Web Services Integration Toolkit for OpenVMS
API Reference Manual

Abstract

This manual provides information about the routines and API calls used in Web Services Integration Toolkit for OpenVMS.

Published: July 2012

© Copyright 2012 Hewlett-Packard Development Company, L.P.
Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

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

Intel and Itanium are trademarks of Intel Corporation in the U.S. and other countries.

Microsoft, Windows, Windows XP, Visual Basic, Visual C++, and Win32 are trademarks of Microsoft Corporation in the U.S. and/or other countries.

Java and all Java-based marks are trademarks or registered trademarks of Oracle and/or its affiliates in the U.S. and/or other countries.

Contents

1 About this document....4
Intended audience....4
Conventions....4
2 Web Services Integration Toolkit....5
WSI$VMS_LOGIN....6
WSI$VMS_LOGOUT....7
WSI$INVOKE....8
WSI$INIT....10
WSI$EXIT....11
WSI$START_SESSION....12
WSI$END_SESSION....13
WSI$ACMS_SIGN_IN....14
WSI$ACMS_SIGN_OUT....15
WSI$VMS_LOGIN....16
WSI$VMS_LOGOUT....17
WSI$INVOKE_DCL....18
WSI$GET_FILE....20
3 Support and other resources....22
HP encourages your comments....22
Related information....22
Index....23

1 About this document

This manual provides information about the routines and API calls used in Web Services Integration Toolkit (WSIT) for OpenVMS.

Intended audience

This manual is intended for developers and application programmers who want to use the WSIT services for OpenVMS.

Conventions

The following conventions are used in this manual:

bold type	Bold type represents the introduction of a new term. It also represents the name of an argument, an attribute, or a reason.
UPPERCASE TYPE	Uppercase type indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
( )	In command format descriptions, parentheses indicate that you must enclose choices in parentheses.

2 Web Services Integration Toolkit

The Web Service Integration Toolkit (WSIT) for OpenVMS contains a collection of integration tools. These tools are easy to use, highly extensible, and are based on open source standards and built on open source technology. The toolkit can be used to call OpenVMS applications written in third generation languages, such as C, BASIC, COBOL, FORTRAN, and Application Control and Management System (ACMS) languages such as Java, Microsoft .NET, Java -RMI, JMS, and web services.

The WSIT is focused on integrating at the application interface (API) level. It generates a JavaBean wrapper for a supplied OpenVMS API. At runtime, you can specify whether the application must run in the process of the caller (in-process) or in a separate process (out-of-process) managed by the WSIT runtime.

WSI$VMS_LOGIN

The WSI$VMS_LOGIN routine enables the client to log into the system using the specified user name and password.

Format

```
unsigned int WSI$VMS_LOGIN sessionID, UserName, Password
```

C Prototype

```
unsigned int WSI$VMS_LOGIN (unsigned int sessionID, char *pUserName, char
*pPassword)
```

Description

The routine WSI$VMS_LOGIN accepts the sessionID, UserName, and Password as arguments and passes these to the pVmsLogin routine. The pVmsLogin routine performs the required login action. If the login UserName, Password, or sessionID values are invalid, then the LIB$SIGNAL routine is started.

The Login Persona and the Login Home directory are reset and the error_status_ok message is returned. If the login attempt is successful, only the error_status_ok message is returned.

Arguments

sessionID

OpenVMS usage:	sessionID
Type:	Unsigned integer
Access:	Read only
Mechanism:	By value

It is a 32-bit value corresponding to the sessionID in which the login action is performed.

UserName

OpenVMS usage:	*pUserName
Type:	Character — coded text string
Access:	Read only
Mechanism:	By reference

It is a 8-bit value corresponding to the UserName in which the login action is performed.

Password

OpenVMS usage:	*pPassword
Type:	Character — coded text string
Access:	Read only
Mechanism:	By reference

It is a 8-bit value corresponding to the Password in which the login action is performed.

WSI$VMS_LOGOUT

The WSI$VMS_LOGOUT routine enables the client to log out of the system.

Format

```
unsigned int WSI$VMS_LOGOUT sessionID
```

C Prototype

```
unsigned int WSI$VMS_LOGOUT (unsigned int sessionID)
```

Description

The routine WSI$VMS_LOGOUT accepts the sessionID as an argument to logout a client from the system. The routine resets the Login Persona and the Login Home directory corresponding to the sessionID and assigns the value 0 and NULL to the Login Persona and the Login Home directory.

Arguments

sessionID

OpenVMS usage:	sessionID
Type:	Unsigned integer
Access:	Read only
Mechanism:	By value

It is a 32-bit value corresponding to the sessionID in which the logout action is performed.

WSI$INVOKE

The WSI$INVOKE routine is the main dispatch method is WSIT.

Format

```
unsigned int WSI$INVOKE sessionID, MethodID, inLen, *pbInData, *pOutLen,
*ppbOutData
```

C Prototype

```
unsigned int WSI$INVOKE (unsigned int sessionID, int MethodID, intinLen,
MSGB *pbInData, int*pOutLen, MSGB **ppbOutData)
```

Description

The WSI$INVOKE routine checks for a valid login corresponding to the sessionID's persona. If the sessionID value is invalid, then LIB$SIGNAL(SS$_INVLOGIN) is returned. Depending on the MethodID passed as a parameter, the corresponding action is performed.

The following list of methods can be performed:

```
cmlLogin
cmlLogout
getFirstAppRoot
getNextAppRoot
findAppRoot
getFirstApplication
getNextApplication
findApplication
addApplication
removeApplication
getApplicationProperties
getApplicationConfiguration
setApplicationConfiguration
getFirstInstance
getNextInstance
findInstance
addInstance
removeInstance
getInstanceProperties
getInstanceConfiguration
restartApplication
restartInstance
clearCache
reloadCache
getTimeStamp
isTimeStampOld
setCmlAttributes
setEventLogTrimAttr
initEventList
getEventList
closeEventList
shutdownManager
startupManager
getManagerLogfile
getInstanceLogfile
```

Arguments

sessionID

OpenVMS usage:	sessionID
Type:	Unsigned integer
Access:	Read only
Mechanism:	By value

It is a 32-bit value corresponding to the sessionID on which the action is performed.

MethodID

OpenVMS usage: `MethodID`
Type: Integer
Access: Read only
Mechanism: By value

It is a 32-bit value used a parameter by the routines to start the corresponding `MethodID` .

inLen

OpenVMS usage: `inLen`
Type: Unsigned integer
Access: Read only
Mechanism: By value

It is a 32-bit value used as a parameter by the routines to call the corresponding `MethodID`.

pbInData

OpenVMS usage: `*pbInData`
Type: Pointer to character
Access: Read only
Mechanism: By reference

It is a 8-bit value used as a parameter by the routines to call the corresponding `MethodID`.

pOutLen

OpenVMS usage: `*pOutLen`
Type: Integer
Access: Read only
Mechanism: By reference

It is a 32-bit value used as a parameter by the routines to call the corresponding `MethodID`.

ppbOutData

OpenVMS usage: `**ppbOutData`
Type: Pointer to character
Access: Read only
Mechanism: By reference

It is a 8-bit value used as a parameter by the routines to call the corresponding `MethodID`.

WSI$INIT

The `WSI$INIT` routine initializes the startup entry point for the application.

Format

```
unsigned int WSI$INIT *pAppBlock
```

C Prototype

```
unsigned int WSI$INIT(Application Context *pAppBlock)
```

Description

The `WSI$INIT` routine takes the `Application Context` as an input and initializes mailbox, assigns a channel to the mailbox for delivery of messages, creates the lock for the application, passes the `Init` call to the `User's shareable` and initializes the application.

Arguments

`Application Context`

OpenVMS usage:	`*pAppBlock`
Type:	`Application Context`
Access:	Read only
Mechanism:	By reference

It is a structure value corresponding to the context of the application that is initiated by the WSIT server.

WSI$EXIT

The WSI$EXIT routine shutdowns the entry point of the application.

Format

```
unsigned int WSI$EXIT
```

C Prototype

```
unsigned int WSI$EXIT()
```

Description

The WSI$EXIT routine takes the Application Context as an input and exits the application by closing all the processes that are initialized by the WSI$INIT routine.

Arguments

There are no arguments for the WSI$EXIT routine.

WSI$START_SESSION

The WSI$START_SESSION routine acts as the session startup entry point.

Format

```
unsigned int WSI$START_SESSION **pSessionID *pMemDispatch
```

C Prototype

```
unsigned int WSI$START_SESSION (internal_context_t **pSessionID, wsi$disp_t
*pMemDispatch)
```

Description

The WSI$START_SESSION routine takes the Application Context and Dispatch Pointer as inputs and returns a status code to the caller. The routine passes the Start Session call to the User's shareable. For the Start Session call, the WSI$START_SESSION routine calls the Application Dispatch function, which includes the Session Management, Invocation, Inquiry, Memory Management, and Transaction Management routines. After this, the WSI$START_SESSION routine calls the Application Interface routine. These are the primary call points and all the wrapped routines (ACMS Tasks, DCL Procedures, and Wrapped Files) are called through one of these entry points.

The routines are identified by a generated MethodID. In MethodID, the input parameters are passed as an encoded stream, and an encoded output stream is produced that is decoded at the far end.

The Manager application is notified that a client connection is started. To notify the Manager application, a Send Connect Message routine is called where an I/O is queued in the Manager's mailbox.

Arguments

pSessionID

OpenVMS usage:	**pSessionID
Type:	MonitorContext
Access:	Read only
Mechanism:	By reference

It is a structure value corresponding to the context of the application. The WSI$START_SESSION routine gets the application information such as if the application is thread safe, the application name, a pointer to the dispatch table, and if a transport is being used in the current deployment while using this parameter.

pMemDispatch

OpenVMS usage:	*pMemDispatch
Type:	wsi$disp_t
Access:	Read only
Mechanism:	By reference

It is a structure value, that contains the Dispatch Pointers for the Helper function used by the User Server Interface routine. The specific Helper functions are established by the WSIT server or middle infrastructure, and then passed at the start of a session.

WSI$END_SESSION

The WSI$END_SESSION routine closes the session.

Format

```
unsigned int WSI$END_SESSION sessionID
```

C Prototype

```
unsigned int WSI$END_SESSION (unsigned int sessionID)
```

Description

The WSI$END_SESSION routine terminates a client connection depending on the sessionID passed as a parameter. The routine fetches the Monitor Context of the application whose sessionID is provided. The routine passes the End Session call to the User's shareable. For this, the WSI$END_SESSION routine calls the Application Dispatch function, which includes the Session Management, Invocation, Inquiry, Memory Management, and Transaction Management routines. After this, the WSI$END_SESSION routine calls the Application Interface routine. These are the primary call points and all the wrapped routines (ACMS Tasks, DCL Procedures, and Wrapped Files) are called through one of these entry points.

The routines are identified by a generated MethodID. In MethodID, the input parameters are passed as an encoded stream, and an encoded output stream is produced that is decoded at the far end. Then it terminates the session.

The Manager application is notified that a client connection is stopped. To notify the Manager application, a Send Disconnect Message routine is called where an I/O is queued in the Manager's mailbox. The WSI$DESTROY_SESSION_CONTEXT routine is passed as the sessionID parameter, which frees the Application Context.

Arguments

sessionID

OpenVMS usage:	sessionID
Type:	Unsigned integer
Access:	Read only
Mechanism:	By value

It is a 32-bit integer value that is used to identify the Application Context.

WSI$ACMS_SIGN_IN

The WSI$ACMS_SIGN_IN routine logs the client into the ACMS system, using the specified user name.

Format

```
unsigned int WSI$ACMS_SIGN_IN sessionID *pUserName
```

C Prototype

```
unsigned int WSI$ACMS_SIGN_IN (unsigned int sessionID, char *pUserName)
```

Description

The WSI$ACMS_SIGN_IN routine takes the sessionID and UserName as input and returns a status code to the caller. The routine passes the Sign In Session call to the User's shareable. For this, the WSI$ACMS_SIGN_IN routine calls the Application Dispatch function, which includes the Session Management, Invocation, Inquiry, Memory Management, and Transaction Management routines. After this, the WSI$ACMS_SIGN_IN routine calls the Application Interface routine. These are the primary call points and all the wrapped routines (ACMS Tasks, DCL Procedures, and Wrapped Files) are called through one of these entry points.

The routines are identified by a generated MethodID. In MethodID, the input parameters are passed as an encoded stream, and an encoded output stream is produced that is decoded at the far end.

Depending on the user name provided, the Sign In Session is started.

Arguments

sessionID

OpenVMS usage:	sessionID
Type:	Unsigned integer
Access:	Read only
Mechanism:	By value

It is a 32-bit integer value that is used to identify the Application Context.

pUserName

OpenVMS usage:	*pUserName
Type:	Character — coded text string
Access:	Read only
Mechanism:	By reference

It is a pointer to a character array that consists of the user name for the Sign In Session.

WSI$ACMS_SIGN_OUT

The WSI$ACMS_SIGN_OUT routine logs the client out of the ACMS system.

Format

```
unsigned int WSI$ACMS_SIGN_OUT sessionID
```

C Prototype

```
unsigned int WSI$ACMS_SIGN_OUT (unsigned int sessionID)
```

Description

The WSI$ACMS_SIGN_OUT routine takes the sessionID as an input and returns a status code to the caller. The routine calls the Application Dispatch function, which includes the Session Management, Invocation, Inquiry, Memory Management, and Transaction Management routines. After this, the WSI$ACMS_SIGN_OUT routine calls the Application Interface routine. These are the primary call points and all the wrapped routines (ACMS Tasks, DCL Procedures, and Wrapped Files) are called through one of these entry points.

The routines are identified by a generated MethodID. In MethodID, the input parameters are passed as an encoded stream, and an encoded output stream is produced that is decoded at the far end.

Depending on the sessionID provided, the application session is terminated.

Arguments

sessionID

OpenVMS usage:	sessionID
Type:	Unsigned integer
Access:	Read only
Mechanism:	By value

It is a 32-bit integer value that is used to identify the Application Context.

WSI$VMS_LOGIN

The WSI$VMS_LOGIN routine logs the client into the system using the specified user name and password.

Format

```
unsigned int WSI$VMS_LOGIN sessionID *pUserName *pPassword
```

C Prototype

```
unsigned int WSI$VMS_LOGIN (unsigned int sessionID, char*pUserName,
char*pPassword)
```

Description

The WSI$VMS_LOGIN routine takes the sessionID, UserName, and Password as inputs and returns a status code to the caller. The routine passes the VMS Login Session call to the User's shareable. The routine calls the Application Dispatch function, which includes the Session Management, Invocation, Inquiry, Memory Management, and Transaction Management routines. After this, the WSI$VMS_LOGIN routine calls the Application Interface routine. These are the primary call points and all the wrapped routines (ACMS Tasks, DCL Procedures, and Wrapped Files) are called through one of these entry points.

The routines are identified by a generated MethodID. In MethodID, the input parameters are passed as an encoded stream, and an encoded output stream is produced that is decoded at the far end.

Depending on the UserName and the Password provided, the session is started and a status code is returned to the caller.

Arguments

sessionID

OpenVMS usage:	sessionID
Type:	Unsigned integer
Access:	Read only
Mechanism:	By value

It is a 32-bit integer value that is used to identify the Application Context.

pUserName

OpenVMS usage:	*pUserName
Type:	Character — coded text string
Access:	Read only
Mechanism:	By reference

It is a pointer to a character array that consists of the user name for the Sign In Session.

pPassword

OpenVMS usage:	*pPassword
Type:	Character — coded text string
Access:	Read only
Mechanism:	By reference

It is a pointer to a character array that consists the password for the Sign In Session.

WSI$VMS_LOGOUT

The WSI$VMS_LOGOUT routine logs the user out of the system.

Format

```
unsigned int WSI$VMS_LOGOUT sessionID
```

C Prototype

```
unsigned int WSI$VMS_LOGOUT (unsigned int sessionID)
```

Description

The WSI$VMS_LOGOUT routine takes the sessionID as an input and returns a status code to the caller. The routine passes the VMS Logout Session call to the User's shareable. The routine calls the Application Dispatch function, which includes the Session Management, Invocation, Inquiry, Memory Management, and Transaction Management routines. After this, the WSI$VMS_LOGOUT routine calls the Application Interface routine. These are the primary call points and all the wrapped routines (ACMS Tasks, DCL Procedures, and Wrapped Files) are called through one of these entry points.

The routines are identified by a generated MethodID. In MethodID, the input parameters are passed as an encoded stream, and an encoded output stream is produced that is decoded at the far end.

Depending on the sessionID provided, the VMS Session is stopped.

Arguments

sessionID

OpenVMS usage:	sessionID
Type:	Unsigned integer
Access:	Read only
Mechanism:	By value

It is a 32-bit integer value that is used to identify the Application Context.

WSI$INVOKE_DCL

The WSI$INVOKE_DCL routine is the dispatch method for DCL procedures.

Format

```
unsigned int WSI$INVOKE_DCL sessionID *ctx *inBuf inLen *outLen **outBuf
```

C Prototype

```
unsigned int WSI$INVOKE_DCL (unsigned int sessionID, wsi$disp_t *ctx,
unsigned char *inBuf, int inLen, int *outLen, unsigned char **outBuf)
```

Description

The WSI$INVOKE_DCL routine takes the sessionID, Dispatch Pointer, Input Buffer, Input Length, Output Length, and Output Buffer for the DCL session as inputs and returns a status code to the caller. The routine calls the Application Dispatch function, which includes the Session Management, Invocation, Inquiry, Memory Management, and Transaction Management routines. After this, the WSI$INVOKE_DCL routine calls the Application Interface routine. These are the primary call points and all the wrapped routines (ACMS Tasks, DCL Procedures, and Wrapped Files) are called through one of these entry points.

The routines are identified by a generated MethodID. In MethodID, the input parameters are passed as an encoded stream, and an encoded output stream is produced that is decoded at the far end.

The sessionID is used to fetch the Application Context. The Application Context along with the Dispatch Pointer, Input Buffer, Input Length, Output Length, and Output Buffer are passed to the RUNDCL routine to start the DCL session.

Arguments

sessionID

OpenVMS usage:	sessionID
Type:	Unsigned integer
Access:	Read only
Mechanism:	By value

It is a 32-bit integer value that is used to identify the Application Context.

ctx

OpenVMS usage:	*ctx
Type:	wsi$disp_t
Access:	Read only
Mechanism:	By reference

It is a structure value that contains the Dispatch Pointer for the Helper function used by the User Server Interface routines. The specific Helper functions are established by the WSIT server or middle infrastructure, and then passed at the start of a session.

inBuf

OpenVMS usage:	*inBuf
Type:	Unsigned character
Access:	Read only
Mechanism:	By reference

It is an unsigned character value that contains the information about the Input Buffer, which is passed to the RUNDCL routine.

outBuf

OpenVMS usage: `*outBuf`

Type: Unsigned character

Access: Read only

Mechanism: By reference

It is an unsigned character value that contains the information about the `Output Buffer`, which is passed to the `RUNDCL` routine.

inLen

OpenVMS usage: `inLen`

Type: Integer

Access: Read only

Mechanism: By value

It is an integer value that is used to determine the `Input Length` for the DCL session.

outLen

OpenVMS usage: `*outLen`

Type: Integer

Access: Read only

Mechanism: By reference

It is an integer value that is used to determine the `Output Length` for the DCL session.

WSI$GET_FILE

The WSI$GET_FILE routine wraps up the text file along with the dispatch method.

Format

```
unsigned int WSI$GET_FILE sessionID fileID *ctx *outLen **outBuf
```

C Prototype

```
unsigned int WSI$GET_FILE (unsigned int sessionID, intfileID, wsi$disp_t
*ctx, int *outLen, unsigned char **outBuf)
```

Description

The WSI$GET_FILE routine takes the sessionID, File ID, Dispatch Pointer, Output Length, and Output Buffer for dispatching a file and returns a status code to the caller. The routine passes the Start Session call to the User's shareable. The routine calls the Application Dispatch function, which includes the Session Management, Invocation, Inquiry, Memory Management, and Transaction Management routines. After this, the WSI$GET_FILE routine calls the Application Interface routine. These are the primary call points and all the wrapped routines (ACMS Tasks, DCL Procedures, and Wrapped Files) are called through one of these entry points.

The routines are identified by a generated MethodID. In MethodID, the input parameters are passed as an encoded stream, and an encoded output stream is produced that is decoded at the far end.

The sessionID is used to fetch the Application Context. The Application Context along with the Dispatch Pointer, Output Length, and Output Buffer are passed to the GetFile routine to dispatch the text file.

Arguments

sessionID

OpenVMS usage:	sessionID
Type:	Unsigned integer
Access:	Read only
Mechanism:	By value

It is a 32-bit integer value that is used to identify the Application Context.

ctx

OpenVMS usage:	*ctx
Type:	wsi$disp_t
Access:	Read only
Mechanism:	By reference

It is a structure value that contains the Dispatch Pointers for the Helper functions used by the User Server Interface routines. The specific Helper functions are established by the WSIT server or middle infrastructure, and then passed at the start of a session.

outBuf

OpenVMS usage:	*outBuf
Type:	Unsigned character
Access:	Read only
Mechanism:	By reference

It is an unsigned character value that contains the information about the Output Buffer, which is passed to the GetFile routine.

outLen

OpenVMS usage: `*outLen`

Type: Integer

Access: Read only

Mechanism: By reference

It is an integer value that is used to determine the `Output Length` for dispatching the text file.

3 Support and other resources

HP encourages your comments

HP welcomes your comments and suggestions on the quality and usefulness of this publication.

Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?

If you find any errors or have any other suggestions for improvement, please indicate the title of the documentation and the chapter, section, and page number (if available). You can send comments to us to:

openvmsdoc@hp.com

Related information

- *Web Secure Integration Toolkit for OpenVMS Installation Guide and Release Notes*: http://h71000.www7.hp.com/openvms/products/ips/wsit/wsit_doc.html
- *Web Secure Integration Toolkit for OpenVMS Developer's Guide*: http://h71000.www7.hp.com/openvms/products/ips/wsit/wsit_doc.html

Index

A

Application Context, 11
audience
  application programmers, 4
  developers, 4

C

Conventions
  ( ), 4
  bold type, 4
  UPPERCASE TYPE, 4

M

Manager, 12

R

routine
  Inquiry, 12
  Invocation, 12
  Memory Management, 12
  Session Management, 12
  Transaction Management, 12

W

WSI$ACMS_SIGN_IN, 14
WSI$ACMS_SIGN_OUT, 15
WSI$END_SESSION, 13
WSI$EXIT
  Application Context, 11
WSI$GET_FILE, 20
WSI$INIT
  Application Context, 10
WSI$INVOKE
  dispatch method, 8
  inLen, 9
  MethodID, 9
  pbInData, 9
  pOutLen, 9
  ppbOutData, 9
  sessionID, 8
WSI$INVOKE_DCL, 18
WSI$START_SESSION
  Application Context, 12
  Application Dispatch, 12
WSI$VMS_LOGIN, 16
  Password, 6
  pVmsLogin, 6
  sessionID, 6
  UserName, 6
WSI$VMS_LOGOUT, 17
  sessionID, 7