>	Optional	0 - 1	String (1)	Y = Yes, enable display order for languages. N = No, don't enable.
<enableDisplayOrderLibtype>	Optional	0 - 1	String (1)	Y = Yes, enable display order for library type. N = No, don't enable.
<enableDisplayOrderProcedure>	Optional	0 - 1	String (1)	Y = Yes, enable display order for procedures. N = No, don't enable.
<enableDisplayOrderPsbOverride>	Optional	0 - 1	String (1)	Y = Yes, enable display order for PSB overrides. N = No, don't enable.
<enableDisplayOrderSite>	Optional	0 - 1	String (1)	Y = Yes, enable display order for sites. N = No, don't enable.
<enableDisplayXmlReport>	Optional	0 - 1	String (1)	Y = Yes, enable display order for XML reports. N = No, don't enable.
<enableStageEditRecovery>	Optional	0 - 1	String (1)	Y = Yes, enable recovery of staging library edits. N = No, don't recover staging edits.
<forceChkOutToPackage>	Optional	0 - 1	String (1)	Y = Yes, force checkout to one package at a time. N = No, don't restrict checkout to one package at a time.
<forceDept>	Optional	0 - 1	String (1)	Y = Yes, require department number at package create. N = No, department number optional.
<forcePackageAudit>	Optional	0 - 1	String (1)	Y = Yes, require audit of all packages prior to approval. N = No, don't force package audit prior to approval.
<forceWorkChangeRequest>	Optional	0 - 1	String (1)	Y = Yes, required work order/change request at package create. N = No, work order/change request optional.
<inUseOwner>	Optional	0 - 1	String (6), variable	TSO user ID of current application (project) user.

Exhibit 10-30. PARMS APL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<infoRule>	Optional	0 - 1	String (1)	Code for IBM INFO notification & approval rules. Values: 0 = No ZMF-to-INFO communication. 1 = INFO notified of package create. 2 = INFO notified of pkg create/update. 3 = Rule 2, plus: INFO subtask must be attached; enable INFO approval of packages. 4 = Rule 3, plus: planned packages must create INFO change record in advance & use its ID in ZMF work order/change request field. 5 = Rule 4, plus: unplanned packages also have INFO change record.

<infoUsingVsam>	Optional	0 - 1	String (1)	Y = Yes, use VSAM file for INFO data mapping.
				N = No, don't use VSAM file for INFO.

<infoVsamCreateTempPkg>	Optional	0 - 1	String (1)	Y = Yes, override ZMF temp package duration with setting in INFO VSAM file, or 1 day if INFO sets no value. N = No, don't override ZMF temp pkg duration with INFO VSAM setting.
<infoVsamRejectPkg>	Optional	0 - 1	String (1)	Y = Yes, turn on selective INFO approval processing; ignore reject & other messages short of full approval. N = No, don't turn on selective INF approval processing.
<infoVsamReuseRecord>	Optional	0 - 1	String (1)	Y = Yes, reuse VSAM file INFO record# from deleted packages for new packages. N = No, don't reuse INFO record#. ***NOTE:*** If Y, deleted packages cannot be undeleted.

<isApplBusy>	Optional	0 - 1	String (1)	Y = Yes, application (project) in use.
				N = No, application (project) not in use.

<isApplLocked>	Optional	0 - 1	String (1)	Y = Yes, application (project) locked.
				N = No, application (project) not locked.

<jobCard01>	Optional	0 - 1	String (72), variable	First of up to four global default JCL job cards.
<jobCard02>	Optional	0 - 1	String (72), variable	Second of up to four global default JCL job cards.
<jobCard03>	Optional	0 - 1	String (72), variable	Third of up to four global default JCL job cards.
<jobCard04>	Optional	0 - 1	String (72), variable	Fourth of up to four global default JCL job cards.

Exhibit 10-30. PARMS APL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<keepBaselineBySite>	Optional	0 - 1	String (1)	Y = Yes, keep independent baseline libraries for each site. N = No, consolidate all sites under one baseline library.
<nextPackageNumber>	Optional	0 - 1	String (6), variable	Unique serial number to assign to next package created for this application.
<onlyMemoDelEmptyPackages>	Optional	0 - 1	String (1)	Y = Yes, enforce memo-delete rules for empty packages. N = No, don't enforce memo-delete for empty packages; allow physical deletion.
<packageAgingPeriod>	Optional	0 - 1	String (6), variable	Package aging period for deletion in days. Value range: 1 to 32,000.
<packageBindLibType>	Optional	0 - 1	String (3), variable	Library type for package bind.
<packageNumInPcver>	Optional	0 - 1	String (1)	Y = Yes, copy package number to PCVER variable. N = No, don't copy package number.

<pcverOutputOnly>	Optional	0 - 1	String (1)	Y = Yes, set PCVER to output only.
				N = No, PCVER not output only.

<processPartPackageByInsDate>	Optional	0 - 1	String (1)	Y = Yes, process participating packages by install date. N = No, don't process by install date.
<prohibitApprovalByCreator>	Optional	0 - 1	String (1)	Y = Yes, exclude package creator from package approval list. N = No, don't exclude package creator from package approval list.
<prohibitApprovalByWorker>	Optional	0 - 1	String (1)	Y = Yes, exclude developers who worked on package from package approval list. N = No, don't exclude package workers from package approval list.
<prohibitJobNameIncrement>	Optional	0 - 1	String (1)	Y = Yes, prohibit job nameincrement. N = No, don't prohibit job name increment.

Exhibit 10-30. PARMS APL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<promotionRule>	Optional	0 - 1	String (1)	Code for package promotion rule. Values: 0 = Allow promote & demote even if package not frozen. 1 = Require freeze before promote. Require selective component demote, unfreeze, edit, refreeze, & re-promote directly to promotion level of package as a whole. 2 = Require freeze before promote. Require selective component demote, unfreeze, edit, refreeze, & re-promote through all intermediate promotion levels to level of package as a whole. 3 = Require freeze before promote. Require full package demote, then selective component unfreeze, edit, refreeze, & full package promotion through all intermediate promotion levels. 4 = Require freeze before promote. Require full package demote, followed by revert to development status, before editing components.
<skelsReleaseId>	Optional	0 - 1	String (4), variable	3-D skeletons release ID.
<stageEntity>	Optional	0 - 1	String (8), variable	Security entity in RACF or other security package to which user ID must belong to pass security entity check on checkout. ***NOTE:*** Returned if value is 2 in `<stageLimitLevel>`.
<stageLibAgingPeriod>	Optional	0 - 1	String (6), variable	Staging library aging period for deletion in days. Value range: 1 to 32,000.
<stageLimitLevel>	Optional	0 - 1	String (1)	Code for staging restriction level. Values: 1 = Allow all users to stage or check-in to a package from a development library. 2 = Allow users that pass entity check to stage or check-in to a package from a development library. 3 = No users allowed to stage or check-in to a package from a development library.
<suppressMsgsForINSJobs>	Optional	0 - 1	String (1)	Y = Yes, suppress messages for install jobs. N = No, do not suppress messages for install jobs.

Exhibit 10-30. PARMS APL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<useApplInCurrentHist>	Optional	0 - 1	String (1)	Y = Use APPL in current history process N = Do not use APPL in current history process

<useCmnScheduler>	Optional	0 - 1	String (1)	Y = Use CMN scheduler
				N = Do not use CMN scheduler

<useLikeLODinSyslib>	Optional	0 - 1	String (1)	Y = Use Like LOD in SYSLIB
				N = Do not use Like LOD in SYSLIB

<useManualScheduler>	Optional	0 - 1	String (1)	Y = Use Manual scheduler
				N = Do not use Manual scheduler

<useOtherScheduler>	Optional	0 - 1	String (1)	Y = Use Other scheduler
				N = Do not use Other scheduler

<useZPrefixForBatchJobs>	Optional	0 - 1	String (1)	Y = Yes, use ZPREFIX not ZUSE for batch.
				N = No, use ZUSE not ZPREFIX for batch.

<validateStageVersion>	Optional	0 - 1	String (1)	Y = Yes, validate version during staging. N = No, do not validate version during staging.

List Global Reason Codes - REASONS SERVICE LIST

This function lists all globally defined reason codes and their definitions for the creation of temporary change packages. If temporary change packages are not permitted at your installation, no results are returned by this function.

The Serena XML service/scope/message names for messages to *list* global reason codes for temporary packages are:

```
<service name="REASONS">
<scope name="SERVICE">
<message name="LIST">
```

These tags appear in both requests and replies.

REASONS SERVICE LIST — Requests

Global reason code list requests take no parameters and contain an empty `<request>` data element. The `<request>` tag is required to identify the message as a request rather than a reply.

Note

XML syntax allows both a long form and a short form for empty tags. An empty `<request>` tag can therefore be coded in one of two ways.

Long form:

```
<request>
</request>
```

Equivalent short form:

```
<request/>
```

REASONS SERVICE LIST — Replies

Zero to many `<result>` data elements are returned in response to a Serena XML request to list global reason codes. Each `<result>` tag defines a single reason code.

A standard `<response>` data structure follows the last `<result>` tag to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last tag in the reply message, the `<response>` tag also serves as an end-of-list marker.

Data structure details for the `<result>` tag appear in *Exhibit 10-31*.

Exhibit 10-31. REASONS SERVICE LIST<result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<displayOrderNo>	Optional	0 - 1	Integer (undefined)	Display order number. This numeric value dictates the default order in which a list of items is displayed.
<reasonCode>	Required	1	String (3), variable	Code identifying a reason for creating a temporary package.
<reasonDesc>	Required	1	String (255), variable	Text description corresponding to code in `<reasonCode>` tag.

Approver and Notification Administration

Serena XML supports the following approver and notification administration tasks for general use:

- *List Application Approvers - APPROVER APL LIST*
- *Download Global Notification File - NOTYFILE SERVICE DOWNLOAD*
- *Notify User - USER SERVICE NOTIFY*

The syntax that identifies these functions appears in the `name` attribute of the `<service>` tag, as follows:

```
<service name="APPROVER">
<service name="NOTYFILE">
<message name="NOTIFY">
```

List Application Approvers - APPROVER APL LIST

This function lists authorized approvers for a named application. Both regular and emergency approvers are included in the scope of this function. Approvers assigned to review an application by a customized ChangeMan ZMF exit are omitted. If no approvers are defined for the application, no results are returned.

The Serena XML service/scope/message tags for a message to *list* application approvers are:

```
<service name="APPROVER">
<scope name="APL">
<message name="LIST">
```

These tags appear in both request and reply messages.

APPROVER APL LIST — Requests

All application approver list requests require the application name in the `<applName>` tag and the approver list type in `<approverListType>`. Serena XML supports four types of application approver lists:

- ***Planned Approvers*** — Enter "0" (zero) in the `<approverListType>` tag to request all regular approvers for planned packages in the named application.
- ***Unplanned Approvers*** — Enter "1" in the `<approverListType>` tag to request all emergency approvers for unplanned packages in the named application.
- ***All Approvers*** — Enter "2" in the `<approverListType>` tag to request both approver types for the named application.
- ***Description of Named Approver*** — Name the approver entity of interest in the `<approverEntity>` tag. Enter "2" in the `<approverListType>` tag to request both approver types. The function returns the named approver's description and notification information, as defined for the named application.

Data structure details for the `<request>` tag appear in *Exhibit 10-32*.

Exhibit 10-32. APPROVER APL LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Required	1	String (4), variable	ZMF application name. ***NOTE:*** OK to omit trailing blanks.
<approverEntity>	Optional	0 - 1	String (8), variable	Security system entity ID for desired package approver or group.

Exhibit 10-32. APPROVER APL LIST <request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<approverListType>	Required	1	String (1)	Code for type of approver to include in results. Values: 0 = Planned package approvers only. 1 = Unplanned package approvers only. 2 = Both planned & unplanned package approvers.

APPROVER APL LIST — Replies

The Serena XML reply to a package approver list request returns zero to many `<result>` tags. Each `<result>` tag contains a description and notification information for one application approver entity defined to the security package for your system. Individual approver TSO user IDs are not included in the results.

A standard `<response>` data element follows the last `<result>` tag, if any, to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher. As the last tag returned in the reply message, the `<response>` tag serves as an end-of-list marker.

Data structure details for the `<result>` tag follow in *Exhibit 10-33*.

Exhibit 10-33. APPROVER APL LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<applName>	Optional	1	String (4), fixed	ZMF application name. Same as first 4 bytes of package name.
<approvalOrder>	Optional	0 - 1	Integer (2), variable	Approval level or sequence assigned to this approver entity for hierarchical approvals.
<approverDesc>	Optional	0 - 1	String (44), variable	Text description of approver level or function (e.g., project leader, QA manager) for `<approverEntity>`.
<approverEntity>	Optional	1	String (8), variable	Security system entity ID of authorized application approver.
<approverListType>	Optional	1	String (1)	Code for type of approver to include in results. Values: 0 = Planned package approvers only. 1 = Unplanned package approvers only. 2 = Both planned & unplanned package approvers.
<isInterfacingApprover>	Optional	0 - 1	String (1), variable	Is this an interfacing approver? **Y** = Yes **N** = No

Exhibit 10-33. APPROVER APL LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<notification>	Optional	0 - 35	Complex	Describes notifications sent when this approver entity takes an approval action. See *Exhibit 10-34*.

The `<result>` data structure contains zero to many instances of the complex subtag, `<notification>`, which contains tags of its own. Each `<notification>` structure identifies a notification agent type and a list of user IDs or email addresses to be used by that agent when sending notifications to the approver entity.

Data structure details follow in *Exhibit 10-34*.

Exhibit 10-34. <notification> Subtag Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<notifierType>	Optional	0 - 1	Integer (1)	ZMF code for notification method to use with notifications sent to users in `<userList>`. Values: **1** = MVSSEND message **4** = Email **5** = SERNET agent email **6** = Batch messaging job
<userList>	Optional	0 - 1	String (44), variable	List of individual approvers to notify when the named approver entity takes an approval action. List consists of user TSO IDs or E-mail addresses separated by commas. ***NOTE:*** TSO user ID required if `<notifierType>` = 1. ***NOTE:*** E-mail addresses required if `<notifierType>` = 5.

Download Global Notification File - NOTYFILE SERVICE DOWNLOAD

Serena XML can request a download of the global notification file to your local client system. The request message takes no parameter values. The reply message contains global notification file records.

The Serena XML service/scope/message tags for a message to *download* the global notification file are:

```
<service name=”NOTYFILE”>
<scope name=”SERVICE”>
<message name=”DOWNLOAD”>
```

These tags appear in both request and reply messages.

NOTYFILE SERVICE DOWNLOAD — Requests

The request message for this function contains an empty `<request>` data element. The `<request>` tag is required to identify the message as a request rather than a reply.

Note

XML syntax allows both a long form and a short form for empty tags. An empty `<request>` tag can therefore be coded in one of two ways.

Long form:

```
<request>
</request>
```

Equivalent short form:

```
<request/>
```

NOTYFILE SERVICE DOWNLOAD — Replies

The reply message for this function returns one `<result>` data element containing zero to many `<line>` tags. Following the `<result>` tag, the function returns a standard `<response>` data element to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Data structure details for the `<result>` data element appear in *Exhibit 10-35*.

Exhibit 10-35. NOTYFILE SERVICE DOWNLOAD <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<line>	Optional	0 - ∞	String, variable	A bytewise image of one record in the ZMF global notification file.

Upload Global Notification File - NOTYFILE SERVICE UPLOAD

Serena XML can request an upload of the global notification file. The request message takes no parameter values. The reply message does not return any result data.

The Serena XML service/scope/message tags for a message to *upload* the global notification file are:

```
<service name="NOTYFILE">
<scope name="SERVICE">
<message name="UPLOAD">
```

These tags appear in both request and reply messages.

NOTYFILE SERVICE UPLOAD — Requests

The request message for this function contains an empty `<request>` data element. The `<request>` tag is required to identify the message as a request rather than a reply.

Note

XML syntax allows both a long form and a short form for empty tags. An empty `<request>` tag can therefore be coded in one of two ways.

Long form:

```
<request>
</request>
```

Equivalent short form:

```
<request/>
```

NOTYFILE SERVICE UPLOAD — Replies

The reply message for this function does not return any `<result>` data. It does, however, return a standard `<response>` data element to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Notify User - USER SERVICE NOTIFY

This function sends either a predefined or free-format notification message to one or more users of ChangeMan ZMF. Values for predefined message variables may be substituted either from the request message or from system data maintained by ChangeMan ZMF. Notifications may be sent by any one of the following means:

- ***MVS/TSO Send*** — MVSSEND message to users on the same mainframe LPAR as the XML client that issues the request.
- ***Email*** — E-mail using SMTP
- ***SERNET Email*** — E-mail using SERNET.
- ***Batch*** — Creates a batch messaging job using ISPF file tailoring, where the appropriate request values and package information are supplied to the file tailoring skeleton in ISPF variables. (Refer to example file tailoring skeleton CMN$$NTF in ChangeMan ZMF.)

The Serena XML service/scope/message tags for a message to *notify* users are:

```
<service name="USER">
<scope name="SERVICE">
<message name="NOTIFY">
```

These tags appear in both request and reply messages.

USER SERVICE NOTIFY — Requests

The user notification request message requires a sender, a recipient, a notification method, and either a `<messageType>` entry or the actual message content in `<textMessage>`. Serena XML supports two types of user notifications:

- ***Generic Text Message*** — To send a generic text message, enter a blank character (not a null or empty tag) in `<messageType>` and supply the free-format text message contents in `<textMessage>`.
- ***Predefined Message*** — To send a predefined text message, enter the desired message type code in `<messageType>`. Required variables for the message type may be populated in the Serena XML request, or the `<loadPackageInfo>` tag can request that ChangeMan ZMF populate their values from the package master.

Data structure details for the `<request>` tag appear in *Exhibit 10-36*.

Exhibit 10-36. USER SERVICE NOTIFY<request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<actionDate>	Optional	0 - 1	Date, yyyymmdd	Date of action taken. Substitute for (Date) variable in predefined message.
<actionTime>	Optional	0 - 1	Time, hhmmss	Time of action taken. Substitute for (Time) variable in predefined message.
<actionType>	Optional	0 - 1	String (1)	Code for action required of recipient. Values: 0 = Information only, no action required. 1 = Action required. ***NOTE:*** Tag is required with value = 1 if value in `<messageType>` is 5.
<applName>	Optional	0 - 1	String (4), variable	ZMF application name. Same as first four bytes of <package>. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`. ***NOTE:*** OK to omit trailing blanks.
<cmnSubSystemId>	Optional	0 - 1	String (1)	ZMF subsystem ID of started task to process batch messaging job. ***NOTE:*** If `<notifierType>` value = 6, this tag is required. ***NOTE:*** Default value is blank.
<creator>	Optional	0 - 1	String (8), variable	TSO user ID of package creator. Substituted for (Creator) variable in predefined message.
<formNumber>	Optional	0 - 1	String (3), variable	Online form number to be substituted for (Form) variable in predefined message.
<ipAddress>	Optional	0 - 1	String (8), variable	TCP/IP address of SERNET email server to override default.
<isApprovalOrderEnabled>	Optional	0 - 1	String (1)	Substituted for variable in predefined message. Values: Y = Yes, use hierarchical approval order. N = No, don't use hierarchical approval.

Exhibit 10-36. USER SERVICE NOTIFY<request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<jobCard01>	Optional	0 - 1	String (72), variable	First of up to four JCL job cards used with batch messaging job. ***NOTE:*** If `<notifierType>` value = 6, this tag is required.
<jobCard02>	Optional	0 - 1	String (72), variable	Second of up to four JCL job cards used with batch messaging job.
<jobCard03>	Optional	0 - 1	String (72), variable	Third of up to four JCL job cards used with batch messaging job.
<jobCard04>	Optional	0 - 1	String (72), variable	Fourth of up to four JCL job cards used with batch messaging job.
<loadPackageInfo>	Optional	0 - 1	String (1)	Y = Yes, retrieve info from package master rather than XML request. N = No, don't retrieve info from package master; use values in XML request. ***NOTE:*** Tag is required with value = Y if value in `<messageType>` is R. ***NOTE:*** If Y, `<package>` also required.

Exhibit 10-36. USER SERVICE NOTIFY<request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<messageType>	Required	1	String (1)	Code for predefined message format. Values: **Blank = Generic message.** **0** = 'CMN4600I - ChangeMan package (Package) awaits your approval.' 1 = 'CMN400I - Package (Package) approved by (User) on (Date) at (Time).' 2 = 'CMN400I - Package (Package) backed out by (User) on (Date) at (Time).' 3 = 'CMN400I - Package (Package) baselined by (User) on (Date) at (Time).' 5 = 'CMN400I - Package (Package) deleted by (User) on (Date) at (Time).' 7 = 'CMN400I - Package (Package) distributed by (User) on (Date) at (Time).' 8 = 'CMN400I - Package (Package) frozen by (User) on (Date) at (Time).' 9 = 'CMN400I - Package (Package) installed by (User) on (Date) at (Time).' B = 'CMN400I - Package (Package) rejected by (User) on (Date) at (Time).' C = 'CMN400I - Package (Package) cycled by (User) on (Date) at (Time).' R = 'CMN4600I - ChangeMan release (Release) awaits your approval.' ***NOTE:*** If blank, tags `<textMessage>` & `<textLength>` are also required. ***NOTE:*** If value is 0, the `<package>` tag is also required. ***NOTE:*** If value is 1, 2, 3, 7, 8, 9, B, or C, tags `<package>`, `<actionDate>`, & `<actionTime>` are also required. NOTE: If value is 5, tags `<package>`, `<actionDate>`, `<actionTime>`, & `<actionType>` are also required. Value in `<actionType>` must be 1. ***NOTE:*** If value is R, a release ID must be retrieved via `<loadPackageInfo>` tag value = Y.

Exhibit 10-36. USER SERVICE NOTIFY<request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<notifierType>	Required	1	String (1)	Code for notification method to use with recipients. Values: **1** = MVSSEND message **4** = Email using SMTP **5** = SERNET agent email **6** = Batch messaging job ***NOTE:*** If value = 6, tags `<skelsName>`, `<jobCard01>`, and `<cmnSubSytemID>` are also required.
<package>	Optional	0 - 1	String (10), variable	ZMF name of package. Value substituted for (Package) variable in predefined messages. ***NOTE:*** Required if `<loadPackageInfo>` value is Y.
<packageId>	Optional	0 - 1	String (6), variable	ZMF package number. Same as last six bytes of <package>. ***NOTE:*** Not recommended as replacement for `<package>` tag. Use `<package>` instead of `<applName>` & `<packageId>`. ***NOTE:*** OK to omit leading zeroes.
<packageLevel>	Optional	0 - 1	String (1)	Code for package complexity or level in hierarchy. Substituted for variable in predefined message. Values: **1** = Simple package **2** = Complex package **3** = Super package **4** = Participating package
<packageStatus>	Optional	0 - 1	String (1)	Code for status of package in lifecycle. Substituted for variable in predefined message. Values: **1** = Approved **2** = Backed out **3** = Baselined **4** = Complex/super pkg closed **5** = Deleted (memo delete) **6** = Development **7** = Distributed **8** = Frozen **9** = Installed **A** = Complex/super pkg open **B** = Rejected **C** = Temporary change cycle completed
<packageTitle>	Optional	0 - 1	String (72), variable	Working title of package. Substituted for variable in predefined message.

Exhibit 10-36. USER SERVICE NOTIFY<request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<packageType>	Optional	0 - 1	String (1)	Package install type code. Substituted for variable in predefined message. Values: **1** = Planned permanent **2** = Planned temporary **3** = Unplanned permanent **4** = Unplanned temporary
<portId>	Optional	0 - 1	String (16), variable	Port ID of SERNET email server to override default.
<recipient>	Required	1	String (44), variable	One or more recipient user IDs or e-mail addresses separated by delimiters, as appropriate for the notification method. ***NOTE:*** If `<notifierType>` value = 1, this tag contains comma-delimited list of TSO user IDs. ***NOTE:*** If `<notifierType>` value = 4 or 5, this tag contains semicolon-delimited list of email addresses. ***NOTE:*** If `<notifierType>` value = 6, tag format depends on batch notification file tailoring skeleton in `<skelsName>`.
<requestorDept>	Optional	0 - 1	String (4), variable	Department code of requestor. Substituted for variable in predefined message.
<requestorName>	Optional	0 - 1	String (25), variable	Name of requestor. Substituted for variable in predefined message.
<requestorPhone>	Optional	0 - 1	String (15), variable	Phone number of requestor. Substituted for variable in predefined message.
<sender>	Required	1	String (8), variable	TSO user ID of user taking reported action. Substituted for (User) variable in predefined messages.
<skelsName>	Optional	0 - 1	String (8), variable	Name of batch notification file tailoring skeleton. Default value is CMN$$NTF. ***NOTE:*** If `<notifierType>` value = 6, this tag is required.
<subject>	Optional	0 - 1	String (50), variable	Entry for email or batch message "Subject" field. ***NOTE:*** Used only if `<messageType>` value is blank. ***NOTE:*** Used only if `<notifierType>` value is 5 or 6.
<textLength>	Optional	0 - 1	Integer (5), variable	Length of `<textMessage>` in bytes. ***NOTE:*** Required if value is blank in `<messageType>`.

Exhibit 10-36. USER SERVICE NOTIFY<request> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<textMessage>	Optional	0 - 1	String (1024), variable	Free-format text of message. ***NOTE:*** Required if value is blank in `<messageType>`. ***NOTE:*** If used, `<textLength>` tag also required.
<workChangeRequest>	Optional	0 - 1	String (12), variable	Work order or change request ID. Substituted for variable in predefined message.

USER SERVICE NOTIFY — Replies

The reply message for this function contains no `<result>` data elements. It does, however, return a standard `<response>` data element to indicate the success or failure of the request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

SYSTEM ENVIRONMENT INFORMATION 11

System integrators may need programmatic access to system-level information about the ChangeMan ZMF started task and its working environment. XML Services provides the following such information for general use:

- *System Setup Parameter List - SYSTEM SERVICE LIST*
- *SERNET Environment Parameter List - SYSTEM ENVIRON LIST*
- *SERNET Security Group List - SYSTEM SECGROUP LIST*
- *ChangeMan ZMF Environment Parameters - ENVIRON SERVICE LIST*
- *ChangeMan ZMF STC DDNAME LIBRARIES - DSS SERVICE STCLIST*

System Setup Parameter List - SYSTEM SERVICE LIST

This multipurpose function accepts a setup keyword as input and returns its value for the current instance of ChangeMan ZMF. Typically these are the names of external software systems with which ChangeMan ZMF communicates, such as the security system, the scheduling system, the job review system, or the mail system.

The Serena XML service/scope/message tags and attributes for a system setup parameter *list* message are:

```
<service name=”SYSTEM”>
<scope name=”SERVICE”>
<message name=”LIST”>
```

These tags appear in both requests and replies.

SYSTEM SERVICE LIST — Requests

This function requires a single tag as input. This tag, `<systemTypeDesc>`, is not really the description of a system type, but rather a keyword used to retrieve a setup parameter value from a dataset coded in the ChangeMan ZMF started task procedure. All keywords entered in this tag must be in upper case.

Example XML — SYSTEM SERVICE LIST Request

```
<?xml version="1.0"?>
<service name="SYSTEM">
 <scope name="SERVICE">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
    <systemTypeDesc>SECURITY</systemTypeDesc>
   </request>
  </message>
 </scope>
</service>
```

Data structure details for the `<request>` data element appear in *Exhibit 11-1*.

Exhibit 11-1. SYSTEM SERVICE LIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<systemTypeDesc>	Required	1	String (8), variable	System setup parameter name or keyword for ZMF started task. Values: CHUNKING - SERNET chunking value CMN - ZMF license status DISTRIB - File distribution system FILE - File system(s) used JES4XJR - Job review system used MAIL - Mail system used SCHEDULE - Scheduler used SECURITY - Security system used ***NOTE:*** Entry is case-sensitive. ***NOTE:*** When value is "FILE", all supported file systems are requested as a group.

SYSTEM SERVICE LIST — Replies

This function returns zero to many `<result>` data elements. In most cases, a single `<result>` contains the requested parameter value in the `<systemName>` tag, or a null value if no such system type is used with ChangeMan ZMF. Two exceptions exist, however:

- ***Chunking Parameter*** — The value returned in `<systemName>` is not a system name, but the ChangeMan ZMF chunk size in bytes.
- ***File System Parameters*** — Multiple `<result>` tags are returned, one for each file system supported by this ChangeMan ZMF instance. Each `<result>` contains a file system name (e.g., "PDSE") in `<systemName>` and its corresponding ChangeMan ZMF code (e.g., "7") in `<systemType>`.

The table in *Exhibit 11-2* lists the possible values of `<systemName>` returned for any request parameter supplied in `<systemTypeDesc>`.

Exhibit 11-2. SYSTEM SERVICE LIST systemTypeDesc, systemName XREF

Parameter Name <systemTypeDesc>	**Returned Values** <systemName>
CHUNKING	Numeric; chunking size in bytes.

CMN	CMN = ZMF is licensed
	blank = ZMF not licensed

DISTRIB	NDM = Network Data Mover (AKA CONNECT:Direct) is ZMF file distribution system
	BDT = IBM bulk data transfer is ZMF file distribution system
	IEBCOPY = IBM IEBCOPY utility is ZMF file distribution system
	XCOM62 = Legent's XCOM62 is ZMF file distribution system
	FTP = File Transfer Protocol is ZMF file distribution system
	NETMASTR = NetMaster is ZMF file distribution system
	NETVIEW = Netview is ZMF file distribution system

FILE	IMS = IMS database mgmt system
	DB2 = DB2 database mgmt system
	PDS = Partitioned data sets
	PDSE = Partitioned data sets, extended
	SEQ = Sequential files
	LIB = CA Librarian
	LIBA = CA Librarian Archie
	PAN = CA Panvalet
	VSAM = VSAM files
	HFS = Hierarchical File System
	OTHER = Other file system

JES4XJR	JES2 = Job review system active under JES2 JES3 = Job review system active under JES3 blank = Job review system not active
MAIL	MVSSEND = MVS SEND messaging used for notifications EMC2 = EMC2 Tao mail system used for notifications EMAIL = E-mail system used for notifications PROFS = VM PROFS mail system used for notifications DISSOS = DISSOS mail system used for notifications MEMO = MEMO mail system used for notifcations CCMAIL = cc:Mail mail system used for notifications

Exhibit 11-2. SYSTEM SERVICE LIST systemTypeDesc, systemName XREF *(Continued)*

Parameter Name <systemTypeDesc>	Returned Values <systemName>

SCHEDULE	CMN = ZMF scheduler used
	CA7 = Computer Associates job scheduler
	ESP = ESP scheduling system
	OPCESA = OPC/ESA scheduling system
	JOBTRAC = JobTrac scheduling system
	CONTROLM = Control-m scheduling
	MANUAL = Manual job scheduling
	OTHER = Other scheduling system

SECURITY	SAF = IBM Security Access Facility
	RACF = IBM Resource Access Facility
	ACF2 = CA Access Control Facility
	TSS = CA Top Secret
	OS2/UPM = OS/2 User Profile Management

Following the `<result>` data element is the standard `<response>` data element, which indicates the success or failure of the XML request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Example XML — SYSTEM SERVICE LIST Reply

```
<?xml version="1.0"?>
<service name="SYSTEM">
 <scope name="SERVICE">
  <message name="LIST">
   <result>
    <systemName>SAF</systemName>
   </result>
   <result>
    <systemName>RACF</systemName>
   </result>
   <result>
    <systemName>ACF2</systemName>
   </result>
   <result>
    <systemName>TSS</systemName>
   </result>
   <result>
    <systemName>OS2/UPM</systemName>
   </result>
   <response>
    <statusMessage>. . . .
    <statusReturnCode>. . . .
    <statusReasonCode>. . . .
   </response>
  </message>
 </service>
```

Data structure details for the `<result>` data element appear in *Exhibit 11-3*.

Exhibit 11-3. SYSTEM SERVICE LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<systemType>	Optional	0 - 1	String (1)	Code corresponding to file system named in `<systemName>`. ***NOTE:*** Returned only for FILE parameter.
<systemName>	Optional	0 - 1	String (8), variable	Value of system setup parameter. Usually a system name such as RACF (for security) or FTP (for distribution). ***NOTE:*** Returns blank if requested system type is not used with ZMF. ***NOTE:*** For CHUNKING parameter, value is SERNET chunk size in bytes.

SERNET Environment Parameter List - SYSTEM ENVIRON LIST

This function accepts a required setup keyword — "ENVIRON" — as input. It returns the values of several parameters associated with the SERNET system environment in which the current instance of ChangeMan ZMF operates. These include server date and time, chunk size, and whether or not SERNET communication is enabled between ChangeMan ZMF and selected file management systems, job scheduling systems, and Serena products.

The Serena XML service/scope/message tags and attributes for a SERNET environment parameter *list* message are:

```
<service name="SYSTEM">
<scope name="ENVIRON">
<message name="LIST">
```

These tags appear in both requests and replies.

SYSTEM ENVIRON LIST — Requests

The syntax of a SERNET environment parameter list request is virtually identical to that for a system setup parameter list request. (See *Exhibit 11-1*.) However, the `<systemTypeDesc>` tag takes a special keyword: "ENVIRON". This value is case-sensitive.

The example shows exactly how you should code your SERNET environment parameter list request. As always, the `<header>` data element is required only for batch jobs. (See *"<header> Tag" in Chapter 2, XML Syntax Basics*.)

SYSTEM ENVIRON LIST — Replies

This function returns one `<result>` data element containing the requested parameters and values. Following the `<result>` data structure is the standard `<response>` data structure, which indicates the success or failure of the XML request and provides a status message.

Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Data structure details for the `<result>` data element appear in *Exhibit 11-4*.

Exhibit 11-4. SYSTEM ENVIRON LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<chunkSize>	Optional	0 - 1	String (8), variable	ZMF started task chunk size in bytes.
<cmnDesDevStatus>	Optional	0 - 1	String (1)	Code for license status of ChangeMan ZDD. Values: **0** = Not licensed **2** = Licensed
<cmnStatus>	Optional	0 - 1	String (1)	Code for license status of ChangeMan ZMF. Values: **0** = Not licensed **2** = Licensed
<cpxStatus>	Optional	0 - 1	String (1)	Code for license status of Comparex. Values: **0** = Not licensed **2** = Licensed
<currentSystemApplid>	Optional	0 - 1	String (3)	Current system application (for example CMN).
<currentSystemDateStamp>	Optional	0 - 1	Date, yyyymmdd	Current system date.
<currentSystemTimeStamp>	Optional	0 - 1	Time, hhmmss	Current system time, 24-hour format
<db2Status>	Optional	0 - 1	String (1)	Code for license status of IBM DB2. Values: **0** = Not licensed **1** = Licensed & inactive **2** = Licensed & active
<eclStatus>	Optional	0 - 1	String (1)	Code for license status of Eclipse. Values: **0** = Licensed **2** = Not licensed
<endevorStatus>	Optional	0 - 1	String (1)	Code for license status of Endevor. Values: **0** = Licensed **2** = Not licensed
<hfsStatus>	Optional	0 - 1	String (1)	Code for SERNET support status of IBM hierarchical file system (HFS). Values: **0** = Supported **2** = Not supported
<hsmStatus>	Optional	0 - 1	String (1)	Code for SERNET support status of IBM hierarchical storage management system (HSM). Values: **0** = Supported **2** = Not supported

Exhibit 11-4. SYSTEM ENVIRON LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<imsStatus>	Optional	0 - 1	String (1)	Code for license status of IBM IMS. Values: **0** = Not licensed **1** = Licensed & inactive **2** = Licensed & active
<libStatus>	Optional	0 - 1	String (1)	Code for SERNET support status of CA Librarian. Values: **0** = Supported **2** = Not supported
<panStatus>	Optional	0 - 1	String (1)	Code for SERNET support status of CA Panvalet. Values: **0** = Supported **2** = Not supported
<passphrase>	Optional	0 - 1	String (1)	Code for support status of passphrase. Values: **0** = Supported **2** = Not supported
<pdseStatus>	Optional	0 - 1	String (1)	Code for SERNET support status of IBM PDSE file system. Values: **0** = Supported **2** = Not supported
<sdsbStatus>	Optional	0 - 1	String (1)	Code for SERNET support status of SDSB. Values: **0** = Supported **2** = Not supported
<serverCCSID>	Optional	0 - 1	String (5)	Server CCSID
<serverVersion>	Optional	0 - 1	String (5)	Server version.
<starToolStatus>	Optional	0 - 1	String (1)	Code for license status of StarTool FDM. Values: **0** = Not licensed **2** = Licensed
<syncTracStatus>	Optional	0 - 1	String (1)	Code for license status of ChangeMan SSM. Values: **0** = Not licensed **2** = Licensed
<unicodeNationalChars>	Optional	0 - 1	String (12), variable	Unicode national characters in hex.
<xchStatus>	Optional	0 - 1	String (1)	Code for license status of ChangeMan ZDD/ XCH. Values: **0** = Not licensed **2** = Licensed

Exhibit 11-4. SYSTEM ENVIRON LIST <result> Data Structure *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<xjrJesType>	Optional	0 - 1	String (1)	Job review JES type. Values: **0** = Not supported **2** = JES2 **3** = JES3
<xjrStatus>	Optional	0 - 1	String (1)	Code for license status of IBM job review system. Values: **0** = Not licensed **1** = Licensed & inactive **2** = Licensed & active
<xsdStatus>	Optional	0 - 1	String (1)	Software delivery status. Values: **0** = Not licensed **2** = Licensed

SERNET Security Group List - SYSTEM SECGROUP LIST

This function lists the RACF groups to which a user is connected.

The Serena XML service/scope/message tags and attributes for a SERNET security group *list* message are:

```
<service name="SYSTEM">
<scope name="SECGROUP">
<message name="LIST">
```

These tags appear in both requests and replies.

SYSTEM SECGROUP LIST — Requests

This function does not accept input data; the request is performed for the user who is running it. The `<request>` tag itself is required, however, to identify the message as a request rather than a reply.

Note

XML syntax allows both a long form and a short form for empty tags. An empty `<request>` tag can therefore be coded in one of two ways.

Long form:

```
<request>
</request>
```

Equivalent short form:

```
<request/>
```

The following example shows how to code the request with an empty `<request>` tag.

Note

Be sure to code "XCH" in the `<product>` subtag, which will return security groups for the user. If you code "CMN" in the `<product>` subtag, you will receive a list of groups to which the started task is connected (because CMN tasks run under the started task's security environment, rather than the user's).

Example XML — SYSTEM SECGROUP LIST Request

```
<?xml version="1.0"?>
<service name="SYSTEM">
 <scope name="SECGROUP">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>XCH</product>
   </header>
  <request>
   </request>
  </message>
 </scope>
</service>
```

SYSTEM SECGROUP LIST — Replies

This function returns zero to many `<result>` data elements, listing all of the RACF security groups to which the current user is connected. Following the `<result>` data structure is the standard `<response>` data structure, which indicates the success or failure of the XML request and provides a status message. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Data structure details for the `<result>` data element appear in *Exhibit 11-5*.

Exhibit 11-5. SYSTEM SECGROUP LIST <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<groupName>	Optional	0 - 1	String (7), variable	RACF security group name.

ChangeMan ZMF Environment Parameters - ENVIRON SERVICE LIST

This function returns the values of ChangeMan ZMF environment parameters for the current started task and the current user. (For batch jobs, the "current" started task is the ChangeMan ZMF instance identified in the `<subsys>` tag in the `<header>` data element of the request.) Returned values include the environment type, the started task job name and subsystem ID, the ChangeMan ZMF options licensed, and the product features (such as administrator access) available to the current user.

The Serena XML service/scope/message tags and attributes for a ChangeMan ZMF environment parameter *list* message are:

```
<service name=”ENVIRON”>
<scope name=”SERVICE”>
<message name=”LIST”>
```

These tags appear in both requests and replies.

ENVIRON SERVICE LIST — Requests

Because this function accepts no input data, the Serena XML request message includes an empty `<request>` tag. The `<request>` tag itself is required, however, to identify the message as a request rather than a reply.

Note

XML syntax allows both a long form and a short form for empty tags. An empty `<request>` tag can therefore be coded in one of two ways.

Long form:

```
<request>
</request>
```

Equivalent short form:

```
<request/>
```

The example shows how to code a ChangeMan ZMF environment parameter list request with an empty `<request>` tag.

Example XML — ENVIRON SERVICE LIST Request

```
<?xml version="1.0"?>
<service name="ENVIRON">
 <scope name="SERVICE">
  <message name="LIST">
   <header>
    <subsys>8</subsys>
    <product>CMN</product>
   </header>
  <request>
   </request>
  </message>
 </scope>
</service>
```

ENVIRON SERVICE LIST — Replies

This function returns one `<result>` data element containing the ChangeMan ZMF environment parameter values for the started task responding to the request. Following the `<result>` data structure is the standard `<response>` data structure, which indicates the success or failure of the XML request. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Example XML — ENVIRON SERVICE LIST Reply

```
<?xml version="1.0"?>
<service name="ENVIRON">
 <scope name="SERVICE">
  <message name="LIST">
   <result>
    <startedTaskJobName>SERT8</startedTaskJobName>
    <startedTaskUser>SERT</startedTaskUser>
    <subSystemId>8</subSystemId>
    <cmnEnvironmentType>3</cmnEnvironmentType>
    <hasGlobalAccess>Y</hasGlobalAccess>
    <hasAdminAccess>Y</hasAdminAccess>
    <hasMonitorAccess>Y</hasMonitorAccess>
    <hasBackoutAccess>Y</hasBackoutAccess>
    <hasRevertAccess>Y</hasRevertAccess>
    <hasReleaseManAccess>Y</hasReleaseManAccess>
    <db2SubSystemId>DSN</db2SubSystemId>
    <imsSubSystemId>C113</imsSubSystemId>
    <cmnResourceClass>$CMNTP</cmnResourceClass>
    <enableTestMode>Y</enableTestMode>
    <enableNetTrace>N</enableNetTrace>
    <printTraceConsole>Y</printTraceConsole>
    <shutdownInProgress>N</shutdownInProgress>
    <disableCmnScheduling>N</disableCmnScheduling>
    <reinitParm>Y</reinitParm>
    <isPanPresent>N</isPanPresent>
    <isLibrPresent>N</isLibrPresent>
    <isRlsSupported>N</isRlsSupported>
    <isSmsVsamPresent>N</isSmsVsamPresent>
    <isDb2Licensed>Y</isDb2Licensed>
    <isInfoSystemInstalled>Y</isInfoSystemInstalled>
    <isImsLicensed>Y</isImsLicensed>
    <isCdfLicensed>Y</isCdfLicensed>
    <isOfmLicensed>Y</isOfmLicensed>
    <isLoadBalancingLicensed>Y</isLoadBalancingLicensed>
    <isEroLicensed>Y</isEroLicensed>
    <maximumSiteCount>00255</maximumSiteCount>
   </result>
   <response>
    <statusMessage>CMN8700I - Environment service completed</statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

Data structure details for the `<result>` data element appear in *Exhibit 11-6*.

Exhibit 11-6. ChangeMan ZMF Environment Parameter List <result>

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<cmnEnvironmentType>	Optional	0 - 1	String (1)	Code for ZMF environment type. Values: **1** = All development & production handled by same ZMF instance; no remote production sites **2** = Development ZMF instance only **3** = Development & production handled by same ZMF instance; remote production sites also supported **4** = Production ZMF instance only
<cmnResourceClass>	Optional	0 - 1	String (8), variable	Security resource class for ZMF.
<db2SubSystemId>	Optional	0 - 1	String (4), variable	DB2 subsystem ID used by ZMF.
<disableCmnScheduling>	Optional	0 - 1	String (1)	**Y** = Yes, disable ZMF scheduling **N** = No, don't disable ZMF scheduling
<enableNetTrace>	Optional	0 - 1	String (1)	**Y** = Yes, net trace on **N** = No, net trace off
<enableTestMode>	Optional	0 - 1	String (1)	**Y** = Yes, test mode on **N** = No, test mode off
<hasAdminAccess>	Optional	0 - 1	String (1)	**Y** = Yes, user has administrator authority **N** = No administrator authority
<hasBackoutAccess>	Optional	0 - 1	String (1)	**Y** = Yes, user has backout authority **N** = No backout authority
<hasGlobalAccess>	Optional	0 - 1	String (1)	**Y** = Yes, user has global authority **N** = No global authority
<hasMonitorAccess>	Optional	0 - 1	String (1)	**Y** = Yes, user has monitor authority **N** = No monitor authority
<hasReleaseManAccess>	Optional	0 - 1	String (1)	**Y** = Yes, user has release mgr authority **N** = No release manager authority
<hasRevertAccess>	Optional	0 - 1	String (1)	**Y** = Yes, user has revert authority **N** = No revert authority
<imsSubSystemId>	Optional	0 - 1	String (4), variable	IMS subsystem ID used by ZMF.

Exhibit 11-6. ChangeMan ZMF Environment Parameter List <result> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<isApsLicensed>	Optional	0 - 1	String (1)	**Y** = Yes, APS licensed **N** = No, APS not licensed
<isCdfLicensed>	Optional	0 - 1	String (1)	**Y** = Yes, Merge+Reconcile licensed **N** = No, Merge+Reconcile not licensed
<isDb2Licensed>	Optional	0 - 1	String (1)	**Y** = Yes, DB2 licensed **N** = No, DB2 not licensed
<isEroLicensed>	Optional	0 - 1	String (1)	**Y** = Yes, ERO licensed **N** = No, ERO not licensed
<isImsLicensed>	Optional	0 - 1	String (1)	**Y** = Yes, IMS licensed **N** = No, IMS not licensed
<isInfoSystemInstalled	Optional	0 - 1	String (1)	**Y** = Yes, INFO installed **N** = No, INFO not installed
<isLibrPresent>	Optional	0 - 1	String (1)	**Y** = Yes, CA Librarian present **N** = No, CA Librarian not present
<isLoadBalancingLicensed>	Optional	0 - 1	String (1)	**Y** = Yes, Load Balancing Option licensed **N** = No, Load Balancing not licensed
<isOfmLicensed>	Optional	0 - 1	String (1)	**Y** = Yes, Online Forms Option licensed **N** = No, Online Forms Option not licensed
<isPanPresent>	Optional	0 - 1	String (1)	**Y** = Yes, CA Panvalet present **N** = No, CA Panvalet not present
<isRlsSupported>	Optional	0 - 1	String (1)	**Y** = Yes, record-level sharing (RLS) active **N** = No, record-level sharing not active
<isSmsVsamPresent>	Optional	0 - 1	String (1)	**Y** = Yes, SMS/VSAM (RLS) present **N** = No, SMS/VSAM (RLS) not present
<licensedLine1>	Optional	0 - 1	String (56), variable	Licensed to line 1.
<licensedLine2>	Optional	0 - 1	String (56), variable	Licensed to line 2.
<maximumSiteCount>	Optional	0 - 1	String (1)	Maximum number remote sites supported.
<printTraceConsole>	Optional	0 - 1	String (1)	**Y** = Yes, trace to SERPRINT/WTO **N** = No, don't trace to SERPRINT/WTO
<printTraceOnly>	Optional	0 - 1	String (1)	**Y** = Yes, trace to SERPRINT only **N** = No, don't trace to SERPRINT only
<reinitParm>	Optional	0 - 1	String (1)	**Y** = Yes, PARM=REINIT **N** = No, reinitialization parameter not set
<shutdownInProgress>	Optional	0 - 1	String (1)	**Y** = Yes, shutdown in progress **N** = No, not shutting down
<startedTaskJobName>	Optional	0 - 1	String (8), variable	ZMF started task job name.

Exhibit 11-6. ChangeMan ZMF Environment Parameter List <result> *(Continued)*

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<startedTaskUser>	Optional	0 - 1	String (8), variable	TSO user ID of current started task user.
<subsystemId>	Optional	0 - 1	String (1)	ZMF subsystem ID.

ChangeMan ZMF STC DDNAME LIBRARIES - DSS SERVICE STCLIST

This function retrieves the dataset and/or library concatenation list for a specified DDNAME associated with the ChangeMan ZMF started task that processes the request.

The Serena XML service/scope/message tags and attributes for a ChangeMan ZMF library concatenation *list* message are:

```
<service name="DSS">
<scope name="SERVICE">
<message name="LIST">
```

These tags appear in both requests and replies.

DSS SERVICE STCLIST — Requests

The request message for the library concatenation list function requires the DDNAME for the ChangeMan ZMF started task as input. Data structure details for the `<request>` tag appear in *Exhibit 11-7*.

Exhibit 11-7. DSS SERVICE STCLIST <request> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<stcDDname>	Required	1	String (8), variable	JCL DDNAME in the ZMF started task procedure.

DSS SERVICE STCLIST — Replies

This function returns zero to many `<result>` data elements. Each `<result>` contains the fully qualified dataset name of a library, together with its ordinal position in the concatenation named in the request message by `<stcDDname>`.

Following the final `<result>` tag is the standard `<response>` data element, which indicates the success or failure of the XML request and provides a status message. Successful requests have a return code of 00. Unsuccessful requests have a return code of 04 or higher.

Data structure details for the `<result>` data element appear in *Exhibit 11-8*.

Exhibit 11-8. Library Concatenation List <result> Data Structure

Subtag	Use	Occurs	Data Type & Length	Values & Dependencies
<mvsLib>	Optional	0 - 1	String (255), variable	Fully qualified dataset name.
<mvsLibSequence>	Optional	0 - 1	Integer (3), variable	Ordinal position of `<mvsLib>` dataset in library concatenation for ZMF instance.

XMLSERV – INTERACTIVE XML PROTOTYPING TOOL

A

XMLSERV is a host application that runs under ISPF on z/OS mainframes. It provides an interactive user interface to SERXMLBC, the Serena XML batch processing client for ChangeMan ZMF. (For more information about SERXMLBC, see *"SERXMLBC – Executing Native XML Service Calls" on page 575*.)

Use XMLSERV to perform the following tasks:

- ***Prototype and test*** Serena XML requests to ChangeMan ZMF interactively, then validate the XML replies.
- ***Browse the contents of ChangeMan ZMF master files*** interactively using Serena XML requests and replies.

Caution

Do not use XMLSERV in ISPF split screen mode. If you use split screen mode with XMLSERV on one screen and ChangeMan ZMF on the other, you may corrupt the system.

Caution

Input and output dataset allocations for XMLSERV have been changed in ZMF 7.1.2 to support long names. Any `*userid*.XMLIN` and `*userid*.XMLOUT` datasets from ZMF 7.1.1 or earlier must be deleted before running XMLSERV for ZMF 7.1.2 or later.

XMLSERV Functional Overview

When you first start XMLSERV, it displays a list of XML functions on the main screen.

```
  File  Help
-------------------------------------------------------------------------------
                           XML Services                             Row 1 from 194
Command ===>                                                        SCROLL ===> HALF
Member .. TEMPNAME Test ..   Subsys .. Z Product .. CMN
S  Service  Scope    Message  Color    Copybook
   APPROVER APL      LIST     G        ****AAPR
   APPROVER PKG      LIST     G        ****PAPR
   BASELIB  SERVICE  LIST     G        ****BASL
   CALENDAR SERVICE  LIST     G        ****CLDR
   CMPONENT GBL_DPRC LIST     G        ****GDCP
   CMPONENT APL_DPRC CHECK    G        ****ADCP
   CMPONENT APL_DPRC FIND     G        ****ADCP
   CMPONENT APL_DPRC LIST     G        ****ADCP
   CMPONENT GBL_SECR LIST     G        ****GCSC
   CMPONENT APL_SECR CHECK    G        ****ACSC
   CMPONENT APL_SECR LIST     G        ****ACSC
   CMPONENT APL_SECR FIND     G        ****ACSC
   CMPONENT CHG_DESC LIST     G        ****PSVD
   CMPONENT GBL_CDSC LIST     G        ****GCGD
   CMPONENT APL_CDSC LIST     G        ****ACGD
   CMPONENT APL_CDSC FIND     G        ****ACGD
   CMPONENT HISTORY  LIST     G        ****CHIS
   CMPONENT HISTORY  LISTCONC G        ****CHIS
```

The main screen lists valid Serena XML functions in alphabetical order by service name. Names in the **Service**, **Scope**, and **Message** columns correspond to the `name` attributes for the Serena XML `<service>`, `<scope>`, and `<message>` tags, respectively. Only valid combinations are shown.

The Color column flags each service with the level of caution recommended for its use:

Color	Meaning
G (Green)	Equivalent to an ISPF interface function in most cases. No special training is required prior to use.
Y (Yellow)	Implements a low-level subset of an ISPF function without checks. ***Caution must be exercised to prevent database integrity issues.*** Special customer training is highly recommended before use.

Note

Yellow services are not normally displayed in XMLSERV. Contact Serena Customer Support for Yellow services training, licensing information, and instructions on how to include Yellow services on XMLSERV panel displays.

The **Copybook** column provides the stem name of the copybook used with each service. These names can be combined with the service color to determine the names of the sample COBOL-to-XML copybooks to be used with SERXMLCC. Sample COBOL copybooks are prefixed with `XMLC` for green services and `XMLY` for yellow services. For example, the copybook for the APPROVER.APL.LIST service would be XMLCAAPR. Copybooks are delivered in the SERCOMC ASMCPY library unloaded from the ChangeMan ZMF installer.

Use the PF8 key to page through the list and find the function you want. Type an "S" beside the function to select it. When you select a function, XMLSERV supplies all the valid XML subtag names for the corresponding `<request>` message. The displayed tag names serve as data entry prompts. After entering the data needed for your request, submit the XML request to ChangeMan ZMF directly from XMLSERV. ChangeMan ZMF executes the request and returns an XML reply. XMLSERV displays the `<reply>` message in XML format online.

Main Screen Menu Options

The XMLSERV main screen provides menu-driven shortcuts to frequently performed tasks. These appear under the File and Help menus.

File Menu

Open	Opens an existing XML input document for editing in XMLSERV. You will be offered a list of the members in the input dataset.
Exit	Exit from XMLSERV.

Help Menu

Help	Provides online information about how to use XMLSERV.
Keys Help	Displays PF key definitions for use with XMLSERV.
About	Gives release-level information about XMLSERV.

Main Screen Primary Commands

XMLSERV supports the following primary commands at the main screen to facilitate its use.

ABOUT Command

The About command displays the 'About' panel, which includes the XML DSECT create date.

Syntax:

```
ABOUT
```

FILTER Command

The Filter command filters the list of XML functions to show only entries matching the specified criteria. Parameters are positional but optional. Parameter values may be masked with a trailing asterisk (`*`) wild card character.

Syntax:

```
FILTER (service|*) (scope|*) (message|*) (color|*) (copybook|*)
```

Examples:

`FILTER SITE`	Lists all functions with the service name `SITE`.
`FILTER * * UNFR*`	Lists all functions with a message name beginning in `'UNFR'`, such as `UNFREEZE`.
`FILTER PACKAGE * * Y`	Lists all Yellow functions with the service name `PACKAGE`.
`FILTER`	Lists all functions for all services. (Default)

The FILTER command uses the ISPF TBSARG service, which locates the scroll position as close to its original position as possible. If you issue a second FILTER command without first restoring the complete list (FILTER command with no parameters) and moving to the top of the list, you may need to scroll up to see the new results.

FIND Command

The Find command finds the next entry that contains a specified string in the service, scope or message name. No masking is available for the search string. If no search string is specified, the previous Find command is repeated.

Syntax:

```
FIND string
```

Examples:

`FIND HIST`	Finds the next XML function with a service, scope, or message name containing the string `'HIST'`.
`FIND`	Repeats the previous Find command. (Default)

LOCATE Command

The Locate command finds the next entry that matches the specified criteria for one or more parameters. Parameters are positional but optional. Parameter values may be masked with a trailing asterisk (`*`) wild card character. If no parameters are specified, the previous Locate command is repeated.

Syntax:

```
LOCATE (service|*) (scope|*) (message|*) (color|*) (copybook|*)
```

Examples:

`LOCATE SITE`	Finds the next function with the service name `SITE`.
`LOCATE * * UNFR*`	Finds the next function with a message name beginning in `'UNFR'`, such as `UNFREEZE`.
`LOCATE`	Repeats the previous Locate command. (Default)

LPRINT Command

The LPRINT command prints the currently displayed list of XML services to DDNAME XMLPRINT. By default, XMLPRINT is allocated to SYSOUT.

Syntax:

```
LPRINT
```

OPEN Command

The Open command opens the named member in the XML input dataset for editing with XMLSERV. If no member name is specified, the member list for the dataset is displayed.

Syntax:

```
OPEN (member)
```

RUN Command

The Run command submits the edited XML member to ChangeMan ZMF for execution.

By default, consecutive blanks in strings are collapsed to a single blank (in compliance with the XML standard) and blank tag entries are dropped before transmission over the network. However, if you wish preserve tag indentation or if blank tag values are intended for submission to ZMF and should not be dropped, use the optional `NODROP` parameter.

Syntax:

```
RUN [NODROP]
```

Examples:

`RUN`	Submits the currently open XML file to ZMF for execution. All blank tags are dropped and consecutive blanks in strings are compressed to single blanks before submission. (Default)
`RUN NODROP`	Submits the currently open XML file to ZMF for execution. All blanks are preserved.

SORT Command

The Sort command sorts the list of XML services by column name or column number. The first column in the parameter list identifies the outermost level of the sort. The last column in the parameter list identifies the innermost level of the sort. If no parameters are supplied, the sort is performed by service name, then scope within service, then message within scope.

Syntax:

```
SORT (column_1) (column_2) ... (column_n)
```

Examples:

`SORT SCOPE`	Sorts function list by scope name.
`SORT MESSAGE SERVICE`	Sorts function list by message name, then service name within message.
`SORT 3 1`	Equivalent to `'SORT MESSAGE SERVICE'`.
`SORT`	Sorts function list by service name, then scope name within service, then message name within scope. (Default)

XML Input and Output Documents

For each XML request and response document in your session, XMLSERV creates a new dataset member in a PDS library. Your administrator sets up the input and output PDS names and dataset allocations when installing XMLSERV. These datasets are passed to SERXMLBC for execution, so they should be allocated using SERXMLBC specifications. (See *Appendix B, "SERXMLBC – Executing Native XML Service Calls," on page 575* for dataset allocation recommendations. See the *ChangeMan ZMF Installation Guide* for information about installing XMLSERV.)

Within the input and output libraries, you can create multiple input and output members. Each member contains the XML input request and the XML output result for a single request/response cycle. Input and output members for the same request/response cycle share the same member name. You can choose a different member name for each of several input/output cycles.

For example, let's say your input library is named *userid*.XMLIN and your output library is named *userid*.XMLOUT. You perform two tasks in a single XMLSERV session. The first task is a package search, so you assign the member name "PACKSRCH" to these input and output XML documents in XMLSERV. The second task is a component search, so you name its input and output document members "COMPSRCH" in XMLSERV. The resulting data set names at the end of your XMLSERV session would then be:

XML Input Documents	XML Output Documents
USER239.XMLIN(PACKSRCH)	USER239.XMLOUT(PACKSRCH)
USER239.XMLIN(COMPSRCH)	USER239.XMLOUT(COMPSRCH)

Because the input and output data sets are saved between XMLSERV sessions, you can reuse your XML request files either with XMLSERV or directly with the SERXMLBC client. Similarly, XML reply files can serve (after file format conversion from EBCDIC to ASCII) as input to XML-savvy reporting tools on the desktop.

Usage Notes

XMLSERV is a prototyping tool. It is provided as-is as an aid to systems integrators already familiar with the internals of ChangeMan ZMF. As such, it does not incorporate the fail-safes and self-checking you might expect to see in a general-purpose utility for end users. Serena recommends that its use be limited to ChangeMan ZMF administrators and systems programmers only.

SAMPLE XMLSERV SESSION

The following exercise walks you through a sample XMLSERV session. In this exercise, you will do the following:

- Start XMLSERV.
- Change the dataset member name for your XML documents to a temporary value.
- Identify the ChangeMan ZMF subsystem that will execute your XML request.
- Specify a test setting.
- Request a general package search using Serena XML.
- Browse the ChangeMan ZMF package search results in Serena XML format.
- Exit XMLSERV.

Step 1: Start XMLSERV

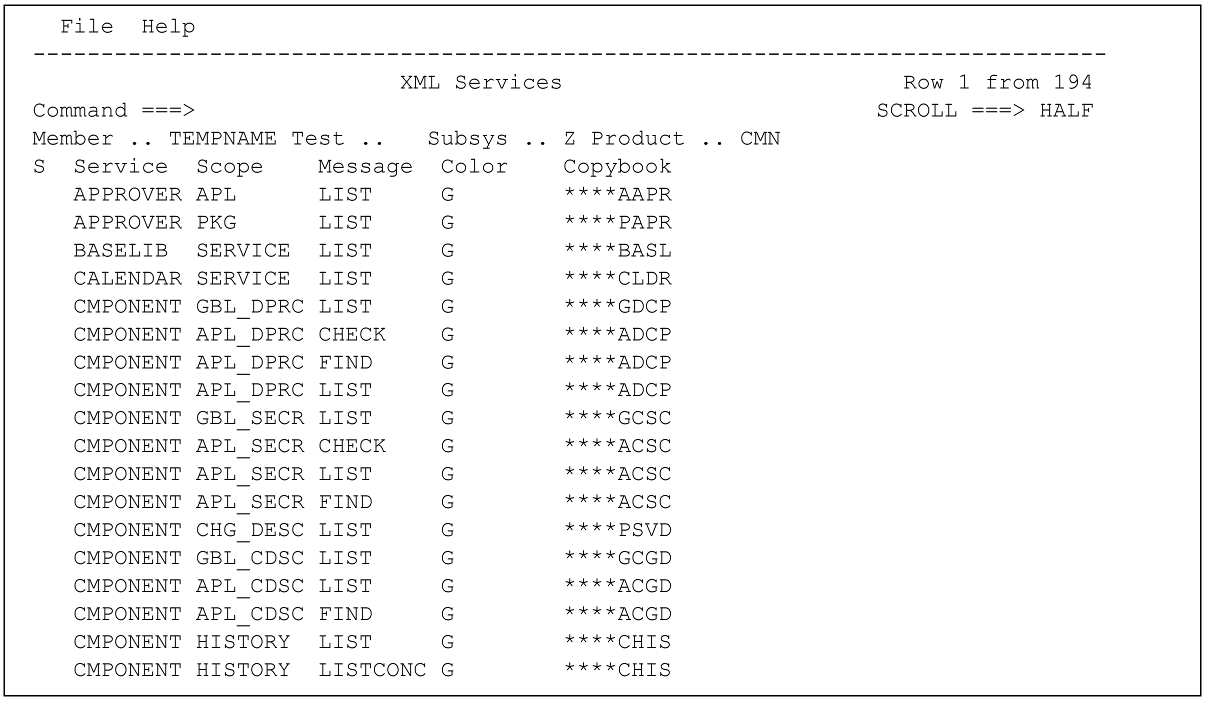
To start XMLSERV, first start ISPF. At the ISPF command line or `Option ==>` prompt, type:

```
TSO XMLSERV
```

Note

If XMLSERV does not start with this command, check with the ChangeMan ZMF administrator who installed XMLSERV for the correct library and/or member name.

XMLSERV establishes connections with the resources it requires, then displays the main screen. In this screen, XMLSERV displays its default parameter settings and lists the XML functions you can choose to execute in this session.



```
  File  Help
 ------------------------------------------------------------------------------
                          XML Services                           Row 1 from 194
 Command ===>                                                    SCROLL ===> HALF
 Member .. TEMPNAME Test ..   Subsys .. Z Product .. CMN
 S  Service  Scope    Message  Color    Copybook
    APPROVER APL      LIST     G        ****AAPR
    APPROVER PKG      LIST     G        ****PAPR
    BASELIB  SERVICE  LIST     G        ****BASL
    CALENDAR SERVICE  LIST     G        ****CLDR
    CMPONENT GBL_DPRC LIST     G        ****GDCP
    CMPONENT APL_DPRC CHECK    G        ****ADCP
    CMPONENT APL_DPRC FIND     G        ****ADCP
    CMPONENT APL_DPRC LIST     G        ****ADCP
    CMPONENT GBL_SECR LIST     G        ****GCSC
    CMPONENT APL_SECR CHECK    G        ****ACSC
    CMPONENT APL_SECR LIST     G        ****ACSC
    CMPONENT APL_SECR FIND     G        ****ACSC
    CMPONENT CHG_DESC LIST     G        ****PSVD
    CMPONENT GBL_CDSC LIST     G        ****GCGD
    CMPONENT APL_CDSC LIST     G        ****ACGD
    CMPONENT APL_CDSC FIND     G        ****ACGD
    CMPONENT HISTORY  LIST     G        ****CHIS
    CMPONENT HISTORY  LISTCONC G        ****CHIS
```

Step 2: Select an XML Service

The default parameters for your XMLSERV request appear at the top of the XMLSERV main screen. They include the member name where XML request and response documents should be saved, a setting for the diagnostic trace (“Test”) option, and the ChangeMan ZMF subsystem to use. You can override these defaults for the duration of your XMLSERV session.

Note

Generally, you should leave the “Test” field blank (i.e., no diagnostic trace data is collected for the SYSOUT log). Blank is the default. However, if you want to run your XML request in test mode and collect trace diagnostics in the ChangeMan ZMF SYSOUT log file, enter a “T” in the “Test” field.

1. Set the XMLSERV session parameters:

   a) ***Tab to the Member field*** in the upper right corner of the screen. The default XML input member name in this case is `TEMPNAME`. Change it to `PACKSRCH`.

   b) ***Tab to the Test field.*** Ensure that this value is blank — meaning no diagnostic trace is desired.

   c) ***Tab to the Subsystem field.*** Change the default subsystem ID to that used by your ChangeMan ZMF instance.

   Note

   For this exercise, the screen prints will show a subsystem ID of `Z` and an application name of `DEMO`.

2. Enter an XMLSERV command to find the PACKAGE.SEARCH.GENERAL function.

   At the* Command ---> *prompt, type `FIND PACKAGE GENERAL SEARCH`, and press Enter. XMLSERV skips down to the first service with a Service name of `PACKAGE`.

3. Select the XML function you wish to execute.

   Type an S in the S(elect) column in the row for PACKAGE.GENERAL.SEARCH.

```
  File  Help
--------------------------------------------------------------------------------
                           XML Services                         Row 85 from 194
Command ===>                                                   SCROLL ===> HALF
Member .. PACKSRCH Test ..   Subsys .. Z Product .. CMN
S  Service  Scope    Message  Color    Copybook
   PACKAGE  CMP_DESC LIST     G        ****PCDS
   PACKAGE  CMPONENT INTEGRTY G        ****PINT
   PACKAGE  FORMS    REFREEZE G        ****PFRZ
   PACKAGE  FORMS    UNFREEZE G        ****PFRZ
   PACKAGE  GEN_DESC LIST     G        ****PDSC
   PACKAGE  GEN_PRMS LIST     G        ****PGPM
   PACKAGE  GEN_PRMS REFREEZE G        ****PFRZ
   PACKAGE  GEN_PRMS UNFREEZE G        ****PFRZ
S  PACKAGE  GENERAL  SEARCH   G        ****PSCH
   PACKAGE  IMP_INST LIST     G        ****PIMI
   PACKAGE  IMS_ACB  LIST     G        ****PIAS
   PACKAGE  IMS_CRGN LIST     G        ****PICR
   PACKAGE  LIMBO    SEARCH   G        ****PSCH
   PACKAGE  NON_SRC  REFREEZE G        ****PFRZ
   PACKAGE  NON_SRC  UNFREEZE G        ****PFRZ
   PACKAGE  PKG_LINK LIST     G        ****PLNK
   PACKAGE  PKG_LINK SEARCH   G        ****PSCH
   PACKAGE  PRM_CMP  LIST     G        ****PPRC
```

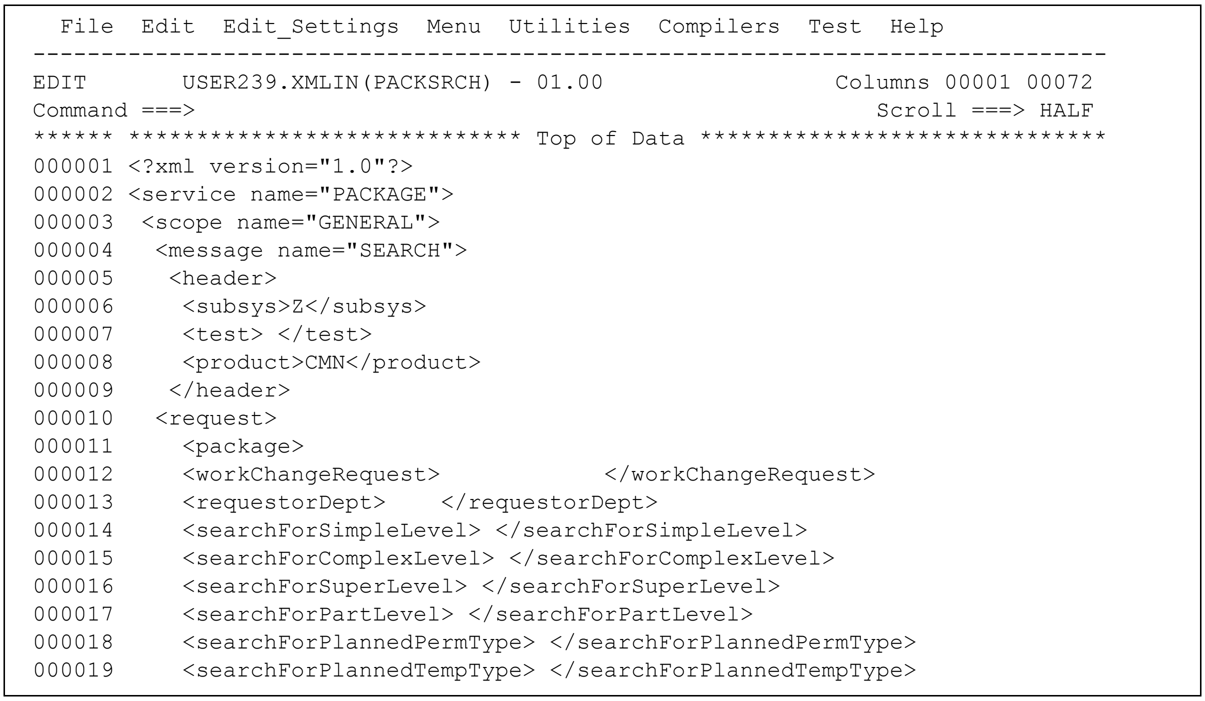
4. ***Press Enter***.

Step 3: Edit the XML Input Document

XMLSERV starts an ISPF edit session for the PACKSRCH member and displays an XML request template. The template contains all valid Serena XML tags predefined for use with the function you selected on the main screen. Most tags are optional and need not be used.

In this example, XMLSERV displays XML tags for the "Package General Search" function.

Your screen should look something like the following:



```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
EDIT       USER239.XMLIN(PACKSRCH) - 01.00                Columns 00001 00072
Command ===>                                                 Scroll ===> HALF
****** ***************************** Top of Data ******************************
000001 <?xml version="1.0"?>
000002 <service name="PACKAGE">
000003  <scope name="GENERAL">
000004   <message name="SEARCH">
000005    <header>
000006     <subsys>Z</subsys>
000007     <test> </test>
000008     <product>CMN</product>
000009    </header>
000010   <request>
000011     <package>
000012     <workChangeRequest>            </workChangeRequest>
000013     <requestorDept>    </requestorDept>
000014     <searchForSimpleLevel> </searchForSimpleLevel>
000015     <searchForComplexLevel> </searchForComplexLevel>
000016     <searchForSuperLevel> </searchForSuperLevel>
000017     <searchForPartLevel> </searchForPartLevel>
000018     <searchForPlannedPermType> </searchForPlannedPermType>
000019     <searchForPlannedTempType> </searchForPlannedTempType>
```

At the top of the XML document template, you will find the header information used to route your XML document to the correct ChangeMan ZMF instance for execution. The content shown in the XML `<header>` tags is the information you entered in *“Step 2: Select an XML Service” on page 568*. (For more information about Serena XML `<header>` tag syntax, see *“High-Level Tags in Serena XML” on page 35*.

Below the `<header>` tag block comes the `<request>` tag block. Opening and closing tag pairs appear on a single line. Between the opening and closing tags are blanks, which represent the maximum data length accepted by the tag. Each tag pair serves as a data entry prompt. XMLSERV assumes you already know what each tag means. Information about allowed data types, valid tag combinations, and the like appear earlier in this manual.

In this exercise, you will search for all packages in an application with package numbers starting in “0000”. Do the following:

Change the <package> tag in the XML `<request>` block to read as follows:

`<package>`*`appl`*`0000*`

where *appl* is your application ID.

The request for this service allows entry of multiple package names delimited by semicolon, so you must scroll to the right to see the closing `</package>` tag. You can leave the spaces between the end of your data and the closing tag, or you can eliminate the spaces.

Note

Be careful not to delete any of the lines after the </request> tag when you edit your XML input document. These are the matching terminators for the opening XML tags at the top of the document.

Step 4: Execute the Edited XML Request

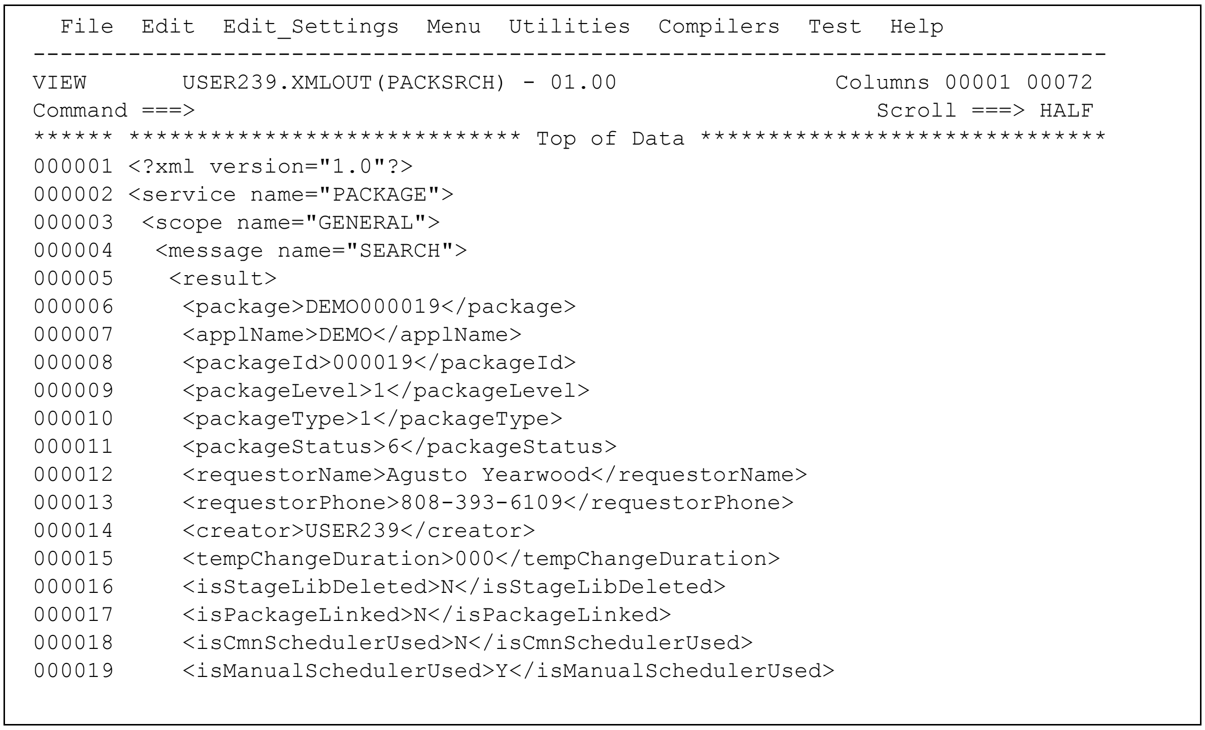
When you are satisfied with the contents of your XML input document, it's time to execute your XML Services request. To do this:

1. ***Type `RUN` at the ISPF command line*** or `COMMAND ===>` prompt.
2. ***Press Enter***.

XMLSERV saves the edits you made to your input document and passes it to SERXMLBC. SERXMLBC submits the request to the specified ChangeMan ZMF server, which executes the request and returns an XML reply to SERXMLBC. SERXMLBC saves the reply as an XML output document, then passes it back to XMLSERV. XMLSERV opens an ISPF session in which you can browse the returned XML.

Step 5: Browse the XML Output Document

XMLSERV starts an ISPF session to display your XML output document in view mode. The screen should look something like this:



```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
VIEW       USER239.XMLOUT(PACKSRCH) - 01.00                Columns 00001 00072
Command ===>                                                  Scroll ===> HALF
****** **************************** Top of Data *****************************
000001 <?xml version="1.0"?>
000002 <service name="PACKAGE">
000003  <scope name="GENERAL">
000004   <message name="SEARCH">
000005    <result>
000006     <package>DEMO000019</package>
000007     <applName>DEMO</applName>
000008     <packageId>000019</packageId>
000009     <packageLevel>1</packageLevel>
000010     <packageType>1</packageType>
000011     <packageStatus>6</packageStatus>
000012     <requestorName>Agusto Yearwood</requestorName>
000013     <requestorPhone>808-393-6109</requestorPhone>
000014     <creator>USER239</creator>
000015     <tempChangeDuration>000</tempChangeDuration>
000016     <isStageLibDeleted>N</isStageLibDeleted>
000017     <isPackageLinked>N</isPackageLinked>
000018     <isCmnSchedulerUsed>N</isCmnSchedulerUsed>
000019     <isManualSchedulerUsed>Y</isManualSchedulerUsed>
```

Page down through the document with PF8. Each package that matches your search criteria appears in a `<result>` block. Nested within the opening `<result>` and closing `</result>` tag delimiters for the block, you will find XML subtags detailing the many items found in the package general record for matching packages. Multiple `<result>` blocks, one for each package found, may be returned in response to a package search request.

Following the final `<result>` block is the `<response>` block for this XML output document. The `<response>` block contains a return code and any status messages concerning the execution of your XML request. For a successful package search request, the `<response>` block looks like this:

```
<response>
 <statusMessage>CMN8600I - The Package search list is complete.</statusMessage>
 <statusReturnCode>00</statusReturnCode>
 <statusReasonCode>8600</statusReasonCode>
</response>
```

Always check the `<response>` block to verify that your XML request executed successfully.

Because XMLSERV displays the XML output in ISPF view mode, you can use standard ISPF edit commands to review the data. This gives you a handy way to see a short list of all packages that satisfied your search criteria. Do this:

At the ISPF command line or `Command ===>` prompt, type the following instructions to filter the contents of the XML output document:

`X ALL; F ALL <package>`

ISPF shows the results as follows:

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
VIEW       USER239.XMLOUT(PACKSRCH) - 01.00              72 CHARS '<PACKAGE>'
Command ===>                                                 Scroll ===> HALF
****** ***************************** Top of Data ******************************
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -   7 Line(s) not Displayed
000006     <package>DEMO000019</package>
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 35 Line(s) not Displayed
000042     <package>DEMO000020</package>
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 35 Line(s) not Displayed
000078     <package>DEMO000021</package>
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 35 Line(s) not Displayed
000114     <package>DEMO000022</package>
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 34 Line(s) not Displayed
000149     <package>DEMO000023</package>
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 35 Line(s) not Displayed
000185     <package>DEMO000027</package>
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 42 Line(s) not Displayed
000228     <package>DEMO000028</package>
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 41 Line(s) not Displayed
000270     <package>DEMO000029</package>
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 35 Line(s) not Displayed
000306     <package>DEMO000030</package>
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  - 35 Line(s) not Displayed
```

Type the RESET command to show all the XML output data once again.

Step 6: Return to the XML Input Document and Exit

When you have finished viewing the XML output document, do the following to finish your XMLSERV session:

1. ***Press PF3*** to return to the XML input document.

   Notice that your request was saved by XMLSERV in condensed form, with null tags removed. Also, spaces before the closing </package> tag are deleted:

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
 -------------------------------------------------------------------------------
 EDIT       USER239.XMLIN(PACKSRCH) - 01.00                Columns 00001 00072
 Command ===>                                                  Scroll ===> HALF
 ****** ***************************** Top of Data ******************************
 000001 <?xml version="1.0"?>
 000002 <service name="PACKAGE">
 000003  <scope name="GENERAL">
 000004   <message name="SEARCH">
 000005    <header>
 000006     <subsys>5</subsys>
 000007     <product>CMN</product>
 000008    </header>
 000009   <request>
 000010     <package>demo0000*</package>
 000011    </request>
 000012   </message>
 000013  </scope>
 000014 </service>
**********************************BottomofData****************************
```

Note

If you think you might want to change an XMLSERV input document after it has been saved, you can keep null tags and trailing spaces by using this form of the RUN command:

```
RUN NODROP
```

2. ***Press PF3*** to return to the XMLSERV main screen.
3. ***To exit XMLSERV, press PF3 or type Exit*** at the ISPF command line or Command ===> prompt.

SERXMLBC – Executing Native XML Service Calls

B

Run batch client program SERXMLBC to execute native XML calls to XML Services. The load module for this program is delivered in the SERCOMC LOAD library in the ChangeMan ZMF installer.

SERXMLBC reads an XML request message from a file and preprocesses the message to ensure a minimum level of well-formed syntax with required tags present. It then passes the message to the SERCLIEN messaging client that handles the connection to the ChangeMan ZMF server. When SERCLIEN receives the XML reply, it passes the reply to SERXMLBC for delivery to an output file.

Input Requirements

SERXMLBC reads an XML request message at DD statement XMLIN. Requirements for input include:

- The input file is a sequential data set, PDS member, or PDSE member.
- DCB requirements for file are flexible, but the following is suggested to accommodate long tags:

```
RECFM=VB
LRECL=255
```

- Request messages must be formatted with one tag per record.
- Tag indenting is allowed (leading spaces are X’40’), but indenting is not required and no information is conveyed by indenting.

Output Requirements

SERXMLBC writes an XML reply message to DD statement XMLOUT. Requirements for output include:

- The output file is a sequential data set, PDS member, or PDSE member.
- The following DCB parameters are set internally by SERXMLBC:

```
RECFM=VB
LRECL=5000
```

- Reply messages are formatted with one tag per record.
- Tag indenting in reply messages expresses tag hierarchy.

JCL Requirements

The following is an example of the JCL you might use to run SERXMLBC:

```
//XML       EXEC PGM=SERXMLBC
//*
//STEPLIB   DD DISP=SHR,DSN=somnode.CMNZMF.LOAD
//          DD DISP=SHR,DSN=somnode.SERCOMC.LOAD
//*
//SER#PARM  DD DISP=SHR,DSN=somnode.TCPIPORT
//SYSPRINT  DD SYSOUT=*
//SERPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//XMLIN     DD DISP=SHR,DSN=somnode.XMLIN(member_name)
//XMLOUT    DD DISP=SHR,DSN=somnode.XMLOUT(member_name)
```

Note the following when you build JCL to execute SERXMLBC:

- The **STEPLIB** statement should use the same load library concatenation that you use for the STEPLIB in the SERNET started task JCL.
- The **SER#PARM** ddname should point to the same partitioned data set you used for the SER#PARM ddname in the SERNET started task JCL. This data set stores TCP/IP addresses and port numbers for Serena applications.
- The **XMLIN** ddname should point to the data set that contains your XML request message.
- The **XMLOUT** ddname should point to the data set that receives the XML reply message.

Return Codes and ABENDs

The following return codes are generated by SERXMLBC. Other return codes are passed through from the low-level service objects on the server through SERXMLBC.

Return Code	General Description
00	Successful completion.
04	Warning message generated by low-level service object on server. Nonfatal error.
08	XML tag is missing a required `name` attribute. Fatal error.
12	XML tag contains a value that is the wrong length. Fatal error.
16	A mandatory XML tag is missing. Fatal error.
24	SERXMLBC is unable to load the SERCLIEN messaging client software. Fatal error.

SERXMLBC terminates abnormally if it cannot open your XML input file. You will receive the following ABEND message:

```
SER622W - Unable to open XMLIN - abending
```

SERXMLAC – CALLING XML SERVICES FROM ASSEMBLER

C

To execute XML Services from an assembler program, invoke client program SERXMLAC. The load module for this program is delivered in the SERCOMC LOAD library in the ChangeMan ZMF installer.

The caller supplies an input buffer containing a well formed XML request, and the reply is returned in an output buffer, wrapped in the appropriate XML tags. The caller is then responsible for parsing the XML reply in the output buffer. The caller preallocates the input and output buffers and must make sure they are big enough to contain the request and reply data.

As with all XML service data, unused tags are not required in the request buffer, and empty tags are not returned in replies.

Program SERXMLAC may be called from ChangeMan ZMF exit programs to access XML Services.

SERXMLAC PARAMETER LIST

SERXMLAC requires a 4-word parameter list as follows:

- +0 = Length of the XML request area.
- +4 = Address of the XML request.
- +8 = Length of the XML reply area.
- +12 = Address of the XML reply area.

You determine the amount of storage allocated for the reply buffer. Once the reply buffer is full, SERXMLAC sends no more data to the service call.

Return Codes and Reason Codes

The following return codes (R15) and reason codes (R0) are generated by SERXMLAC.

Return Code	Reason Code
00	00 - Successful completion
04	08 - No information found for this request
16	08 - The following tag missing name
16	12 - The following tag wrong length
16	16 - The following tag is mandatory
16	20 - Reply buffer exceeded
20	26 - Invalid parameter list

Sample Call to APPROVER PKG LIST

This section provides an example of how to call the APPROVER PKG LIST service from an assembler program.

Setting SERXMLAC Parameter List Values

In this example, we allocate 4K bytes for the request buffer and 32K bytes for the replies.

```
**********************************************************************
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
*
* Allocate storage areas for XML request and reply (SERXMLAC)
*
*        IXP$PARM +0  = length of request area
*        IXP$PARM +4  = address of request area
*        IXP$PARM +8  = length of reply area
*        IXP$PARM +12 = address of reply area
*        IXP$PARM +16 = address of current '<result>'
*        IXP$PARM +20 = total getmain'd storage
*
**********************************************************************
         XC    IXP$PARM,IXP$PARM   clear parmlist
         LA    R2,4+32             36k request
         SLL   R2,10 bits
       STORAGE OBTAIN,LENGTH=(2),  get storage
               COND=NO,SP=3,       subpool 3
               LOC=(ANY,ANY)       above, backed anywhere
         ST    R1,IXP$PARM+4       adr of request area
         ST    R2,IXP$PARM+20      total getmained length
         LA    R2,X09@RQLN         actual request length
         ST    R2,IXP$PARM         length of request area
         LA    R2,4                4k request
         SLL   R2,10 bits
         AR    R1,R2               bump +4k
         LA    R2,32               32k reply length
         SLL   R2,10 bits
         ST    R2,IXP$PARM+8       length of reply area
         ST    R1,IXP$PARM+12      adr of reply area
         ST    R1,IXP$PARM+16      save current result pointer
```

Building the XML Services Request Buffer

In this example, we are calling the APPROVER PKG LIST service. The request must include the package name and subsystem ID at tags `<package>` and `<subsys>` respectively.

XML Request Area

The following code shows the request area for the APPROVER PKG LIST request.

```
***********************************************************************
*
*        XML Request
*
***********************************************************************
X09@XMLR DS    0H                  XML request block
         DC    C'<?xml version="1.0"?>'
         DC    C'<service name="APPROVER">'
         DC    C' <scope name="PKG">'
         DC    C'   <message name="LIST">'
         DC    C'    <header>'
         DC    C'     <subsys>'
X09@SUBS EQU   *-X09@XMLR,1
         DC    C'x'
         DC    C'</subsys>'
         DC    C'     <test>T</test>'
         DC    C'     <product>CMN</product>'
         DC    C'    </header>'
         DC    C'   <request>'
         DC    C'     <package>'
X09@PKGN EQU   *-X09@XMLR,10
         DC    C'aaaannnnnn'
         DC    C'</package>'
         DC    C'    </request>'
         DC    C'  </message>'
         DC    C' </scope>'
         DC    C'</service>'
X09@RQLN EQU   *-X09@XMLR          request length
```

Setting Request Tag Values

To make the program reentrant and reusable, move the request block to allocated storage, and then move data values to fields imbedded in the appropriate tag names.

In this example, the subsystem ID comes from `X09@SUBS` and the package name comes from `X09@PKGN`.

```
**********************************************************************
*
*        Move request to SERXMLAC request buffer
*
**********************************************************************
         L     R0,IXP$PARM+4       target request area
         L     R1,IXP$PARM         target request length
         LA    R14,X09@XMLR        source tags
         LR    R15,R1              ditto request length
         MVCL  R0,R14              source request to area
**********************************************************************
*
*        Move service filtering data to SERXMLAC request buffer
*
**********************************************************************
         L     R1,IXP$PARM+4       restore request area
         MVC   X09@SUBS(,R1),IXP$SUBS    ZMF subsys id
         MVC   X09@PKGN(,R1),IXP$PNAM    package name
```

Calling SERXMLAC

After the request buffer has been populated, the address of the parameter list is passed in register 1, per normal convention, and SERXMLAC is called.

```
**********************************************************************
*
*        Call SERXMLAC with service request.
*
**********************************************************************
         LA    R1,IXP$PARM         parameter list for serxmlac
         ST    R1,IXP$WORK         store parmlist address
         LOAD  EP=SERXMLAC
         LR    R15,R0              epa
         LA    R1,IXP$WORK
         BASSM R14,R15             call it
         LTR   R15,R15             okay?
         BNZ   X09$9000            .no, error
```

Processing the Reply Buffer

All replies are returned from the call in a single burst. The caller is responsible for ensuring that the amount of storage allocated for the reply buffer is large enough to hold as many results as are expected or required.

Reply data is wrapped in XML tags, starting with the usual header tags (XML version, service, scope, and message), one or more sets of results tags, followed by the response tags (return message, return code, and reason code).

When you process replies, scan the reply buffer for the tags you are interested in, and examine the enclosed data. The buffer may contain multiple sets of result tags, so scan for the `<result>` tag first, which indicates the beginning of one result. Each result is terminated by a `</result>` ending tag. The `<response>` tag indicates the end of all results.

GETTAG Subroutine

This is the code for the GETTAG subroutine that parses the reply buffer to extract package approver information returned by XML Services.

```
**********************************************************************
*        GETTAG - find tag at R1
*
*Input   R0 = length of tag value
*        R1 = adr of tag value
*        XML$PARM+16 start of search area
*
*Output  R0 = length of data value
*        R1 = adr of data value
*        R15= +0 tag and data found
*        R15= +4 tag not found before </result>
*        R15= +8 tag not found before </response>
**********************************************************************
GETTAG   DS    0H
         BAKR  R14,0                stack registers
         LR    R14,R1               save tag value adr
         LR    R3,R0                save tag value length
         BCTR  R3,0                 -1 for ex instr
         L     R4,XML$PARM+16       start of search
GETT0100 DS    0H
         TRT   0(256,R4),TRTBL      look for chevron
         BZ    GETT9100             not found - end of data
         CLC   0(9,R4),=C'</result>' end of result?
         BE    GETT9000             .yes, end of this result
         CLC   0(11,R4),=C'</response>' end of response?
         BE    GETT9100             .yes, end of data
         EX    R3,GETTCMPR          compare tag with request
         LA    R4,1(,R4)            bump past chevron
         BNE   GETT0100             not found, keep looking
         LA    R4,0(R3,R4)          bump over tag
         LR    R5,R4                save start of reply data
         TRT   0(256,R4),TRTBL      look for next chevron
         BZ    GETT9100             not found - some error
         SR    R1,R5                R1 = length of data
         LR    R0,R1                R0 = length of data
         LR    R1,R4                R1 = adr of data
         XR    R15,R15              good return code
         B     GETT9900             return
GETT9000 DS    0H
         LR    R1,R4               adr of </result>
         LA    R15,4               </result> found before tag
         B     GETT9900            return
GETT9100 DS    0H
         LR    R1,R4               adr of </response>
         LA    R15,8               </response> found before tag
*        B     GETT9900            return
GETT9900 DS 0H
         PR    ,                   unstack, return
**********************************************************************
*        DATA AREAS
**********************************************************************
GETTCMPR CLC 0(*-*,R4),0(R14)      matching tag name?
         LTORG
**********************************************************************
* Translate table for '<'
**********************************************************************
TRTBL    DC    256AL1(0)
         ORG   TRTBL+C'<'
         DC    C'<'
         ORG
```

GETTAG Subroutine Processing

This sample GETTTAG subroutine scans the reply buffer searching for a tag pointed to by register 1. If a </result> tag is discovered before the required tag name is found, the return code is set to +4, indicating the end of the current reply. If a </response> tag is discovered before the required tag name is found, the return code is set to +8, indicating that all results have been exhausted.

The sequence of events for managing reply tags should be:

1. Point Register 1 (R1) at the start of the reply buffer.
2. Scan for the first <result> tag.

3. Set R1 (stored in IXP$PARM+16 in this example) to the start of the <result>.
4. Point R1 to a tag name you wish to interrogate.
5. Call the GETTAG routine. If found, the data within the tag will be located at R1, and its length in R0.
6. Continue calling the GETTAG routine with whatever tag names you wish to interrogate.
7. When GETTAG returns RC=4 (end of <result>), point R1 at the next set of <result>s, and go to step 3.
8. When GETTAG returns with RC=8 (end of <response>), all results have been processed.

SERXMLCC - Calling XML Services from COBOL

D

COBOL-to-XML Copybooks

To execute Serena XML Services calls, you need not work directly with XML and SERXMLBC. Instead, you can use COBOL copybooks in your custom COBOL programs.

ChangeMan ZMF provides a set of copybooks that enable COBOL programs to generate Serena XML message streams. The copybooks wrap literal strings containing XML tag names and other syntax around COBOL variables. Your COBOL program populates these variables with the values you want to submit to ChangeMan ZMF in your Serena XML request. The data values for these COBOL variables are identical to those you would use to populate their corresponding XML data elements.

If Serena XML Services returns one or more XML `<result>` tag, all tags are removed and the result is presented to the user program as a table of COBOL variables with one row for each `<result>`.

Copybook Member Names

In many cases, there is one COBOL-to-XML copybook for each combination of `<service>`, `<scope>`, and `<message>` names supported by Serena XML Services. Some copybooks are used for more than one `<scope>` and `<message>` combination. Copybook member names have the following form:

XMLC***xxxx***

— where ***xxxx*** is a four-letter mnemonic for one or more service/scope/message combinations.

For example, XMLCPAPV is the COBOL copybook file that accesses the “package approval” function of the package management service object. It is associated with the following `name` attributes in the XML `<service>`, `<scope>`, and `<message>` tags:

```
<service name=”package”>
<scope name=”service”>
<message name=”approve”>
```

Cross reference tables showing COBOL copybook names for each of the XML services is provided in *Exhibit 2-10* (for core ZMF functionality) and *Exhibit 2-11* (for ERO services).

Caution

Do not modify the COBOL-to-XML copybooks. These code modules must remain synchronized with various XML-to-DSECT mapping files in order to work correctly. If you have a requirement that is not easily met using the COBOL-to-XML copybooks, you should use Serena XML Services directly.

COBOL Variable Names

The COBOL variable names used in the COBOL-to-XML copybooks are of two types: control variables and content variables.

Control Variables

Control variable names appear in all capitals. Most control variables are used by the COBOL batch client subroutine, SERXMLCC, to manipulate the data exchanged with user programs. Some control variables map to the Serena XML Services subtags within the `<header>` and `<response>` data structures. The function of a control variable is consistent across all COBOL-to-XML copybooks.

The table below lists the COBOL control variables used in all COBOL-to-XML copybooks.

COBOL Variable Name	XML Parent Tag	XML Subtag Name	Page #
REQUEST-BUFFER-LENGTH	N/A	N/A	N/A
RESULT-COUNT	N/A	N/A	N/A
HEADER-SUBSYS	<header>	<subsys>	
HEADER-TEST	<header>	<test>	
HEADER-SCOPE	<service>	<scope>	
HEADER-MESSAGE	<scope>	<message>	
RESULT-RC	<response>	<statusReturnCode>	
RESULT-REASON	<response>	<statusReasonCode>	
RESULT-MESSAGE	<response>	<statusMessage>	

Content Variables

Content variables in the COBOL copybooks map one-to-one against Serena XML Services subtags within the `<request>` and `<result>` data structures. Different sets of content variables appear in each COBOL-to-XML copybook.

COBOL content variable names correspond closely to the names of their matching Serena XML Services tags. Given an XML tag name, you can obtain its equivalent COBOL variable name by applying the following transformation rules:

- ***Input variables*** (i.e., those corresponding to XML tags in the `<request>` data structure) are prefixed in COBOL with **`I-`**.
- ***Output variables*** (i.e., those corresponding to XML tags in the `<result>` data structure) are prefixed with **`O-`**.
- ***Underscores*** in XML tag names are replaced by hyphens in COBOL variable names.
- ***Delimiting angle brackets*** are not part of the XML tag name and are omitted from the corresponding COBOL variable name.
- ***Case is preserved***. Upper-case characters in XML tag names are upper-case in COBOL variable names. Lower-case characters in XML tag names are lower-case in COBOL variable names.
- ***XML tag names are truncated*** to 30 characters in COBOL variable names. When the I/O prefix is added, the XML tag names are truncated to 28 characters.

The table below shows example transformations from XML tag names to their COBOL variable names. This subset is selected from the "list general parameters" function of the package management service object. The relevant copybook name is XMLCPGPM.

XML Parent Tag	XML Subtag Name	COBOL Variable Name
<request>	<package>	I-package
	<applName>	I-applName
	<isLinkedPackage>	I-isLinkedPackage
<result>	<package>	O-package
	<applName>	O-applName
	<packageLevel>	O-packageLevel
	<packageType>	O-packageType
	<packageStatus>	O-packageStatus
	<dateFrozen>	O-dateFrozen
	<dateApproved>	O-dateApproved
	<dateInstalled>	O-dateInstalled
	<isLinkedPackage>	O-isLinkedPackage

Data Types, Values, and Constraints

Data type, value constraints, and interdependencies for a given XML tag (or group of tags) apply equally to any corresponding COBOL variable(s) in a COBOL-to-XML copybook. This information is detailed in the *Serena XML Services Reference Tables*, which contain a series of indexed HTML files that describe each service.

Tip

Serena XML Services data restrictions are defined broadly enough to accommodate both mainframe and distributed ChangeMan products. Where ChangeMan ZMF has a more restrictive internal data requirement than the XML interface allows (e.g., for data length), the more restrictive requirement should be followed.

Input/Output Buffers

All COBOL-to-XML copybooks place a REQUEST-BUFFER and a RESULT-BUFFER data structure in your COBOL program's working storage section. These data structures organize the COBOL variables of the copybook for I/O processing.

The REQUEST-BUFFER data structure formats your populated COBOL variables into a Serena XML Services request message. Included in REQUEST-BUFFER are control variables for the XML `<header>` data structure, content variables for the XML `<request>` data structure, and appropriate XML tags coded as literals in VALUE clauses. Your program passes the populated REQUEST-BUFFER to program SERXMLCC, the COBOL batch subroutine client for Serena XML Services.

The RESULT-BUFFER data structure contains a parsed XML reply message that is returned to your program by SERXMLCC. The RESULT-BUFFER contains up to four subordinate data structures.

This subordinate data structure is included in all reply messages.

- **STATUS-MESSAGES** — Contains COBOL variables for return code, reason code, and message (i.e., the parsed contents of the Serena XML Services `<response>` tag). STATUS-MESSAGES is always present in a reply.

These three subordinate structures appear only if a Serena XML Services `<result>` is expected.

- **RESULT-TAGS** — Contains Serena Services XML tag names and their internal lookup codes for all permitted subtags in the `<result>` type expected for a particular COBOL copybook. These are literals provided for XML reply processing, and they provide no information that you can use in your COBOL user program.
- **RESULT-COUNT** — Control variable populated with the number of XML `<result>` data structures returned by Serena XML Services. RESULT-COUNT provides the maximum value of a table index for the RESULT-AREA data structure below. If XML Services returns no XML `<result>` data structure, then RESULT-COUNT is zero.
- **RESULT-AREA** — An table of parsed XML `<result>` data structures. Each table row contains the populated COBOL content variables that correspond to the subtags in a single instance of an XML `<result>` data structure returned by Serena XML Services.

The subtags are not included in a RESULT-AREA table row. The number of populated table rows in RESULT-AREA is indicated in RESULT-COUNT.

COBOL Batch Subroutine Client SERXMLCC

To process requests and receive replies from the SERERNA XML Services interface, your custom COBOL program must call the COBOL batch client program, SERXMLCC. You issue this call after initializing and populating a set of request variables in the REQUEST-BUFFER data structure of a particular COBOL copybook. Your custom COBOL program may issue only one Serena XML Services request at a time. The associated reply — which is delivered to the RESULT-BUFFER data structure by SERXMLCC — should be processed before another request is sent.

SERXMLCC preprocesses your generated XML request message to ensure a minimum level of well-formed syntax and to verify that all required XML tags are present. It then passes your message to the SERCLIEN messaging client for delivery to SERNET and the ChangeMan ZMF server. On receipt of an XML reply, SERXMLCC parses the XML message and populates the COBOL variables in the RESULT-BUFFER.

The SERXMLCC subroutine is designed to run in batch mode only.

Compiling Programs That Call SERXMLCC

Your custom program that uses COBOL-to-XML copybooks and calls SERXMLCC observe the following JCL requirements.

Compile JCL

The SYSLIB DD statement in your compile JCL must include the Serena SERCOMC ASMCPY library that you unloaded from the ChangeMan ZMF installer. This is where the COBOL-to-XML copybooks reside.

Make dynamic COBOL calls to SERXMLCC and use compile parameter DYNAM to avoid having to relink your program each time a new version of SERXMLCC is released by Serena.

Link-edit or Binder JCL

If you choose to link SERXMLCC statically to your custom program, then the SYSLIB DD statement in your link-edit JCL must include the Serena SERCOMC LOAD library you unloaded from the ChangeMan ZMF installer. This is where the COBOL batch subroutine client SERXMLCC and all other Serena subroutines reside.

Link edit your program into your custom CMNZMF LOAD library.

Running Programs That Call SERXMLCC

When you call SERXMLCC from your user program, you connect indirectly to the SERNET started task with the subsystem ID that you moved to the HEADER-SUBSYS variable in the COBOL-to-XML copybook in your program.

The libraries you use in the execution JCL for your programs must be the same as the libraries in the SERNET started task JCL.

- STEPLIB or JOBLIB — Use the same load library concatenation that you use in the STEPLIB for the SERNET started task JCL.
- SER#PARM — Point to the same PDS that is coded in the SERNET started task JCL at the SER#PARM statement.

Return Codes

COBOL batch subroutine client SERXMLCC generates the following return codes.

Return Code	General Description
00	Successful completion.
04	Warning message generated by low-level service object on server. Nonfatal error.
08	XML tag is missing a required `name` attribute. Fatal error.
12	XML tag contains a value that is the wrong length. Fatal error.
16	A mandatory XML tag is missing. Fatal error.

Other reason codes and error messages are passed through from the low-level service objects in ChangeMan ZMF to your COBOL program in the STATUS-MESSAGES variables in the COBOL copybooks.

Sample COBOL Program CMNOPSCH

Program CMNOPSCH delivered in the CMNZMF SAMPLES library is an example of a COBOL program that calls Serena XML Services through the COBOL batch client program, SERXMLCC. The sample program uses the PACKAGE GENERAL SEARCH service to list packages with IDs that fit mask JONH00057* in the ChangeMan ZMF instance that runs under SERNET subsystem L.

Program CMNOPSCH passes XML requests to SERXMLCC using the PSCH02-REQUEST data structure in copybook XMLCPSCH. Replies from SERXMLCC are received in the PSCH02-RESULT data structure in the same copybook.

These are the significant processing steps in program CMNOPSCH.

1. Prepare request data in the PSCH02-REQUEST data structure in copybook XMLCPSCH in Working Storage.
   a) Initialize the entire REQUEST-BUFFER.
   a) Set `<subsys>` name by moving a literal to the HEADER-SUBSYS variable. The literal value is the subsystem ID of the target SERNET instance.

   b) Set `<package>` name by moving a literal to the I-PACKAGE variable. The literal value is the search mask for the package search operation.

   c) Set the REQUEST-BUFFER-LENGTH variable in copybook XMLCPSCH by using the MOVE LENGTH OF operator.

2. Call COBOL batch client program SERXMLCC, passing the address of the REQUEST-BUFFER in the PSCH02-REQUEST data structure and the RESULT-BUFFER in the PSCH02-RESULT in copybook XMLCPSCH in Working Storage.

3. Process data returned by SERXMLCC in the PSCH02-RESULT data structure in copybook XMLCPSCH in Working Storage.

   a) If the STATUS-RETURN-CODE is 0, increment Working Storage subscript RESULT-SUBSCR from 1 to the value of RESULT-COUNT returned by SERXMLCC, and display data from each occurrence of RESULT-AREA in the RESULT-BUFFER.

      Note: Working Storage subscript STATUS-SUBSCR is used to translate the numeric code in O-PACKAGESTATUS into the familiar three-character abbreviation for package status in lookup table STATUS-VALUES in Working Storage.

   b) If the STATUS-RETURN-CODE value is not 0, display the values of the STATUS variables in the RESULT-BUFFER, and set the return code for program CMN0PSCH to the STATUS-RETURN-CODE.

4. Exit the program.

Compile, Link, and Execution JCL for CMNOPSCH

This JCL shows steps to compile and link edit sample COBOL program CMNOPSCH with dynamic calls to XML Services batch client program SERXMLCC.

Note

You must copy the source for sample program CMNOPSCH from the delivered CMNZMF SAMPLES library to a library with LRECL=80 to input the source to the compiler. The SAMPLES library has LRECL=4096.

```
//COBOL     EXEC  PGM=IGYCRCTL,
//                PARM='LIB,DYNAM',
//                REGION=2048K
//STEPLIB     DD  DSN=SYS2.ECOB310.SIGYCOMP,DISP=SHR
//SYSPRINT    DD  SYSOUT=*
//SYSLIB      DD  DSN=somnode.SERCOMC.ASMCPY,DISP=SHR
//SYSIN       DD  DSN=somnode.CMNZMF.CUSTOM.SAMPLES(CMNOPSCH),
//                DISP=SHR
//SYSLIN      DD  DSN=&&LOADSET,
//                DISP=(MOD,PASS),
//                UNIT=SYSDA,SPACE=(TRK,(3,3)),
//                DCB=(BLKSIZE=3200)
//SYSUT1      DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT2      DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
```

```
//SYSUT3     DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT4     DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT5     DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT6     DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSUT7     DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//*
//LKED      EXEC PGM=HEWL,
//              COND=(4,LT),
//              REGION=1024K
//SYSLIB     DD DSN=CEE.SCEELKED,DISP=SHR
//SYSPRINT   DD SYSOUT=*
//SYSLIN     DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//SYSLMOD    DD DSN=somnode.SERCOMC.CUSTOM.LOAD(CMNOPSCH),
//              DISP=SHR
//SYSUT1     DD UNIT=SYSDA,SPACE=(TRK,(10,10))
//*
//EXECUTE   EXEC PGM=CMNOPSCH,
//              COND=(4,LT)
//STEPLIB    DD DSN=somnode.CMNZMF.CUSTOM.LOAD,DISP=SHR
//           DD DSN=somnode.SERCOMC.CUSTOM.LOAD,DISP=SHR
//           DD DSN=somnode.CMNZMF.LOAD,DISP=SHR
//           DD DSN=somnode.SERCOMC.LOAD,DISP=SHR
//           DD DSN=CEE.SCEERUN,DISP=SHR
//SYSPRINT   DD SYSOUT=*
//SYSUDUMP   DD SYSOUT=*
//SNAPOUT    DD SYSOUT=*
//SER#PARM   DD DSN=somnode.SERCOMC.TCPIPORT,DISP=SHR
//SERPRINT   DD SYSOUT=*
```

Display from Sample Program CMNOPSCH

This is an example of the data displayed to SYSOUT by sample program CMNOPSCH.

```
PACKAGE    STA TYP LVL INSTALL  CREATOR
========== === === === ======== ========
DEMO000021 DEV PLN SMP 20081024 USER240
DEMO000022 DEV PLN SMP 20090323 USER240
DEMO000023 DEV PLN SMP 20081023 USER240
DEMO000027 BAS PLN SMP 20081117 USER239
DEMO000028 DEV PLN SMP 20081015 USER239
DEMO000029 DEV PLN SMP 20081025 USER239
```

SERXMLRC - CALLING XML SERVICES FROM REXX

E

REXX batch execution client SERXMLRC provides an interface to Serena XML Services from REXX programs.

The function of SERXMLRC is to generate Serena XML documents from REXX stem variables. The generated Serena XML data stream is validated for well-formed XML syntax, then routed by the SERCLIEN interface on the client to SERNET on the host. SERNET passes the XML request to the ChangeMan ZMF instance named in the `<header>` tag.

ChangeMan ZMF output is passed back to SERXMLRC as a separate XML document. This document is parsed, and the Serena XML tag names are presented to the calling REXX program in the same stem as the input. The raw XML result is also made available to the caller of the service should native parsing be a requirement of the caller.

Your custom REXX program creates and populates the request stem variable, calls SERXMLRC, and processes the REXX result.

SERXMLRC can run in batch, and it can be called from ISPF panel exits. The load module for SERXMLRC is delivered with other XML Services batch clients in the SERCOMC LOAD library in the ChangeMan ZMF ChangeMan ZMF installer.

SAMPLE JCL TO INVOKE XML REXX EXEC

The following is an example of JCL used to invoke CMN010. This JCL is included in member REPORTS in the CMNZMF CNTL library delivered in the ChangeMan ZMF installer. The REXX EXEC, which is included in the CMNZMF REX library, produces the Summary of Planned and Unplanned Packages report:

```
//REXX     EXEC PGM=IRXJCL,REGION=0M,
//         PARM='CMN010 ABCD 1 USER000 T'  <=== Change Accordingly
//* ZMF/SERCOMC LOAD LIBRARIES
//STEPLIB  DD DISP=SHR,DSN=somnode.CMNZMF.LOAD
//         DD DISP=SHR,DSN=somnode.SERCOMC.LOAD
//* REXX LIBRARIES
//SYSEXEC  DD DISP=SHR,DSN=somnode.CMNZMF.REX
//* SER#PARM
//SER#PARM DD DISP=SHR,DSN=somnode.SERCOMC.TCPIPORT
//* TCP/IP OUTPUT
//SYSPRINT DD SYSOUT=*
//* XML REPORT OUTPUT
//SYSTSPRT DD SYSOUT=*,DCB=(RECFM=FBA,LRECL=133)
//* XML DIAGNOSTIC TRACE DATA
//SERPRINT DD SYSOUT=*
```

```
//* ABEND OUTPUT
//SYSABEND DD SYSOUT=*
```

SAMPLE REXX EXEC CMN010 PROLOGUE

```
/* REXX                                                               */
/* ****************************************************************** */
/* Copyright 2003-2007 (C) SERENA Software, Inc.                      */
/* Licensed material.  All rights reserved.                           */
/* ChangeMan is a registered trademark of SERENA (R) Software Inc.    */
/* ****************************************************************** */
/* USE OF THE SAMPLE CODE CONTAINED HEREIN IS SUBJECT TO THE TERMS    */
/* CONDITIONS OF THE LICENSE AGREEMENT LOCATED IN THE MEMBER LICENSE  */
/* ****************************************************************** */
/* Date       Author             Reason                               */
/* 2003-06-01 Serena             Original version                     */
/* ****************************************************************** */
/* REXX CMN010 Summary of Planned and Unplanned Packages              */
/*                                                                    */
/* This report makes use of two XML Services                          */
/*                                                                    */
/*   Service     Scope    Message   Description                       */
/*                                                                    */
/* 1 PARMS       APL      LIST      Obtain the list of Appl. names    */
/* 2 PACKAGE     SUMMARY  SERVICE   Obtain counts about Package types */
/*                                  and statuses                      */
/*                                                                    */
/* Parameters                                                         */
/*                                                                    */
/* Application name         1 to 4 character mnemonic which may       */
/*                          include the asterisk '*' character to     */
/*                          represent a wild card. If omitted '*'     */
/*                          is assumed. Omission is indicated by a '.'*/
/*                          in the parm list.                         */
/*                                                                    */
/* Subsystem letter         1 character indicative of the ChangeMan   */
/*                          system that is being reported upon. Must  */
/*                          be present. A '.' indicates the default   */
/*                          subsystem of ' ' (blank).                 */
/*                                                                    */
/* TSO userid               1 to 8 character TSO id used to perform   */
/*                          security checking. Required parameter.    */
/*                                                                    */
/* Test switch              An indicator with the value 'T' which     */
/*                          specifies that diagnostic trace           */
/*                          information is to be sent to the          */
/*                          SERPRINT DD. The default is no value      */
/*                          other words Tracing is Off.               */
/*                          Since this is the last parm a positional  */
/*                          placeholder is not required.              */
/*                                                                    */
/* ****************************************************************** */
```

SAMPLE REXX EXEC CMN010 MAINLINE

```
/* Mainline ************************************************************ */
arg arguments

/* validate users input parms */

 ok=CMN000(CMN010 arguments)

 if ok= 'OK' then nop
            else call Error_Message /* never to return */

/* Read input parms */

arg appname subname tsoname tst .

/* initialize variables and set defaults where appropriate */

call Init_Variables

/* set up first xml service call */

call Init_XMLStem1

/* make first xml service call */

call Serxmlrc

/* for each application returned perform 2nd XML call */

do jx=1 to SER1.result.0
  call Init_XMLStem2               /* set up 2nd XML call */
  call Serxmlrc                    /* make 2nd XML call */
  if rxrc=0 then call Output_result /* if ok, print out result */
end

/* Print out totals */

call Output_Totals

/* terminate ZMF session */

call Disconnect
```

Notes:

This sample consists of three stages: The initialization of variables, the calls to SERXMLRC and the presentation of the results.

SAMPLE REXX EXEC CMN010 XML SETUP and CALL

```
/* Set variables for XML call */

Init_XMLStem1:
     rxrc           = 0        /* initialize our return code */
     stem           = "SER1." /* set outgoing stem name */
     SER2.          = ""       /* initialize outgoing stem */  NOTE 1
     SER1.          = ""       /* initialize outgoing stem */
     SER1.Subsys    = subname /* subsystem name to query */   NOTE 2
     SER1.Userid    = tsoname /* userid */
     SER1.Test      = tst      /* set test value */
     SER1.Product   = "CMN "   /* set product */              NOTE 3
     SER1.Service   = "PARMS" /* set service*/               NOTE 4
     SER1.Message   = "LIST"   /* set message */
     SER1.Scope     = "APL"    /* set scope   */
     SER1.applname = appname /* set application name */
                               /* set result set to return */
     SER1.includeInResult.1 = "applName"
Return
```

Notes:

1. This clears the stem and defines all possible values starting with an empty string.
2. This nominates the subsystem that you are communicating with.
3. The product code defaults to CMN. There may be other products later.
4. Both Service and Message are compulsory. Scope is optional and defaults to SERVICE if omitted.

```
/* XML service call */

Serxmlrc:
  address LINKMVS "SERXMLRC stem"
  rxrc=rc                               NOTE 1
  if rxrc<>0 then call Diagnose_Error  NOTE 2
Return
```

Notes:

1. Rxrc is a copy of the return code. The REXX variable rc is special in that it is the return code of the latest external operation. Therefore, it is good practice to record the rc in a separate variable after every external call.
2. Error diagnostics are only needed if there is an error.

SAMPLE REXX EXEC CMN010 XML PRINT OUTPUT

```
/* print output lines */

Output_Result:
  do ix=1 to SER2.result.0                                  NOTE 1
    ctSimple    = formit(SER2.totalsBySimpleLevel.ix)       NOTE 2
    ctComplex   = formit(SER2.totalsByComplexLevel.ix)
    ctSuper     = formit(SER2.totalsBySuperLevel.ix)
    ctPart      = formit(SER2.totalsByPartLevel.ix)
    ctPlanPerm  = formit(SER2.totalsByPlannedPermType.ix)
    ctPlanTemp  = formit(SER2.totalsByPlannedTempType.ix)
    ctUnplanPerm= formit(SER2.totalsByUnplannedPermType.ix)
    ctUnplanTemp= formit(SER2.totalsByUnplannedTempType.ix)
    call Print_Line left(SER1.applname.jx,11),
         right(ctSimple,8),
         right(ctComplex,8),
         right(ctSuper,8),
         right(ctPart,13),
         right(ctPlanPerm,9),
         right(ctPlanTemp,9),
         right(ctUnplanPerm,9),
         right(ctUnplanTemp,9)
    /* keep running totals */
    GTSimple =GTSimple+ctSimple
    GTComplex=GTComplex+ctComplex
    GTSuper  =GTSuper+ctSuper
    GTPart   =GTPart+ctPart
    GTPPerm  =GTPPerm+ctPlanPerm
    GTPTemp  =GTPTemp+ctPlanTemp
    GTUPerm  =GTUPerm+ctUnplanPerm
    GTUTemp  =GTUTemp+ctUnplanTemp
  end
Return
```

Notes:

1. Stem.result.0 is the number of results.
2. Stem.result.1 is the first result. Stem.result.2 is the 2nd and so on.

SAMPLE REXX EXEC CMN010 XML DIAGNOSE ERROR

```
/* Print out any error we do not expect, 6504 means 'no data found' */
/* and is not necessarily an error */

Diagnose_Error:
  Select
.
.
.
    when stem="SER1." then
```

```
      do
        if SER1.reasonCode<> '6504' then
          do
            call Print_Line "Return Code:" rc "from SERXMLRC"           NOTE 1
            call Print_Line "Subsystem  :" SER1.Subsys
            call Print_Line "Service    :",
                 SER1.Service SER1.Message SER1.Scope
            call Print_Line "Reason Code:" SER1.reasonCode              NOTE 2
            call Print_Status
            call Disconnect rc
          end
        else
          do
            call Print_Line "No applications found"
            call Disconnect 4
          end
      end
.
.
.
     end
    otherwise  nop
  end
Return
```

Notes:

1. When RC is not 0, then SERXMLRC completed abnormally.
2. When RC is not 0, there may be a reason code from the service.

SAMPLE REXX EXEC CMN010 XML DISCONNECT CODE

```
/* Disconnect and set return code */
Disconnect:

 arg exitcode
 if exitcode =' ' then exitcode ='0'
 call Init_XMLstem0
 call Serxmlrc
 drop SER0.
 exit exitcode

 /* Set variables for XML call */

 Init_XMLStem0:
      rxrc          = 0          /* initialize our return code */
      stem          = "SER0."    /* set outgoing stem name */
      SER0.         = ""         /* initialize outgoing stem */
      SER0.Subsys   = subname    /* subsystem name to query */
      SER0.Userid   = tsoname    /* userid */
      SER0.Test     = tst        /* set test value */
      SER0.Product = "CMN "      /* set product */
      SER0.Service = "         " /* set service */
      SER0.Message = "DISCONCT" /* set message */
      SER0.Scope   = "SERVICE"  /* set scope   */
 Return
```

All messages good and bad are recorded in the REXX variable STEM.Message.n. The number of messages is in STEM.Message.0. The most common error is when the subsystem is not up. In this case, the above example fails:

Following is the output for an unsuccessful execution of CMN010:

```
Report CMN010 generated from subsystem: 8 on: 24 Mar 2009 at: 09:42:35 Page: 1
-----------------------------------------
Summary of Planned and Unplanned Packages
-----------------------------------------
Application Simple   Complex  Super    Participating Planned   Planned   Unplanned Unplanned
Name        Packages Packages Packages Packages      Permanent Temporary Permanent Temporary
----------- -------- -------- -------- ------------- --------- --------- --------- ---------
Return Code: 8 from SERXMLRC
Subsystem  :
Service    :
Reason Code: 6028
Message    : SER6602I Using defined TEST option T
Message    : SER6604I Using specified IncludeInResult:  applName
Message    : SerNet started task "8" is not active (Error=5)
Message    : SerNet started task "8" is not active (Error=5)
Message    : Connection failed
Message    : SerNet started task "8" is not active (Error=5)
Message    : Connection failed
******************************** BOTTOM OF DATA ********************************************
```

Following is the output for a successful execution of CMN010:

```
Report CMN010 generated from subsystem: 8 on: 24 Mar 2009 at: 05:58:55 Page: 1
-----------------------------------------
Summary of Planned and Unplanned Packages
-----------------------------------------
Application Simple   Complex  Super    Participating Planned   Planned   Unplanned Unplanned
Name        Packages Packages Packages Packages      Permanent Temporary Permanent Temporary
----------- -------- -------- -------- ------------- --------- --------- --------- ---------
ACTP              14        0        0             0        14         0         0         0
----------- -------- -------- -------- ------------- --------- --------- --------- ---------
Totals            14        0        0             0        14         0         0         0
----------- -------- -------- -------- ------------- --------- --------- --------- ---------
******************************** BOTTOM OF DATA ********************************************
```

Calling SERXMLRC From Panel Exits

If you add ISPF panel exits to ChangeMan ZMF that call XML Services through SERXMLRC, allocate DD name SERLOAD in your ChangeMan ZMF logon CLIST. Concatenate the same load libraries at SERLOAD as you do in the LIBDEF for ISPLLIB in the logon CLIST.

```
ALLOC DD (SERLOAD) DSN(          +
  'somnode.CMNZMF.CUSTOM.LOAD'   +
  'somnode.SERCOMC.CUSTOM.LOAD'  +
  'somnode.CMNZMF.LOAD'          +
  'somnode.SERCOMC.LOAD'         +
  ) SHR REU
```

SERXMLRC checks for a SERLOAD DD statement before it executes the standard search order of STEPLIB, JOBLIB, etc., which are allocated to your TSO/ISPF session and are difficult to alter dynamically.

Add an ALLOC statement for SERLOAD to other CLISTS and REXX execs that run in a user's TSO session and call SERXMLRC.

Note

DD name SERLOAD is used only by SERXMLRC. Keep the LIBDEF for ISPLLIB in your ChangeMan ZMF logon CLIST and any other procedures that execute SERXMLRC in an ISPF environment.

DD name SERLOAD remains allocated after you exit from the logon CLIST, which is similar to the current behavior of DD names SER#PARM, SERLIC, and SERPRINT in the logon CLIST.

Problem Analysis and Troubleshooting Tools

F

This appendix provides information about tools you can use to resolve errors you get when using XML Services. If you seek assistance from Serena Support with an XML Services issue, Support may ask you to use some of these tools to provide them with information to diagnose the problem.

Warn - XML Tag Name Warning

The Warn facility in XML Services displays SERNET messages in the SERPRINT data set that describe tag name errors in XML Services requests.

When XML Services processes a request, an unrecognized tag name is ignored, and processing continues with the next tag.

If the data for that tag is critical to the execution of the request, the request fails and an error message based on the effect of the missing data is displayed in the response. If the data is not critical to the execution of the request, the request is executed without the data, and no error message is displayed. However, the result might not be the desired result.

The Warn facility helps you detect syntax errors in an XML Services request that may effect the response.

Warn Tag Name Error Examples

This section shows two examples of how the Warn facility alerts you to syntax errors in an XML Services message. These examples were executed in XML Services prototyping tool XMLSERV.

Example 1: Failed Request

In this example, the <applName> tag is misspelled, and since application is required to find any approval entities, the request fails. The response tells you that the request failed, but it does not describe the tag name error. The tag name error is described in a SERPRINT message.

REQUEST

```
<?xml version="1.0"?>
<service name="APPROVER">
 <scope name="APL">
```

```
  <message name="LIST">
   <header>
    <subsys>5</subsys>
    <test> </test>
    <warn>Y</warn>
    <product>CMN</product>
   </header>
  <request>
    <approverListType>1</approverListType>
    <xxxxName>ACTP</applName>
    <approverEntity>OPS     </approverEntity>
   </request>
  </message>
 </scope>
</service>
```

RESULT

```
<?xml version="1.0"?>
<service name="APPROVER">
 <scope name="APL">
  <message name="LIST">
   <response>
    <statusMessage>CMN8490I - package name was not specified.</
statusMessage>
    <statusReturnCode>08</statusReturnCode>
    <statusReasonCode>8490</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

SERPRINT MESSAGES

```
SER8209I Logon accepted for user USER239; Local CCSID=00037
SER8414W Unrecognized tag in request for user USER239, tag: xxxxName,
service: APPROVER, scope: APL, message: LIST
SER2005I CMN  Detach user USER239: TCA=1718D000 ASID=00B1
```

Example 2: Successful Request With Error

In this example, tag <yyyyyyyyEntity> is invalid and ignored, but since there is only one unplanned approver for this application, the result looks valid. However, the message in SERPRINT reveals the tag name error.

REQUEST

```
<?xml version="1.0"?>
<service name="APPROVER">
 <scope name="APL">
  <message name="LIST">
   <header>
    <subsys>5</subsys>
    <test> </test>
    <warn>Y</warn>
    <product>CMN</product>
   </header>
  <request>
    <approverListType>1</approverListType>
    <applName>ACTP</applName>
    <yyyyyyyyEntity>OPS     </approverEntity>
   </request>
  </message>
 </scope>
</service>
```

RESULT

```
<?xml version="1.0"?>
<service name="APPROVER">
 <scope name="APL">
  <message name="LIST">
   <result>
    <approverListType>1</approverListType>
    <applName>ACTP</applName>
    <approverEntity>OPS</approverEntity>
    <approverDesc>OPERATIONS SUPERVISOR</approverDesc>
    <approvalOrder>00</approvalOrder>
    <notification>
     <notifierType>1</notifierType>
     <userList>USER239</userList>
    </notification>
   </result>
   <response>
    <statusMessage>CMN8700I - LIST     Approver service completed</
statusMessage>
    <statusReturnCode>00</statusReturnCode>
    <statusReasonCode>8700</statusReasonCode>
   </response>
  </message>
 </scope>
</service>
```

SERPRINT MESSAGES

```
SER8209I Logon accepted for user USER239; Local CCSID=00037
SER8414W Unrecognized tag in request for user USER239, tag: yyyyyyyyEntity,
service: APPROVER, scope: APL, message: LIST
SER2005I CMN  Detach user USER239: TCA=1718D000 ASID=00B1
```

Enabling XML Tag Name Error Warning

There are three methods for turning on the Warn facility of XML Services:

- `<warn>` tag in an XML Services message header
- `WARN` keyword option for SERNET
- `WARN` modify command

<warn> Tag in an XML Services Message Header

When you set the `<warn>` tag in a request to Y, the Warn facility is active for that request when it is executed. The Warn facility is not invoked for other requests unless they are also coded with the `<warn>` tag set to Y.

Example:

```
<warn>Y</warn>
```

Setting the tag to `<warn>N</warn>` has no effect.

WARN Keyword Option for SERNET

The Warn facility can be turned on for all XML Services messages executed on a server by including the SERNET keyword option `WARN` in the parameters input to the started task when it is initiated.

Examples:

```
WARN
WARN(YES)
```

The `WARN` keyword option is described in the ChangeMan ZMF Installation Guide in the appendix titled "SERNET Keyword Options." Methods to input SERNET keyword options to a SERNET started task are described in topic "Passing Parameters to SERNET" in the same book.

WARN Modify Command

The Warn facility can be toggled on and off for all XML Services messages executed on a server by issuing the `WARN` modify command through the operator console.

Examples:

Command: `/F SERT5,WARN,YES`

SERPRINT Messages:

```
SER0850I Operator command: WARN,YES
SER0960I XML syntax warning has been turned on
```

Command: `/F SERT5,WARN,NO`

SERPRINT Messages:

```
SER0850I Operator command: WARN,NO
SER0959I XML syntax warning has been turned off
```

Hierarchy of Warn Facility Controls

The hierarchy of Warn facility controls is:

1. `<warn>Y</warn>` in an XML Services request
2. WARN modify command
3. WARN keyword option.

The `WARN` keyword option turns the Warn facility on for all XML Service requests.

The `WARN` modify commands toggles the Warn facility on or off, overriding the WARN keyword option.

`<warn>Y</warn>` in an XML Services request turns on the Warn facility for that request only, regardless of the status of the `WARN` keyword option or `WARN` modify command.

`<warn>N</warn>` in an XML Services request has no effect.

TEST - XML Batch Client Trace

The `<test>` tag in the header of an XML Services request turns on SERCLIEN tracing in clients SERXMLAC, SERXMLBC, SERXMLCC and SERXMLRC.

A trace on the client side of the communication between the client and the SERNET started task is output to the SERPRINT DD statement in the client job. This trace is similar to the communication trace in the SERNET started task generated by the NETTRACE modify command.

Values:

`<test>T</test>` turns on the SERCLIEN trace for this request

`<test> </test>` leaves the SERCLIEN trace off for this request

The `<test>` tag is generally used only at the request of Serena Support for diagnostic purposes. This client side trace is unaffected by the NETTRACE or TRACE settings in the SERNET started task.

TRACE and NETTRACE in the SERNET Started Task

The TRACE facility in the SERNET started task provides functional traces for SERNET and for the ChangeMan ZMF application running under SERNET. This facility is controlled by the TRACE keyword option input to SERNET at startup, and by the TRACE modify command, which dynamically overrides the keyword option setting.

Serena Support may ask you to turn on the TRACE facility to help diagnose problems you have with XML Services.

The NETTRACE keyword option for controls the display of SERNET communication buffer contents. Serena Support may ask you to turn on the NETTRACE facility to diagnose communication issues between XML Services clients and the server.

Details about SERNET keyword options and modify commands are in the Appendices of the *ChangeMan ZMF Installation Guide*.

Troubleshooting Tips

rc=08, reason code = 8130 error

The '8130' (SEND function failed - connection lost) indicates that the client job did not get a response from the server started task. This can occur either because of a communication setup problem or because of an abend in the ChangeMan ZMF started task. The following are some things you should check to make sure your XML service is communicating with the ChangeMan ZMF started task:

1. Is there an abend in the ChangeMan ZMF started task? If so, report the abend to SERENA Technical Support.
2. If the ChangeMan ZMF started task did not abend, does the '8130' error occur for all XML requests coded in the job? If so, check the following:
    a) Verify ChangeMan ZMF started task IP address and port# are correct.
    b) Verify the ChangeMan ZMF started task JCL has the parm 'XML=YES'.
    c) Verify XML job SER#PARM DSN references the same IP address and port# used by the ChangeMan ZMF started task.
    d) Verify client LPAR has IP connectivity to ChangeMan ZMF started task LPAR.
    e) Verify the XMLSPACE DD is in the ChangeMan ZMF started task JCL.
    f) Verify the correct version of MAPDATA was loaded into the XMLSPACE VSAM file for the version of ChangeMan ZMF being run.

Troubleshooting Variable Length Name Issues

If ChangeMan ZMF asserts in an error message that a known component or library type is not found, this may be due to a bug in variable length name processing by the XML service. As a workaround, try filling the `<component>` or **<libtype>** tag with trailing blanks so that the length of the marked data is the maximum allowed.

INDEX