Developing xml interfaces for COBOL applications

Contents

Chapter 1. Introduction to XML Services for the Enterprise

XML Services for the Enterprise tools let you easily adapt COBOL-based business applications so that they become web services and can process and produce XML messages. This new kind of interface to a called application allows (for example) an Internet user to access an existing CICS® or IMS™ application. The tools also can help you to embed a COBOL application in a larger system that uses XML for data interchange.

The XML Services for the Enterprise tools consist of:

- The Web Services Enablement wizard that allows to generate a new web service interface. Typically, this is called ″bottom-up″ approach since the existing COBOL application is at the ″bottom″ of the new Web services creation process.
- The XML to COBOL mapping tools that allow to map an existing web service interface or an XML data definition to the existing COBOL program. Typically, this is called ″meet-in-the-middle″ approach, since the exiting Web services definition ″meets″ or ″maps″ to the existing COBOL interface.
- The Batch processor that allows to run generate the web service interface in unattended (″batch″) mode. The Batch processor currently supports the ″bottom-up″ Web services creation. The functionality of the Batch processor is equivalent to those of the Web Services Enablement wizard described above.

In most cases, you can leave the existing business application as is, so that other existing programs that supply COBOL data (rather than XML) can access the COBOL program as before. Restrictions apply, however, as described later.

Related concepts

XML to COBOL mapping tools
Batch Processor

Related tasks

Creating a web service interface with the Web Services Enablement wizard
Setting preferences for XML Services for the Enterprise
Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file
Starting the batch processor

Related reference

Mapping reference

Introduction to XML Services for the Enterprise

XML Services for the Enterprise tools let you easily adapt COBOL-based business applications so that they become web services and can process and produce XML messages. This new kind of interface to a called application allows (for example) an Internet user to access an existing CICS or IMS application. The tools also can help you to embed a COBOL application in a larger system that uses XML for data interchange.

The XML Services for the Enterprise tools consist of:

- The Web Services Enablement wizard that allows to generate a new web service interface. Typically, this is called ″bottom-up″ approach since the existing COBOL application is at the ″bottom″ of the new Web services creation process.
- The XML to COBOL mapping tools that allow to map an existing web service interface or an XML data definition to the existing COBOL program. Typically, this is called ″meet-in-the-middle″ approach, since the exiting Web services definition ″meets″ or ″maps″ to the existing COBOL interface.
- The Batch processor that allows to run generate the web service interface in unattended (″batch″) mode. The Batch processor currently supports the ″bottom-up″ Web services creation. The functionality of the Batch processor is equivalent to those of the Web Services Enablement wizard described above.

In most cases, you can leave the existing business application as is, so that other existing programs that supply COBOL data (rather than XML) can access the COBOL program as before. Restrictions apply, however, as described later.

Related concepts

XML to COBOL mapping tools
Batch Processor

Related tasks

Creating a web service interface with the Web Services Enablement wizard
Setting preferences for XML Services for the Enterprise
Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file
Starting the batch processor

Related reference

Mapping reference

Restrictions and limitations

General restrictions:

- Each XML schema definition must be based on a single data declaration. (The definition for input and output may be the same.)
- The data declarations must not include objects, pointers, or procedure pointers.
- If a called COBOL program receives data by way of a user interface (by displaying a CICS map, for example), you must take the user interaction out of that program; the called COBOL must be invoked programmatically rather by a user interface.
- If you wish a COBOL program to send an XML message, you need to change the logic of that program:
  - The caller must invoke an output converter to create an XML message from the COBOL data that is required by the called program
  - The caller then includes the resulting XML string (not the COBOL data) in the call to the called program

For **inbound message processing** the following statements are true:

- User defined entity references in an internal or external DTD are not supported.
- For DBCS (double-byte character set), only XML documents encoded in Unicode (UTF-16 or UTF-8) are supported.

- The entire XML message must be scanned. No way exists to trigger end of parsing when, for example, input is acquired or specific element is seen.
- XML element attributes are ignored.
- Error handling is provided by LE exceptions with following limitations:
  - No error counts are provided.
  - The diagnostic level is ″flag(e).″ (Exceptions are thrown by the converter for a predefined set of errors: LE exceptions, conversion exceptions thrown by functions NUMVAL(C), and conversion errors defined by the converter

For **outbound message generation** the following statements are true:

- In character content only trailing blanks are removed. In numeric content leading and trailing blanks are removed, and leading zeros are removed except the last significant zero before the decimal point (or decimal comma).
- Creating XML element attributes is not supported.
- XML declaration is no longer included in the generated messages. This makes it possible to include the XML in the SOAP message body.
- Error handling is provided by LE exceptions with following limitations:
  - No error counts are provided.
  - The diagnostic level is ″flag(e).″ (Exceptions are thrown by the converter for a predefined set of errors: LE exceptions and conversion errors defined by the converter.)

The following *data structure limitations* exist:

- PICTURE N USAGE NATIONAL (UTF-16) data types are allowed with Unicode input or output.
- Pictures N and G usage DISPLAY-1 (DBCS) are allowed with Unicode input or output.
- If the minimum number of occurrences of an ODO group is 0 (occurs 0 to n times), the XML Schema generated by the XML Enablement tool will erroneously have a minOccurs attribute of 1.
- COBOL data types including POINTER, COMP-X, INDEX, and PROCEDURE-POINTER are not supported. However, they can reside elsewhere in the COBOL file.
- COBOL level 66 and level 77 records terminate the data structure of the CICS transaction. The COBOL importer does not consider these to be part of the data structure.
- Hexadecimal binary values cannot be attributed to nonnumeric literals. They cannot reside in the data structures that are imported by the COBOL importer. However, they can reside elsewhere in the COBOL file. Alternatively, you can convert the hexadecimal value to a char string for PIC X or to a decimal number for PIC 9.
- The figurative constants LOW-VALUES(S), HIGH-VALUES(S), and NULL are not supported. They cannot reside in the data structures to be imported by the COBOL importer, but they can reside elsewhere in the COBOL file.
- Object-oriented extensions to COBOL 85 are not supported. For example, OBJECT-REFERENCE is not supported. These extensions cannot reside in the data structures to be imported by the COBOL importer. They will prevent the COBOL file from being imported correctly. They can, however, reside elsewhere in the COBOL file.

Related concepts

XML to COBOL mapping tools

Related tasks

Creating a web service interface with the Web Services Enablement wizard
Setting preferences for XML Services for the Enterprise

Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Mapping reference

Chapter 2. Creating a web service interface with the Web Services Enablement wizard

The Web Services Enablement wizard lets you create a web services interface for an existing COBOL business function or application.

To use the tool, do as follows:

1. In the Navigator view, right-click on a COBOL file and select **Enable Web service->Generate enablement components...**
2. Work through the pages of the wizard:
   - "Data structures"
   - "COBOL Import Properties" on page 6
   - "Generation Options" on page 7
   - "IMS SOAP Gateway" on page 10
   - "Web Services in CICS" on page 9
   - "File, data set or member selection" on page 12

Related concepts

Introduction to XML Services for the Enterprise
XML to COBOL mapping tools

Related tasks

Setting preferences for XML Services for the Enterprise
Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Mapping reference

Data structures

The data structures have been imported from the language source. Select the inbound and outbound data structures.

Inbound data structure

Select the Inbound data structure tab to identify the data structure items that you want to expose as the content for the inbound XML message.

Outbound data structure

Select the Outbound data structure tab to identify the data structure items that you want to expose as the content for the outbound XML message.

Change COBOL Options

Select the Change COBOL Options button to modify the current COBOL importer option settings.

Related concepts

Introduction to XML Enablement for the Enterprise
XML to COBOL mapping tools

Related tasks

Creating a web service interface with the Web Services Enablement wizard
Setting preferences for XML Enablement for the Enterprise
Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Mapping reference
Web Services in CICS
COBOL Import Properties
Generation Options
IMS SOAP Gateway
File, data set or member selection

COBOL Import Properties

Use the COBOL Import Properties window to specify the platform properties. See the compiler options reference in the COBOL Programming guide for details on specific options.

Platform
Specifies the target generation platform. Select the appropriate value from the pull down list.

Code Page Selection
Specifies the code page for the locale. Select the appropriate value from the popup window. See Supported code pages (CCSIDs).

Floating point format
Specifies the floating point format. Select the appropriate value from the pull down list.

endian
Select the appropriate value from the available options.

Remote integer endian
Select the appropriate value from the available options.

External decimal sign
Select the appropriate value from the available options.

Specify the COBOL options

QUOTE
Select the appropriate value from the available options.

TRUNC
Select the appropriate value from the available options.

NSYMBOL
Select the appropriate value from the available options.

Compile time locale name
: Specifies the compile time locale. Select the appropriate value from the pull down list.

  Please see Locale and code pages for additional information.

ASCII code pages
: Specifies the ASCII code page. Select the appropriate value from the pull down list.

Error messages language
: Specifies the default language for the error messages. Select the appropriate value from the pull down list.

Currency sign
: Specifies the currency sign. Enter the appropriate value in the space provided.

SOSI Select the checkbox to enable SOSI.

COLLSEQ
: Select the appropriate value from the available options.

NCOLLSEQ
: Select the appropriate value from the available options.

File Extension Support
: Specify the file extension support. To change the value, select support for a specific file extension and then, using the pulldown arrow, select the new value from the pull down list.

To set the defaults for the values on the COBOL Import properties page, select **Window > Preferences**, then expand **Importer**, then select **COBOL**.

Related concepts

Introduction to XML Enablement for the Enterprise
XML to COBOL mapping tools
JCA documentation

Related tasks

Creating a web service interface with the Web Services Enablement wizard
Setting preferences for XML Enablement for the Enterprise
Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file
COBOL Importer

Related reference

Mapping reference
Web Services in CICS
Data structures
Generation Options
IMS SOAP Gateway
File, data set or member selection

Generation Options

At the Generation options page, specify characteristics of the generated files:

XML Converter Options

Converter type
Specify one of the following driver types to generate based on the environment you are targeting for deployment:

- Batch, TSO and USS
- IMS SOAP Gateway
- SOAP for CICS
- Web Services in CICS

Specify identification attributes

Program name
Specify the stem of the program name that is included in the IDENTIFICATION DIVISION of each generated COBOL program. If you type ACCT, for example, the wizard identifies the input converter program as ACCTI, the output converter program as ACCTO, and the driver as ACCTD

Author name
Specify the value to be included in the AUTHOR paragraph of each generated COBOL program.

Business program name
Specifies the existing business program that the XML converters call. This is the program that you are enabling for processing and/or producing XML messages (to act as a web service, for example.)

Specify character encodings

Inbound code page
Specify the code page for encoding the XML input message.

Host code page
Specify the code page for the z/OS® host system.

Outbound code page
Specify the code page for encoding the XML output message

WSDL and XSD Options

Specify the WSDL properties

Endpoint URI:
Specify the web service endpoint URI. This URI is used in the binding section of the WSDL file. In the case of the SOAP binding, this URI is generated into the content of the soap:address element.

Specify XML Schema properties

Inbound namespace
Specify the XML namespace you want the Inbound converter to use

Note: By default, the source program name and a character designator `I` for inbound is used. You can override the default by editing the text in the field.

Outbound namespace
Specify the XML namespace you want the Outbound converter to use

Note: By default, the source program name and a character designator `O` for outbound is used. You can override the default by editing the text in the field.

To set defaults for the Enable Web Services wizard, select **Window > Preferences**, then expand **XML Services for the Enterprise** and **Enable Web Service Wizard**, then select **COBOL Generator**.

Related concepts

Introduction to XML Enablement for the Enterprise
XML to COBOL mapping tools

Related tasks

Creating a web service interface with the Web Services Enablement wizard
Setting preferences for XML Enablement for the Enterprise
Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Mapping reference
Web Services in CICS
Data structures
COBOL Import Properties
IMS SOAP Gateway
File, data set or member selection

Web Services in CICS

WSBind Properties

Specify targets for the WSBind file

WSBind file folder
Specify the path for the folder containing the WSBind file. Use the Browse button to select an existing folder.

WSBind file name
Specify the name of the WSBind file. To specify a file name, enter the file name in the area provided.

Note: Do not add an extension to the file name. The extension .wsbind is automatically appended to the file name.

Overwrite WSBind file
Select this checkbox to allow the Enable Web Services wizard to overwrite the WSBind file if it exists.

Specify CICS application program properties

Program interface
Specify whether the CICS application program communicates via a DFHCOMMAREA or CONTAINER. Select the appropriate value from the pull down list.

Container name
If the CICS application program communicates via a CONTAINER, you must specify container name.

Advanced WSBind Properties

Specify web services properties

Local URI
Specify the desired local URI to for the web service, for example, ″/exampleApp/InquireSingle″. Note that this URI is different from the location of

the web service for example, http://server:port[local URI]. If you do not specify this property it will have to be defined at install time during manual creation of the web service resource definitions in CICS.

Pipeline name
Specify the name of the CICS PIPELINE resource under which this web service should be installed. If you do not specify this property it will have to be defined at install time during manual creation of the web service resource definitions in CICS.

WSDL HFS file path
Specify the full HFS path to the WSDL file that CICS should use for validation of SOAP request and response messages, for example, ″/u/svltest/pickup/inquireSingle.wsdl″. If you do not specify this property it can be defined at install time during manual creation of the web service resource definitions in CICS.

Related concepts

Introduction to XML Enablement for the Enterprise
XML to COBOL mapping tools

Related tasks

Creating an XML interface with XML Enablement for the Enterprise
Setting preferences for XML Enablement for the Enterprise
Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Mapping reference
Data structures
COBOL Import Properties
Generation Options
IMS SOAP Gateway
File, data set or member selection

IMS SOAP Gateway

When you select IMS, the IMS SOAP Gateway window prompts you for the following IMS SOAP Gateway Correlator properties.

Correlator properties

Specify the SOAP properties

SOAPAction
Specifies the Soap Action. The default is the data source file name (no extension) concatenated with ″urn″. For example, `urn:`*`source file name`*`.`

Specify targets for the correlator

Correlator file folder
Specify the Correlator file folder.

Correlator file name
Specify the Correlator file name.

Overwrite correlator file
: Select this checkbox to allow subsequent generations to overwrite the contents of a previously generated correlator file.

Specify integration properties

Socket timeout
: Specifies how long (in milliseconds) the SOAP Gateway waits for a response from IMS Connect.

Execution timeout
: Specify how long (in milliseconds) the IMS Connect waits for a response from IMS.

LTERM name
: Specify IMS specific properties that you provide. You can set the value of this property if the client application wants to provide an LTERM override name. This name is in the IMS application program's I/O PCB, with the intent that the IMS application makes logic decisions based on the override value.

Connection bundle name
: Specify the name of the connection bundle the web service uses to connect to IMS. Connection bundles are defined in the connection specification XML file maintained by the IMS SOAP Gateway and can be updated using the IMS SOAP Gateway Deployment tool

Adapter type
: Specify the adapter that is used for data transformation. You can choose the given value or enter an alphanumeric value up to 8 characters in length.

Related concepts

Introduction to XML Enablement for the Enterprise
XML to COBOL mapping tools

Related tasks

Creating a web service interface with the Web Services Enablement wizard
Setting preferences for XML Enablement for the Enterprise
Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Mapping reference
Web services in CICS
Data structures
COBOL Import Properties
Generation Options
File, data set or member selection

File, data set or member selection

At the File, data set, or member selection page, either accept the default values or specify file names and locations:

XML Converters

Select targets for the XML Converters and Converter Driver

Converter folder
Specifies the path for the folder containing the Converter. Use the Browse button to select an existing folder.

Converter driver file name
Specifies the name of the Converter driver file. To specify a file name, select the checkbox and enter the file name in the area provided.

Note: Do not add an extension to the file name. The extension .cbl is automatically appended to the file name.

Inbound Converter file name
Specifies the name of the Inbound Converter file. To specify a file name, select the checkbox and enter the file name in the area provided.

Note: Do not add an extension to the file name. The extension .cbl is automatically appended to the file name.

Outbound Converter file name
Specifies the name of the Outbound Converter file. To specify a file name, select the checkbox and enter the file name in the area provided.

Note: Do not add an extension to the file name. The extension .cbl is automatically appended to the file name.

Generate all to driver
Select this checkbox to generate all of the COBOL files to the driver.

Overwrite files without warning
Select this checkbox to allow the wizard to overwrite existing files without warning.

Select targets for the XML Schemas and Web Services Definition Language

Specify web services properties

WSDL folder
Specifies the path for the folder containing the WSDL file. Use the Browse button to select an existing folder.

WSDL file name
Specifies the name of the WSDL file. To specify a file name, select the checkbox and enter the file name in the area provided.

Note: Do not add an extension to the file name. The extension .wsdl is automatically appended to the file name.

Inbound XSD file name
Specifies the name of the Inbound XSD file. To specify a file name, select the checkbox and enter the file name in the area provided.

Note: Do not add an extension to the file name. The extension .xsd is automatically appended to the file name.

Outbound XSD file name
: Specifies the name of the Outbound XSD file. To specify a file name, select the checkbox and enter the file name in the area provided.

  Note: Do not add an extension to the file name. The extension .xsd is automatically appended to the file name.

Overwrite files without warning
: Select this checkbox to allow the Enable Web Services wizard to overwrite existing files without warning.

Related concepts

Introduction to XML Enablement for the Enterprise
XML to COBOL mapping tools

Related tasks

Creating a web service interface with the Web Services Enablement wizard
Setting preferences for XML Enablement for the Enterprise
Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Mapping reference
Web Services in CICS
Data structures
COBOL Import Properties
Generation Options
IMS SOAP Gateway

Chapter 3. XML to COBOL mapping tools

XML to COBOL Mapping tools allow you to enable existing COBOL applications to process and produce XML documents when the XML documents do not identically match the COBOL data items in terms of names or even data types. This situation occurs, for example, when the XML documents are derived from sources other than the target COBOL data structure.

After defining the mappings, you can generate converter and driver programs. The converters are generated in COBOL and their function is to transform the content of the mapped XML elements to the content of COBOL data items and the content of COBOL data items to the content of mapped XML elements. A sample COBOL converter driver program is also generated that illustrates how to invoke the converters in conjunction with the existing COBOL program.

These tools are very useful when, for example, one company acquires or merges with one or more other companies, and intends to merge and consolidate various parts of the enterprise information systems (EIS) resulting from the merger or acquisition. The interfaces between various enterprise applications most likely do not precisely match. Another use of the mapping tools is when an existing enterprise application is required to process an XML document described by a schema originating from a third party such as a standards committee.

The XML to COBOL mapping editor allows you to map one or more existing source XML documents to an existing target COBOL data structure and inversely, from a source COBOL data structure to an existing target XML document. Specifically, elements of the XML documents can be mapped visually to COBOL data items.

Related concepts

XML to COBOL mapping concepts

Related tasks

Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Mapping reference

XML to COBOL mapping concepts

Mapping of the XML elements to COBOL data items is achieved based on specific COBOL and XML *data models*. In particular:

- The COBOL data model for a given data structure is expressed as an instance of the COBOL Common Application Metamodel (CAM)
- The XML data model for a given data structure is expressed as an instance of the XML Schema model (whether it is an XML document, WSDL types definition, XML Schema or its DTD representation.)

Therefore, you can perform mappings on the following types of files:

- WSDL documents (extension must be .wsdl)
- XML instance documents (extension must be .xml)

- XML schema (XSD) documents (extension must be .xsd)
- XML document type declaration (DTD) files (extension must be .dtd)
- COBOL source files (extensions must be .cbl, .cob, .ccp, .cpy)

COBOL files, both original source files and files generated by the tool can be located on remote systems such as PDS members on z/OS. Other files can only reside on the workstation.

Related concepts

XML to COBOL mapping tools

Related tasks

Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Mapping reference

Mapping sessions

In a web service or XML-enabled enterprise application, XML data can flow in and out of a COBOL program. Therefore, with respect to the direction of the data flow, the *inbound mapping* applies to an XML document entering the COBOL program and the *outbound mapping* applies to the XML document leaving the COBOL program. In a generic case inbound and outbound mappings are different. Creation of mappings is referred to as *mapping session*. During a mapping session a single *mapping session file* is produced, which contains mapping metadata (information). This mapping metadata represents such information as names of the source files (such as a COBOL file or an XML document) for the mapped entities, location of the mapped elements and items within those files etc. Inbound and outbound sessions have separate session files. You start a mapping session by running the *Mapping session wizard* and creating a mapping session file. You can then open and customize mapping session files by using the *Mapping editor*.

Because the mapping file contains links to the COBOL and XML files relative to the workbench project, these links may not resolve correctly when you move the mapping file. Use caution when moving the mapping file from the directory where it was created (relative to the workbench project) or when importing already existing mapping files into the workbench. In general, moving or importing the mapping file will only work if the file is moved or imported into a similar directory structure (relative to the workbench project) with similar subdirectory names.

The *inbound mapping session* starts when you select an XML related file (WSDL, XML, XSD or DTD) as your source file. This selection will define the selection of your target file as being of COBOL type.

The *outbound mapping session* starts when you select a COBOL file as your source file. This selection will define the selection of your target file as being of XML-related type (WSDL, XML, XSD or DTD).

Related concepts

XML to COBOL mapping concepts
XML to COBOL mapping tools

Related tasks

Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Mapping reference

Elementary item and XSD simple type element mapping

During the inbound mapping session, you specify which COBOL data items at runtime will receive data from specific elements of the inbound XML document. In a *one-to-one mapping*, you can map a single *simple-type sender* element to a single *target receiver*. This is a *simple mapping*. You cannot map multiple senders to multiple receivers. These *many-to-many mappings* are not supported by the mapping tools.

During the outbound mapping session, you specify which simple-type XML *receiver* elements will receive data from specific COBOL *sender* items. The outbound session only allows a one-to-one type mapping.

When XML elements (of built-in or user-derived simple types) are mapped to the COBOL elementary data items in both inbound and outbound mapping, the senders and receivers must be *compatible* with one another. For details on this *type compatibility*, see the related reference link below.

In both inbound and outbound sessions you can only map XML elements. You cannot map XML attributes or other entities.

Related concepts

XML to COBOL mapping concepts
XML to COBOL mapping tools

Related tasks

Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Mapping reference

Elementary item and XML instance document element mapping

Because it is impossible, in a generic case, to precisely determine types of elements based on an XML instance document content, the mapping tools allow mappings between any XML instance document elements and any COBOL data type described in COBOL language types. See the related reference below for further details.

Related concepts

XML to COBOL mapping concepts
XML to COBOL mapping tools

Related tasks

Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Mapping reference

Elementary item and DTD element mapping

Since Document Type Declarations (DTD) describe a very broad set of XML elements, the mapping tools allow mappings between any DTD elements that are described as PCDATA and any COBOL data type described in COBOL language types. See the related reference below for further details.

Related concepts

XML to COBOL mapping concepts
XML to COBOL mapping tools

Related tasks

Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Mapping reference

Isomorphic and non-isomorphic simple mapping

The term ″isomorphic″ (as used here) means the following:

- Each composed element (in other words, an element containing other elements) of the XML instance document starting from the root has one and only one corresponding COBOL group item whose nesting depth is identical to the nesting depth of its XML equivalent

and

- Each non-composed element (in other words, an element that does not contain other elements) in the XML instance document starting from the top has one and only one corresponding COBOL elementary item whose nesting depth is identical to the nesting level of its XML equivalent and whose memory address at runtime can be uniquely identified.

The mapping tools define all other COBOL data structure/XML instance document pairs as *non-isomorphic structures*.

Isomorphic simple mapping is a simple mapping of COBOL items and XML elements that have both of the following properties:

- They belong to XML documents and COBOL groups that are identical in shape (isomorphic)
- With respect to the location of the structure, the mapping relates two isomorphic elements.

Isomorphic mapping can also exist between isomorphic subsets of otherwise non-isomorphic structures.

Non-isomorphic simple mapping is a simple mapping of COBOL items and XML elements belonging to XML documents and COBOL groups that are not identical in shape (non-isomorphic). Non-isomorphic mapping can also be created between non-isomorphic elements of isomorphic structures.

Mapping between non-isomorphic structures can be isomorphic if it maps corresponding elements of isomorphic subsets.

Both isomorphic and non-isomorphic simple mapping are supported for inbound and outbound mapping. See the related reference link below for examples of isomorphic and non-isomorphic mapping in Isomorphic and non- isomorphic element mapping.

Related concepts

XML to COBOL mapping concepts
XML to COBOL mapping tools

Related tasks

Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Mapping reference

Mapping repeating items

Elementary repeating items of the COBOL structure can only be mapped to the XML repeating elements that have the same number of occurrences. Multi-dimensional arrays (also called ″Nested tables″ in COBOL) can be mapped only to an isomorphic XML structure. Repeating items can be mapped only when the following conditions are true:

- Nested tables and the XML structure must have the same number of dimensions
- Each dimension must occur the same number of times
- Structures of each dimension are isomorphic

You cannot map individual dimensions of arrays.

Additionally, simple COBOL ″Occurs Depending On″ (ODO) repeating elementary and group items can be mapped to XML documents with the following restrictions (derived from the COBOL language requirements):

- If an ODO item is mapped to an XML element, the corresponding ODO object must also be mapped
- In the XML document, the element mapped to the COBOL ODO object item must appear before the XML element that is mapped to the corresponding COBOL ODO subject.

Note that the mapping editor will only check the ODO item requirement when you create the ODO mapping. If you later remove the ODO item mapping and leave the ODO itself mapped, the results of running the generated code will be unpredictable.

Related concepts

XML to COBOL mapping concepts
XML to COBOL mapping tools

Related tasks

Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Automatic group mapping

In an automatic group mapping, groups of XML elements (user-defined complex types) can be mapped to groups of COBOL elementary items (COBOL group items). If you create a mapping between an XML complex type and a COBOL group item in an inbound session, the child elements of the XML complex types will be automatically matched to the subordinate COBOL items. For the outbound session, you can map COBOL groups to XML complex types in a similar way. You cannot use automatic group mapping on groups that have any elementary items that are already mapped.

You can invoke the automatic group mapping function by selecting 'Match Mapping' action on the pop-up action menu in the Mapping Editor or from the Mapping pull-down menu on the Mapping Editor toolbar.

For automatic group mappings, the structure of COBOL group items must be compatible with the structure of XML complex types. Mapped groups are considered structurally compatible if the following conditions are met:

1. The XML instance document and the COBOL data structure it is being mapped to meet the following requirements:
   a. A *composed element* is an element containing other elements. Each composed element of the XML instance document starting from the root has one and only one corresponding COBOL group item whose nesting depth is identical to the nesting depth of its XML equivalent.
   b. A *non-composed element* is an element that does not contain other elements. Each non-composed element in the XML instance document starting from the top has one and only one corresponding COBOL elementary item whose nesting depth is identical to the nesting level of its XML equivalent and whose memory address at runtime can be uniquely identified.
   c. For the inbound mapping, the innermost complex type contains at least one simple type compatible in type with that of its COBOL mapped item.
   d. For the outbound mapping, every XML complex type must contain the same number of simple types compatible with that of their COBOL mapped items.
2. The mapped COBOL group does not contain subordinate redefining items. (The mapped group itself can be a redefining item)
3. The mapped COBOL group does not contain OCCURS DEPENDING ON constructs.

Automatic group mappings are always one-to-one.

Top-level mapping

When you create the mapping session file, the top-level object shown on the XML side is an XML node associated with the source or target XML document. The top-level COBOL object on the COBOL side is the COBOL file containing level 01 data structure that you selected in the Mapping session wizard.

Related concepts

XML to COBOL mapping concepts
XML to COBOL mapping tools

Related tasks

Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Mapping reference

Mapping XML model group elements

XML model group elements except for disjunction (or choice) will be mapped without regard to the specified constraint. In other words, the generated code will not perform any validation on whether, for example, XML elements that are part of sequence arrive in the inbound converter in the order specified by the XML schema's sequence constraint.

During the mapping session, you can map XML elements that are part of the choice group to COBOL items just as described above for other XML elements. The mapping editor user interface provides facilities to select a specific choice element.

Related concepts

XML to COBOL mapping concepts
XML to COBOL mapping tools

Related tasks

Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Mapping reference

Using mapping session files

You can use the mapping session files outside of the mapping editor session to generate the COBOL conversion programs and drivers. You can also start generation by invoking an action from the Mapping editor menu. In case of the inbound mapping session, the result will be an *inbound converter* and its associated *driver* program. In case of the outbound mapping session the *outbound converter* and its associated driver are generated.

Related concepts

XML to COBOL mapping concepts
XML to COBOL mapping tools

Related tasks

Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Mapping reference

Locales and code pages

Supported locales and code pages

COBOL for Windows® uses the POSIX-defined locale conventions. Locale value syntax: ll _CC.codepageID where

- ll -> A lowercase two-letter ISO language code
- CC - > An uppercase two-letter ISO country code
- codepageID -> The code page to be used for native DISPLAY and DISPLAY-1 data

When specifying locale and code page information, you must code a valid value for the locale name (ll CC) and a valid code page (codepageID) that corresponds to the locale name, as shown in the Locale and code pages table below.

You can use the characters that are represented in a supported code page in COBOL names, data definitions, literals, and comments. The locale in effect determines the code page for compiling source programs (including alphanumeric literal values). That is, the code page that is used for compilation is based on the locale setting at compile time. Thus, the evaluation of literal values in the source program is handled with the locale in effect at compile time.

The default value for Compile time Locale name is en_US. The default value for the ASCII code page is IBM-1252.

Note: In the table below, for a given locale name, the last ASCII code page listed in the set is the default.

The following table shows the locales that COBOL for Windows supports and the code pages that are valid for each locale.

The first column, Locale name, shows the valid combinations of ISO language code and ISO country code (language_COUNTRY) that are supported. The second column show the associated language, and the third column shows the associated country or area.

The fourth column, ASCII code pages, shows the ASCII code pages that are valid as the code page ID for the locale with the corresponding language_COUNTRY value.

Locale name	**Language**	**Country or area**	**ASCII code pages**	**Language group**
ar_AA	Arabic	Arabic Countries	IBM-864, IBM-1256	Arabic
be_BY	Byelorussian	Belarus	IBM-866, IBM-1251	Latin 5
bg_BG	Bulgarian	Bulgaria	IBM-855, IBM-1251	Latin 5
ca_ES	Catalan	Spain	IBM-850, IBM-1252	Latin 1
cs_CZ	Czech	Czech Republic	IBM-852, IBM-1250	Latin 2
da_DK	Danish	Denmark	IBM-437, IBM-850, IBM-1252	Latin 1
de_CH	German	Switzerland	IBM-437, IBM-850, IBM-1252	Latin 1
de_DE	German	Germany	IBM-437, IBM-850, IBM-1252	Latin 1
el_GR	Greek	Greece	IBM-1253	Greek
en_AU	English	Australia	IBM-437, IBM-1252	Latin 1
en_BE	English	Belgium	IBM-850, IBM-1252	Latin 1
en_GB	English	United Kingdom	IBM-437, IBM-850, IBM-1252	Latin 1
en_JP	English	Japan	IBM-437, IBM-850, IBM-1252	Latin 1
en_US	English	United States	IBM-437, IBM-850, IBM-1252	Latin 1
en_ZA	English	South Africa	IBM-437, IBM-1252	Latin 1
es_ES	Spanish	Spain	IBM-437, IBM-850, IBM-1252	Latin 1
fi_FI	Finnish	Finland	IBM-437, IBM-850, IBM-1252	Latin 1
fr_BE	French	Belgium	IBM-437, IBM-850, IBM-1252	Latin 1
fr_CA	French	Canada	IBM-863, IBM-850, IBM-1252	Latin 1

Locale name	Language	Country or area	ASCII code pages	Language group
fr_CH	French	Switzerland	IBM-437, IBM-850, IBM-1252	Latin 1
fr_FR	French	France	IBM-437, IBM-850, IBM-1252	Latin 1
hr_HR	Croatian	Croatia	IBM-852, IBM-1250	Latin 2
hu_HU	Hungarian	Hungary	IBM-852, IBM-1250	Latin 2
is_IS	Icelandic	Iceland	IBM-861, IBM-850, IBM-1252	Latin 1
it_CH	Italian	Switzerland	IBM-850, IBM-1252	Latin 1
it_IT	Italian	Italy	IBM-437, IBM-850, IBM-1252	Latin 1
iw_IL	Hebrew	Israel	IBM-862, IBM-1255	Hebrew
ja_JP	Japanese	Japan	IBM-943	Ideographic languages
ko_KR	Korean	Korea, Republic of	IBM-1363	Ideographic languages
lt_LT	Lithuanian	Lithuania	IBM-1257	Lithuanian
lv_LV	Latvian	Latvia	IBM-1257	Latvian
mk_MK	Macedonian	Macedonia,	IBM-855, IBM-1251	Latin 5
nl_BE	Dutch	Belgium	IBM-437, IBM-850, IBM-1252	Latin 1
nl_NL	Dutch	Netherlands	IBM-437, IBM-850, IBM-1252	Latin 1
no_NO	Norwegian	Norway	IBM-437, IBM-850, IBM-1252	Latin 1
pl_PL	Polish	Poland	IBM-852, IBM-1250	Latin 2
pt_BR	Portuguese	Brazil	IBM-850, IBM-1252	Latin 1
pt_PT	Portuguese	Portugal	IBM-860, IBM-850, IBM-1252	Latin 1
ro_RO	Romanian	Romania	IBM-852, IBM-850, IBM-1250	Latin 2
ru_RU	Russian	Russian federation	IBM-866, IBM-1251	Latin 5
sh_SP	Serbian (Latin)	Serbia	IBM-852, IBM-1250	Latin 2
sk_SK	Slovak	Slovakia	IBM-852, IBM-1250	Latin 2
sl_SL	Slovenian	Slovenia	IBM-852, IBM-1250	Latin 2
sq_AL	Albanian	Albania	IBM-850, IBM-1252	Latin 1
sv_SE	Swedish	Sweden	IBM-437, IBM-850, IBM-1252	Latin 1
th_TH	Thai	Thailand	IBM-874	Thai
tr_TR	Turkish	Turkey	IBM-857, IBM-1254	Turkish
uk_UA	Ukranian	Ukraine	IBM-866, IBM-1251	Latin 5
zh_CN	Chinese (simplified)	China	IBM-1386	Ideographic languages
zh_TW	Chinese (traditional)	Taiwan	IBM-950	Ideographic languages

Chapter 4. Mapping XML to COBOL

The steps that you would typically take to map your XML files to COBOL and generate converter and driver files are as follows

1. Switch to the Resource perspective
2. Create a project and a folder to hold your existing WSDL, XML, XSD, DTD, COBOL, and, possibly, generated converter and driver files.
3. Select the folder and create a new XML to COBOL mapping. Using the XML to COBOL mapping session wizard (Mapping wizard), create a new XML to COBOL mapping session file to specify or import your source file and target file, identify the root element for any of DTD or XML files that you specified, identify the COBOL data structure.
4. In the XML to COBOL mapping editor (Mapping editor), map the source and target structures by mapping nodes.
5. Generate the converters and the drivers using the XML to COBOL mapping converter generator wizard (Converter generator wizard).

After you generate your converters and drivers you can deploy and run them in conjunction with the existing COBOL application similar to the converters generated by the Web Service Enablement wizard.

Note: None of the code generated from the mapping tools will perform any type or structure validation beyond what is described in the documentation.

After you complete these mapping tasks, you are ready to generate conversion code.

Related concepts

XML to COBOL mapping tools

Related tasks

Creating a web service interface with the Web Services Enablement wizard
Creating a mapping session file
Editing a mapping session file
Generating mapping code

Related reference

Mapping reference

Creating a mapping session file

The first task you must complete to map XML to COBOL is to create a new mapping session file. You can use the XML to COBOL mapping session wizard (Mapping wizard) to create the mapping session file. In this wizard, you specify or import your source file and target file, identify the root element for any of DTD or XML files that you specified, and identify the COBOL data structure.

To create a mapping session file:

1. Launch the Mapping wizard. Select **File** > **New** > **Other**... > **XML Services for Enterprise** > **XML to COBOL mapping**.
2. Select the folder that will contain the mapping session file. The mapping session file is a file containing mapping metadata.

3. Type the name of the mapping session file, for example: `CBLMap.cmx`. Your filename must have extension `.cmx`. Click **Next**.
4. Select a WSDL, DTD, XSD, XML, or a COBOL file that you want to use as source file. If you want to use files that are not currently in the workbench, click **Import File** and fill in the fields in the Import wizard as necessary. Any files that you import will be listed in the Workbench Files list; you can now add them to the Selected Files list. When you have selected your source file, click Next.

   Note: You can only select dissimilar files as input and output. In other words if you selected COBOL as input you can only select WSDL, DTD, XSD, or XML as output and, if you selected WSDL, DTD, XSD or XML as input, you can only select COBOL as output. See the related link below for valid combinations.
5. Select a COBOL, WSDL, DTD, XSD or an XML file that you want to use as your target file. If you want to use a file that is not currently in the workbench, click Import File and fill in the fields in the Import wizard as necessary. Any file that you import will be listed in the Workbench Files list; you can now select it as the target file. When you have selected your target file, click **Next**.
6. Select the appropriate source and target items:
   - If you selected WSDL, DTD, XSD, XML as source (inbound session), select one of the root elements from the list that was imported from the XML file to serve as the source and select a top level COBOL data structure which will serve as the target.
   - If you selected COBOL as source (outbound session), select a COBOL data structure from the list to serve as source and select an XML root element to serve as the target.
7. Click **Finish**. The XML to COBOL mapping editor opens automatically. You are now ready to edit the mapping session file.

Related concepts

XML to COBOL mapping tools

Related tasks

Mapping XML to COBOL
Editing a mapping session file
Generating mapping code

Related reference

Mapping reference

Editing a mapping session file

The second task you must complete to map XML to COBOL is to edit an existing mapping session file. You use the XML to COBOL mapping editor (Mapping editor) to map the source and target structures by mapping nodes.

Complete the following instructions to create mappings between source XML elements and target COBOL data items, an inbound session:
1. Open either the Resource or the XML perspective (other perspectives may also work).
2. Open your XML to COBOL mapping session file. As described above, the mapping editor opens automatically upon finishing the new mapping session wizard. For existing mapping session files (`.cmx` files), you can use standard Eclipse opening mechanism (e.g. double-click) on the file to open the XML to COBOL mapping editor.
3. Select an element in the source view.
4. Select a COBOL data item in the target view.

5. Right-click and select **Create Mapping**.
6. Alternatively to steps 2-4 you can drag and drop the source XML element selection to the target COBOL data item. The mapping will be created automatically.

Note: The Create Mapping action and the drag/drop operation is disabled for incompatible items.

Use the CTRL key to select more than one element in the source or in the target.

To create mappings between source COBOL data items and target XML elements, an outbound session:
1. Open your XML to COBOL mapping session file as described above.
2. Select a COBOL data item in the source view.
3. Select an XML element in the target view.
4. Right-click and select **Create Mapping**.
5. Alternatively to steps 2-4 you can drag and drop the source COBOL item selection to the target XML element. The mapping will be created automatically.

In order to assist you in determining the data types of XML elements and COBOL data items, the type information will appear as a tooltip when you move the mouse pointer over the element or the data item.

Note that if the mapping session file is not in sync with the source files that it references, the mapping editor will generate an error message. This can happen, for example, if you changed the content of the mapped data structure without re-generating the mapping.

Related concepts

XML to COBOL mapping tools

Related tasks

Mapping XML to COBOL
Creating a mapping session file
Generating mapping code

Related reference

Mapping reference

Generating mapping code

When your goal is to map XML to COBOL, you need to generate COBOL converters and drivers that correspond to a particular mapping session. In the case of the inbound session, an inbound converter and its associated driver are generated. In the case of an outbound session, an outbound converter and its driver are generated. The naming convention for the driver and converter programs is the same as in the Web Service Enablement wizard. Code generation options are also the same as in the Web Service Enablement wizard .

To generate mapping code:
- Invoke the mapping converter generator wizard. Select **Mapping** -> **Generate mapping code** from within the mapping editor view toolbar. Alternatively, right mouse click on the mapping session file and select **Enable XML** -> **Generate Mapping Code**. The mapping converter generator wizard opens.
- The first and second pages gather information about locations and names of the target files. Enter target folder name and the names for the converter file and the driver file or accept the defaults. By

default the inbound converter name is formed by concatenating the original COBOL source file and a character ″I″. The default outbound converter file name is formed by concatenating the original COBOL source file and a character ″O″.

Note: If no elements/items were mapped in the mapping session, the Next and Finish buttons will be disabled and an informational message will be displayed. If some elements were mapped, you will be able to proceed to the next page.
Click **Next** to proceed to the XML converter options page or click Finish to accept default values for the rest of the wizard.

- On the XML converter options page specify the generation options for the converters. These are the same options as in the Web Service Enablement wizard. Click **Finish**.
- Similar to the Web Service Enablement wizard, either an inbound converter and a driver or an outbound converter and its driver are generated.

Related concepts

XML to COBOL mapping tools

Related tasks

Creating a web service interface with the Web Services Enablement wizard
Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Related reference

Mapping reference

Chapter 5. XML Converter Diagnostics

The XML Services for the Enterprise (XSE) generated outbound converters attempt to patch up data in outbound data structures so that a legal XML message can be produced.

By default the outbound converter filters characters from the data structure that are illegal in an XML document. Using the XML Services for the Enterprise COBOL generator preferences page, you can configure this behavior for debugging and performance purposes. From the WebSphere® Developer for zSeries® desktop **Window**->**preferences**->**XML Services for the Enterprise**->**Enable Web Service Wizard**->**COBOL Generator**]

On the preferences page, lists two options that are associated with converter filters.

- Filter characters illegal in XML 1.0 (outbound)

  The default for this option is **on**. The outbound converter scans both non-numeric and numeric data in the data structure and converts to an EBCDIC, ASCII or UNICODE space (depending on the outbound codepage), any character that is illegal in an XML document according to the XML 1.0 specification. For numerics, the scan is done after the number is converted to a textual representation.
- Halt on characters illegal in XML 1.0 (outbound)

  This option is turned **off** by default. The outbound converter scans both non-numeric and numeric data in the data structure and returns an exception if characters illegal in XML 1.0 are found. For numerics, the scan is done after the number is converted to a textual representation.

  If illegal non-numeric data is found, a message stating that the Data structure to XML conversion could not complete because the content of a non-numeric member of the data structure contained characters that are not legal in an XML document.

  If illegal numeric data is found, a message stating that the Data structure to XML conversion could not complete because the content of a numeric member of the data structure is invalid.

You can select only one of these options at a time or, you can select neither option for maximum performance of the outbound converter.

Note: Disabling both options provides no protection against including illegal characters in the XML document produced by the outbound converter.

Chapter 6. Setting preferences for XML Services for the Enterprise

To set preferences for XML Services for the Enterprise, do as follows:

1. Click **Window > Preferences**, then expand **XML Services for the Enterprise** and **Enable Web Service Wizard**, then click **COBOL Generator**
2. Specify defaults for settings used in XML Services for the Enterprise:
   a. In the **Program name prefix** field, specify the stem of the program name that is included in the IDENTIFICATION DIVISION of each generated COBOL program. If you type ACCT, for example, the wizard identifies the input converter program as ACCTI, the output converter program as ACCTO, and the driver as ACCTD
   b. In the **Author name** field, specify the value to be included in the AUTHOR paragraph of each generated COBOL program.
   c. In the **Inbound code page** list box, select the code page for encoding the XML input message
   d. In the **Host code page** list box, select the code page for the z/OS host system
   e. In the **Outbound code page** list box, select the code page for encoding the XML output message
3. Specify advanced generation options:
   a. Select the check box for **Decimal Point is Comma** if you wish the XML converter to interpret a comma and a period in numeric data as follows:
      - A comma is a decimal point
      - A period is a thousand separator

      Clear the check box for the opposite effect, as is true by default:
      - A comma is a thousand separator
      - A period is a decimal point

   b. Select the check box for **Generate minimum hierarchy in XML Schemas** if you wish the XML converter to reduce the data structure hierarchy when it is not needed to uniquely identify each element.

      When there are elements with the same tag name, the name of the element that occurs later in the document will be prefixed with enough of its parent tags to produce a unique name. This provides efficiency for message processing clients, reducing the number and complexity of objects that need to be instantiated.

      Clear the check box to create the full data structure hierarchy.

      If you require both hierarchical and flat XML messages, run the generation step twice, with and without this option selected.
   c. Select the check box for **Generate groups in XML Schemas** if you wish the XML converter to include groups in the generated XML schemas.

      Clear the check box to include group ″contents″ inline instead of using group references. This is useful for applications which do not support the use of groups and group references in XML schemas.

      The two forms of the schema are functionally equivalent, but if you require both kinds, run the generation step twice, with and without this option selected.
   d. Select the check box for **Filter characters illegal in XML 1.0 (outbound)** if you wish to scan both non-numeric and numeric data in the data structure and convert any character that is illegal in an XML document to an EBCDIC, ASCII or UNICODE space (depending on the outbound codepage).
   e. Select the check box for **Halt on characters illegal in XML 1.0 (outbound)** if you wish to scan both non-numeric and numeric data in the data structure and receive an exception if characters illegal in XML 1.0 are found.

4. Select **Apply** or, if you wish to restore the settings to the default values, Select **Restore Defaults**

Related concepts

Introduction to XML Services for the Enterprise

Related tasks

Creating a web service interface with the Web Services Enablement wizard

Chapter 7. Batch Processor

The batch processor is a command-line interface for creating enterprise web services descriptions (WSDL) and message converters for CICS and IMS applications. You communicate generation options to the processor in options files. This tool is useful if you need to create many services from many COBOL source files.

You have three files in which to provide the necessary options to the batch processor:

- Container.xml
- PlatformProperties.xml
- ServiceSpecification.xml

The reason for specifying most of the options in XML files is to allow automation of the options generation by customization tools such as WSED converter and service generation wizards.

Related tasks

Creating and populating options files
Starting the batch processor

Related references

Container.xml
PlatformProperties.xml
ServiceSpecification.xml

Creating and populating options files

Follow these steps to create the environment and populate the options in the properties files for the batch processor:

1. Create a directory to hold the batch processor input files. The remaining steps refer to this directory as your input directory.
2. In the PlatformProperties.xml file, set the default options properties that reflect your target run-time environment.

   You can copy the sample file (PlatformProperties.xml) to your input directory from the following location within the directory where WebSphere Developer is installed:

   wdz\wstools\eclipse\plugins\com.ibm.etools.xmlent.batch_6.0.0\Samples

   If you use this sample, modify it with a text editor (or a specialized XML editor) to suit your purpose.
3. In the ServiceSpecification.xml file, put the options properties for generating the specific sets of converters and service definitions. You can also override certain options specified in the platform properties file.

   You can copy the sample file to your input directory from the following location within the directory where WebSphere Developer is installed and rename the file as desired:

   wdz\wstools\eclipse\plugins\com.ibm.etools.xmlent.batch_6.0.0\Samples

   You can then edit the new file with a text editor (or a specialized XML editor) and set the options.
4. In the Container.xml file, specify the options that globally affect the generation of converters and of service definitions. You can also specify the locations of platform properties files and files that control options for generating individual sets of converters and service definitions.

You can copy the sample file Container.xml to your input directory from the following location within the directory where WebSphere Developer is installed and rename the file as desired:

wdz\wstools\eclipse\plugins\com.ibm.etools.xmlent.batch_6.0.0\Samples

Open the resulting file with a text editor (or a specialized XML editor) and set the appropriate PlatformProperties.xml and, for each individual set of converter and service definition generation options (ServiceSpecification.xml) files that you created in the previous step, add a GenerationSpec element that references the file that you created.

When you have finished creating your options files, you can start the batch processor.

Related concepts

Batch processor

Related tasks

Starting the batch processor

Related references

Container.xml
PlatformProperties.xml
ServiceSpecification.xml

Starting the batch processor

The batch processor is a command-line interface for creating enterprise web services descriptions (WSDL) and message converters for CICS and IMS applications.

Before you run the batch processor, close any running WebSphere Developer instance that uses the target workspace.

Start the batch processor by entering the following command from the command line (or executing it from a script):

```
xsebatch -s languageFile [-c | -w serviceName] | [-c -w serviceName]
        -f containerFile [-d workspace] [-e WS_installdir] [-verbose] [-version]
        [-overwrite=yes|no]
```

In this command, languageFile is the name of the language source file that contains the message definition. You can override this name by using the message specification option in the ServiceSpecification.xml file.

You specify either the -c parameter or the -w parameter or both, as follows:

- -c causes the set of language converters, the driver, and XML schemas to be generated. You can override this option by using the generateConverters and the generateSeparateXSD options in the Container.xml file and in the ServiceSpecification.xml file. The option generateSeparateXSD=true will produce XSD files only if -c (or generateConverters=true) is specified.
- -w serviceName causes the service definition files to be generated using the specified name for the web service. You can override this option by using the generateWSDL option in the Container.xml file and in the ServiceSpecification.xml file. The value of this parameter can be overridden by the value attribute of the EISService element in the ServiceSpecification.xml file. The default value is set to ″esvc″.

The variable containerFile is the name of the Container.xml file that holds the generation options. Most of the content of items in this file are optional, but a few are required and must be specified.

The following parameters are optional:

- -d workspace is the fully qualified path of the workspace to be used for the import . If this path is not specified, the default is taken from the environment variable called %workspace%. If that environment variable is not set, the default is set to %eclipse_root%\workspace
- -e WS_installdir is the eclipse subdirectory of the directory in which WebSphere Developer is installed. If not specified, the default is taken from the environment variable %eclipse_root%. If that environment variable is not set, the default is set to: **C:\Program Files\IBM\Rational\SDP\6.0\eclipse**

  Note: If the directory names contain spaces (for example, c:\test one\WDZ) you need to put these names in double quotes (for example, ″c:\test one\WDZ″). You need to use the double quotes only when specifying values for command line parameters -d and -e. If you use environment variables, omit double quotes for these values. Do not use a trailing backslash ('\') in any of the pathnames for the -d and -e options and the %workspace% and %eclipse_root% environment variables.
- -verbose causes the diagnostic messages to be printed to the console.
- -version causes the version, release and modification information to be printed to the console.
- -overwrite when set to ″yes″ (the default) causes the tool to write over all the generated files. If set to ″no″ a new file name is generated for each file that exists. The new name of the file will contain an integer number as suffix which will be incremented for each duplicate file until a unique name is found (For example, myfile12o.xsd). Overwriting converters and the XSD file can be further refined by the value of the overwrite attribute of the file generation specification elements of the XseSpec group described in Reference: Elements in XML Schemas for batch processing.

All other generation options are specified in XML files (either directly in the container file or in the XML files that the container file contains).

You can see the progress of the xsebatch command in the console along with any error messages.

After the xsebatch program has finished running, restart WebSphere Developer to view the generated files in the workspace, or browse the file system with Windows Explorer.

Related concepts

Batch processor

Related tasks

Creating and populating options files

Related references

Container.xml
PlatformProperties.xml
ServiceSpecification.xml

Container.xml

Container.xml is one of the options files for batch processing to create enterprise web services descriptions (WSDL) and message converters for CICS and IMS applications.

Purpose

Container.xml contains the generation options and references to the Platform.xml and ServiceSpecification.xml file.

Elements

Most of the content of items in this file is optional, but a few items are required. You can use the following elements in the Container.xml document:
- GenerationSpec
- GenerationSpecArray

Sample

The Container.xml sample illustrates the use of applicable elements and their attributes.

Schema

The schema for Container.xml provides the structure for the contents of the Container.xml document.

Related concepts

Batch processor

Related tasks

Creating and populating options files
Starting the batch processor

Related references

PlatformProperties.xml
ServiceSpecification.xml

GenerationSpec

Use this element to provide information about the ServiceSpecification.xml file, which specifies the options for generating individual sets of converters and web service definition (WSDL) files.

Contained by

GenerationSpecArray

Contains

None

Attributes

Name	Required?	Default	Description	
name	Yes	none	Specifies the location of the ServiceSpecification.xml file.	

Example

```
<GenerationSpecArray
              platformProperties="Platform.xml">
        <GenerationSpec name="ServiceSpecification.xml"/>
</GenerationSpecArray>
```

GenerationSpecArray

Use this top-level element of the Container.xml document to provide information that globally affects the generation of converters and service definitions. This element is required.

Contained by

None

Contains

GenerationSpec

Attributes

Name	Required?	Values	Default	Description
platform	No	OS390 **Note:** OS390 is the currently supported value. Other values are reserved for future use.	OS390	Specifies the target generation platform.
platformProperties	Yes			Specifies the location of the platform property file. The path can be relative (as in ./PlatformProperties.xml) or absolute.
generateConverters	No	true | false	false	Specifies whether to generate the converter set (inbound and outbound converters, driver). Note that the ″false″ setting of this attribute will cause the generateSeparateXSD to be forced to ″false″.
generateSeparateXSD	No	true | false	false	Specifies whether to generate a separate set of XML schemas that define the message (inbound and outbound). Note that the ″false″ setting of this attribute is enforced if the generateConverters attribute is set to false and is meaningful only if generateWSDL is set to true. A value of false specifies that the service definition file contains an embedded schema.
generateWSDL	No	true | false	false	Specifies whether to generate a service definition.

Example

```
<GenerationSpecArray generateConverters="true"
                     generateSeparateXSD="true"
                     generateWSDL="true"
                     platformProperties="Platform.xml">
        <GenerationSpec name="ServiceSpecification.xml"/>
</GenerationSpecArray>
```

Container.xml sample

```
<?xml version="1.0" encoding="UTF-8"?>
<GenerationSpecArray generateConverters="true"
                     generateSeparateXSD="true"
                     generateWSDL="true"
                     platform="OS390"
                     platformProperties="Platform.xml"
                     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <GenerationSpec name="ServiceSpecification.xml"/>
  <GenerationSpec name="ServiceSpecification2.xml"/>
</GenerationSpecArray>
```

Related references

Container.xml

Schema for Container.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="GenerationSpec">
        <xsd:complexType>
            <xsd:attribute name="name" type="xsd:string" use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="GenerationSpecArray">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="1" ref="GenerationSpec"/>
            </xsd:sequence>
            <xsd:attribute name="platform" type="xsd:string" use=" optional"/>
            <xsd:attribute name="platformProperties" type="xsd:string" use="required"/>
            <xsd:attribute name="generateConverters" type="xsd:boolean" use="optional"/>
            <xsd:attribute name="generateSeparateXSD" type="xsd:boolean" use="optional"/>
            <xsd:attribute name="generateWSDL" type="xsd:boolean" use="optional"/>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Related references

Container.xml

PlatformProperties.xml

PlatformProperties.xml is one of the options files for batch processing to create enterprise web services descriptions (WSDL) and message converters for CICS and IMS applications.

Purpose

PlatformProperties.xml has the default options properties that reflect your target run-time environment. The options affect, for example, the processing of the language types (such as COBOL data types) that are used in producing XML schema descriptions of web service messages that are based on that language type.

Elements

You can use the following elements in the PlatformProperties.xml document:
- CodegenProperty
- CodegenPropertyArray
- ConnectionProperty
- ConnectionPropertyArray
- ImportProperty
- ImportPropertyArray
- Platform
- PlatformArray

Sample

The PlatformProperties.xml sample illustrates the use of applicable elements and their attributes.

Schema

The schema for PlatformProperties.xml provides the structure for the contents of the PlatformProperties.xml document.

Related concepts

Batch processor

Related tasks

Creating and populating options files
Starting the batch processor

Related references

Container.xml
ServiceSpecification.xml

CodegenProperty

Use this element in the PlatformProperties.xml document to specify the properties for generating the converter code, or in the ServiceSpecification.xml document to override the code generation properties that you set in PlatformProperties.xml.

Contained by

CodegenPropertyArray

Contains

None

Attributes

Start the value of the name attribute with com.ibm.etools.xmlent.ui. The valid pairs of names and values for COBOL are as follows:

Name	Values	Default	Description
name	GEN_PROG_NAME		The generated program name. It must be formed according to COBOL language rules. The default is set to the COBOL copybook (or source program) file name shortened to 7 characters and converted to uppercase.
value		See Description	
name	GEN_AUTH_NAME		The generated author name. It must be formed according to COBOL language rules.
value		WSED	
name	GEN_IN_CP_LIST		Code page for the inbound (input) message. See Supported code pages (CCSIDs).
value		1140	
name	GEN_CP_LIST		Code page for the host converter program. See Supported code page combinations.
value		1140	
name	GEN_OUT_CP_LIST		Code page for the outbound (output) message. See Supported code page combinations.
value		1140	
name	GEN_DEC_COMMA		Property to control the format of the decimal fraction separator. If this option is set to true, the decimal fraction separator is set to be a comma.
value	true | false	false	
name	GEN_XSD_GROUPS		Property to control generation of groups in the XML schema. If this option is set to true, the XML schema can contain schema groups and grouprefs.
value	true | false	false	
name	GEN_FLAT_GEN		Property to control whether generated XML schemas have flat or hierarchical structure. If this option is set to true, the generators are set to generate flat schemas.
value	true | false	false	

Note: If you are generating artifacts for the IMS SOAP Gateway, you must specify UTF-8 (value=″1208″) as Code page for the inbound (input) message (GEN_IN_CP_LIST) and UTF-8 (value=″1208″) as Code page for the outbound (output) message (GEN_OUT_CP_LIST). Any other values are not allowed and will cause a runtime error if specified.

Example

```
<CodegenPropertyArray type="Cobol">
        <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_PROG_NAME" value="XCNV"/>
        <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_AUTH_NAME" value="WSED"/>
        <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_IN_CP_LIST" value="1140"/>
        <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_CP_LIST" value="1140"/>
        <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_OUT_CP_LIST" value="1140"/>
        <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_DEC_COMMA" value=" false"/>
        <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_FLAT_GEN" value=" false"/>
</CodegenPropertyArray>
```

Supported code pages (CCSIDs)

CCSIDs	Description
813	ISO 8859-7 Greek / Latin
819	ISO 8859-1 Latin 1 / Open Systems
920	ISO 8859-9 Latin 5 (ECMA-128, Turkey TS-5881)
930	Japanese EBCDIC (Katakana-based)
933	Korean EBCDIC
935	Simplified Chinese EBCDIC
937	Traditional Chinese EBCDIC
939	Japanese EBCDIC (Latin-based)
1047	Latin 1 / Open Systems
1200	Unicode, UTF-16
1208	Unicode, UTF-8
1364	Korean EBCDIC, including Euro
1371	Traditional Chinese EBCDIC, including Euro
1388	Simplified Chinese EBCDIC, including GBK and GB18030
1390	Japanese EBCDIC (Katakana-based), including Euro
1399	Japanese EBCDIC (Latin-based), including Euro
5026	Japanese EBCDIC (Katakana-based), subset of 930
5035	Japanese EBCDIC (Latin-based), subset of 939
01140, 00037	USA, Canada, etc. Euro Country Extended Code Page (ECECP), Country Extended Code Page
01141, 00273	Austria, Germany ECECP, CECP
01142, 00277	Denmark, Norway ECECP, CECP
01143, 00278	Finland, Sweden ECECP, CECP
01144, 00280	Italy ECECP, CECP
01145, 00284	Spain, Latin America (Spanish) ECECP, CECP
01146, 00285	UK ECECP, CECP
01147, 00297	France ECECP, CECP
01148, 00500	International ECECP, CECP
01149, 00871	Iceland ECECP, CECP

Related references

CodegenProperty

Supported code page combinations

Inbound	Host	Outbound
813	813	813, 1200, 1208
819	819	819, 1200, 1208
920	920	920, 1200, 1208
1047	1047	1047, 1200, 1208

Inbound	Host	Outbound
01140, 00037	01140, 00037	37, 1140, 1200, 1208
01141, 00273	01141, 00273	273, 1141, 1200, 1208
01142, 00277	01142, 00277	277, 1142, 1200, 1208
01143, 00278	01143, 00278	278, 1143, 1200, 1208
01144, 00280	01144, 00280	280, 1144, 1200, 1208
01145, 00284	01145, 00284	284, 1145, 1200, 1208
01146, 00285	01146, 00285	285, 1146, 1200, 1208
01147, 00297	01147, 00297	297, 1147, 1200, 1208
01148, 00500	01148, 00500	500, 1148, 1200, 1208
01149, 00871	01149, 00871	871, 1149, 1200, 1208
1200, 1208	01140, 00037	37, 1140, 1200, 1208
1200, 1208	01141, 00273	273, 1141, 1200, 1208
1200, 1208	01142, 00277	277, 1142, 1200, 1208
1200, 1208	01143, 00278	278, 1143, 1200, 1208
1200, 1208	01144, 00280	280, 1144, 1200, 1208
1200, 1208	01145, 00284	284, 1145, 1200, 1208
1200, 1208	01146, 00285	285, 1146, 1200, 1208
1200, 1208	01147, 00297	297, 1147, 1200, 1208
1200, 1208	01148, 00500	500, 1148, 1200, 1208
1200, 1208	01149, 00871	871, 1149, 1200, 1208
1200, 1208	930	1200, 1208
1200, 1208	933	1200, 1208
1200, 1208	935	1200, 1208
1200, 1208	937	1200, 1208
1200, 1208	939	1200, 1208
1200, 1208	1364	1200, 1208
1200, 1208	1371	1200, 1208
1200, 1208	1388	1200, 1208
1200, 1208	1390	1200, 1208
1200, 1208	1399	1200, 1208
1200, 1208	5026	1200, 1208
1200, 1208	5035	1200, 1208

Note: For the IMS SOAP Gateway, only UTF-8 (value 1208) as Code page for the inbound (input) message and UTF-8 (value 1208) as Code page for the outbound (output) message are supported. Any other values are not allowed and will cause a runtime error if specified.

Related references

CodegenProperty

CodegenPropertyArray

Use this element to specify the programming language for the code to be generated and as the container for the code generation properties for the converter. You can use this element in either PlatformProperties.xml or ServiceSpecification.xml. If you use it in both documents, the specification in ServiceSpecification.xml overrides the specification in PlatformProperties.xml.

Contained by

EISService (in ServiceSpecification.xml)

Platform (in PlatformProperties.xml)

Contains

CodegenProperty

Attributes

Name	Required?	Values	Default	Description
type	No	Cobol **Note:** Cobol is the currently supported value. Other values are reserved for future use.	Cobol	Specifies the programming language.

Example

```
<CodegenPropertyArray type="Cobol">
        <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_PROG_NAME" value="XCNV"/>
        <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_AUTH_NAME" value="WSED"/>
        <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_IN_CP_LIST" value="1140"/>
        <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_CP_LIST" value="1140"/>
        <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_OUT_CP_LIST" value="1140"/>
        <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_DEC_COMMA" value=" false"/>
        <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_FLAT_GEN" value=" false"/>
</CodegenPropertyArray>
```

ConnectionProperty

Use this element in the PlatformProperties.xml document to specify the properties for a server connection, or in the ServiceSpecification.xml document to override the server connection properties that you set in PlatformProperties.xml. This element is required in the PlatformProperties.xml document.

Contained by

ConnectionPropertyArray

Contains

None

Attributes

Name	Values	Required?	Description	
name		Yes	Specifies connection protocol and location of the web service.	
value	See Description			

Examples

```
<ConnectionProperty name="connectionURI" value="http://winmvsn0.cpit.hursley.ibm.com:8888/CICS/XMLS/DFHWSDSH/ACTDSOAP"/>
```

```
<ConnectionProperty name="connectionURI" value="http://localhost:8080/imssoap/services/IMSPHBKPort"/ >
```

ConnectionPropertyArray

Use this element as the container for connection properties in either PlatformProperties.xml or ServiceSpecification.xml. If you use it in both documents, the specification in ServiceSpecification.xml overrides the specification in PlatformProperties.xml. This element is optional in both files.

Contained by

EISService (in ServiceSpecification.xml)

Platform (in PlatformProperties.xml)

Contains

ConnectionProperty

Attributes

None

Example

```
<ConnectionPropertyArray>
 <ConnectionProperty name="connectionURI"
  value="http://winmvsn0.cpit.hursley.ibm.com:8888/CICS/XMLS/DFHWSDSH/ACTDSOAP"/>
</ConnectionPropertyArray>
```

ImportProperty

Use this element to provide information about the compiler options as appropriate for the programming language that you specify in the ImportPropertyArray element. You can use this element in the ServiceSpecification.xml document to override the compiler options that you set in PlatformProperties.xml. If you want to generate language converters, you are required to specify this element in the PlatformProperties.xml document.

Contained by

ImportPropertyArray

Contains

None

Attributes

Start the value of the name attribute with com.ibm.etools.cobol. Note that if you are generating language converters, the defaults do not apply: You must specify all of the properties. The valid pairs of names and values for COBOL are as follows:

Name	Values	Default	Description
name	COBOL_TRUNC		Specifies the setting of the COBOL TRUNC compile-time option.
value	STD | OPT | BIN	STD	
name	COBOL_NSYMBOL		Specifies the setting of the COBOL NSYMBOL compile-time option.
value	NATIONAL | DBCS	NATIONAL	
name	COBOL_QUOTE		Specifies the setting of the COBOL QUOTE compile-time option.
value	DOUBLE | SINGLE	DOUBLE	
name	COBOL_EXTENSION_CBL		Specifies the default extension behavior. A file is assumed to be a complete COBOL program if it has the extension of .cbl, .ccp or .cob. A file is assumed to be a copy book if it has the extension .cpy. If the file is a copybook member then it should consist of only one or more 01 data structures or 01 or 77 elementary data item definition. The user can change the default extension behavior by setting it to ″FP″ or ″DS″.
value	FP | DS	FP (FP - Full program, DS - Only data structures)	
name	COBOL_EXTENSION_CCP		Specifies the default extension behavior. A file is assumed to be a complete COBOL program if it has the extension of .cbl, .ccp or .cob. A file is assumed to be a copy book if it has the extension .cpy. If the file is a copybook member then it should consist of only one or more 01 data structures or 01 or 77 elementary data item definition. The user can change the default extension behavior by setting it to ″FP″ or ″DS″.
value	FP | DS	FP (FP - Full program, DS - Only data structures)	
name	COBOL_EXTENSION_COB		Specifies the default extension behavior. A file is assumed to be a complete COBOL program if it has the extension of .cbl, .ccp or .cob. A file is assumed to be a copy book if it has the extension .cpy. If the file is a copybook member then it should consist of only one or more 01 data structures or 01 or 77 elementary data item definition. The user can change the default extension behavior by setting it to ″FP″ or ″DS″.
value	FP | DS	FP (FP - Full program, DS - Only data structures)	
name	COBOL_EXTENSION_CPY		Specifies the default extension behavior. A file is assumed to be a complete COBOL program if it has the extension of .cbl, .ccp or .cob. A file is assumed to be a copy book if it has the extension .cpy. If the file is a copybook member then it should consist of only one or more 01 data structures or 01 or 77 elementary data item definition. The user can change the default extension behavior by setting it to ″FP″ or ″DS″.
value	FP | DS	FP (FP - Full program, DS - Only data structures)	

Name	Values	Default	Description

name	COBOL_COMPILE_TIME_LOCALE		Specifies the index number for the locale. From the Locale and code pages table, you can determine the index number by counting the row number of the locale name you are interested in. For example:
			` 1 - ar_AA`
			`14 - en_US`
			`50 - zh_TW`
			Please see Locale and code pages for additional information.

value	1 through 50	14	

name	COBOL_ASCII_CODEPAGE		Specifies the code page for a locale. From Locale and code pages table, the rightmost code-page entry in **ASCII code pages** column is the default for a given locale name.
			For example: for locale en_US, the Locale and code pages table specifies:
			Locale name: `en_US`
			Language: `English`
			Country or area: `United States`
			ASCII code pages: `IBM-437, IBM-850, IBM-1252`
			Language group: `Latin 1`
			for the en_US locale, the values for COBOL_ASCII_CODEPAGE correspond to the ASCII code pages as follows:
			`2 - IBM-437`
			`1 - IBM-850`
			`0 - IBM-1252`

value	0 | 1 | 2	0	

name	COBOL_ERROR_MSGS_LANG		Specifies the language used to display the syntax errors.
			The values correspond to the languages as follows:
			`0 - en_US    1 - ja_JP    2 - zh_TW`
			`3 - zh_CN    4 - ko_KR    5 - it_IT`
			`6 - fr_FR    7 - es_ES    8 - de_DE`
			`9 - pt_BR`
			The default is en_US. To change this value, use the preferences page.

value	0 through 9	0	

Example

```
<ImportPropertyArray type="Cobol">
        <ImportProperty name="com.ibm.etools.cobol.COBOL_TRUNC" value="STD"/>
        <ImportProperty name="com.ibm.etools.cobol.COBOL_NSYMBOL" value="DBCS"/>
        <ImportProperty name="com.ibm.etools.cobol.COBOL_QUOTE" value="DOUBLE"/>
</ImportPropertyArray>
```

ImportPropertyArray

Use this element to specify the programming language for the file to be imported and as the container for import properties. You can use this element in the ServiceSpecification.xml document to override the programming language that you set in PlatformProperties.xml. If you want to generate language converters, you are required to specify this element in the PlatformProperties.xml document.

Contained by

EISProject (in ServiceSpecification.xml)

Platform (in PlatformProperties.xml)

Contains

ImportProperty

Attributes

Name	Required?	Values	Default	Description
type	Yes	Cobol **Note:** Cobol is the currently supported value. Other values are reserved for future use.	Cobol	Specifies the programming language.

Example

```
<ImportPropertyArray type="Cobol">
 <ImportProperty name="com.ibm.etools.cobol.COBOL_TRUNC" value="STD"/>
 <ImportProperty name="com.ibm.etools.cobol.COBOL_NSYMBOL" value="DBCS"/>
 <ImportProperty name="com.ibm.etools.cobol.COBOL_QUOTE" value="DOUBLE"/>
</ImportPropertyArray>
```

Platform

Use this element to specify the platform for which the web service will be generated and as a container for properties that affect global service generation options.

Contained by

PlatformArray

Contains

ImportPropertyArray, ConnectionPropertyArray, CodegenPropertyArray

Attributes

Name	Required?	Values	Default	Description
name	Yes	OS390 **Note:** OS390 is the currently supported value. Other values are reserved for future use.	OS390	Specifies the target generation platform.

Example

```
<PlatformArray>
<Platform name="OS390">
   <ImportPropertyArray type="Cobol">
      <ImportProperty name="com.ibm.etools.cobol.COBOL_TRUNC" value="STD"/>
      <ImportProperty name="com.ibm.etools.cobol.COBOL_NSYMBOL" value="DBCS"/>
      <ImportProperty name="com.ibm.etools.cobol.COBOL_QUOTE" value="DOUBLE"/>
   </ImportPropertyArray>
   <ConnectionPropertyArray>
      <ConnectionProperty name="connectionURI"
          value="http://winmvsn0.cpit.hursley.ibm.com:8888/CICS/XMLS/DFHWSDSH/ACTDSOAP"/>
   </ConnectionPropertyArray>
   <CodegenPropertyArray type="Cobol">
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_PROG_NAME" value="XCNV"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_AUTH_NAME" value="WSED"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_IN_CP_LIST" value="1140"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_CP_LIST" value="1140"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_OUT_CP_LIST" value="1140"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_DEC_COMMA" value=" false"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_FLAT_GEN" value=" false"/>
   </CodegenPropertyArray>
</Platform>
</PlatformArray>
```

PlatformArray

This top-level element of the PlatformProperties.xml document provides information about the target run-time environment.

Contained by

None

Contains

Platform

Attributes

None

Example

```
<PlatformArray>
<Platform name="OS390">
   <ImportPropertyArray type="Cobol">
      <ImportProperty name="com.ibm.etools.cobol.COBOL_TRUNC" value="STD"/>
      <ImportProperty name="com.ibm.etools.cobol.COBOL_NSYMBOL" value="DBCS"/>
      <ImportProperty name="com.ibm.etools.cobol.COBOL_QUOTE" value="DOUBLE"/>
   </ImportPropertyArray>
   <ConnectionPropertyArray>
      <ConnectionProperty name="connectionURI"
          value="http://winmvsn0.cpit.hursley.ibm.com:8888/CICS/XMLS/DFHWSDSH/ACTDSOAP"/>
   </ConnectionPropertyArray>
   <CodegenPropertyArray type="Cobol">
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_PROG_NAME" value="XCNV"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_AUTH_NAME" value="WSED"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_IN_CP_LIST" value="1140"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_CP_LIST" value="1140"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_OUT_CP_LIST" value="1140"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_DEC_COMMA" value=" false"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_FLAT_GEN" value=" false"/>
   </CodegenPropertyArray>
</Platform>
</PlatformArray>
```

PlatformProperties.xml sample

```
<PlatformArray>
<Platform name="OS390">
   <ImportPropertyArray type="Cobol">
      <ImportProperty name="com.ibm.etools.cobol.COBOL_TRUNC" value="STD"/>
      <ImportProperty name="com.ibm.etools.cobol.COBOL_NSYMBOL" value="DBCS"/>
      <ImportProperty name="com.ibm.etools.cobol.COBOL_QUOTE" value="DOUBLE"/>
   </ImportPropertyArray>
   <ConnectionPropertyArray>
       <ConnectionProperty name="connectionURI"
                           value="http://winmvsn0.cpit.hursley.ibm.com:8888/CICS/XMLS/DFHWSDSH/ACTDSOAP"/>
   </ConnectionPropertyArray>
   <CodegenPropertyArray type="Cobol">
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_PROG_NAME" value="XCNV"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_AUTH_NAME" value="WSED"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_IN_CP_LIST" value="1140"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_CP_LIST" value="1140"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_OUT_CP_LIST" value="1140"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_DEC_COMMA" value=" false"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_FLAT_GEN" value=" false"/>
   </CodegenPropertyArray>
</Platform>
</PlatformArray>
```

Related references

PlatformProperties.xml

Schema for PlatformProperties.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="CodegenProperty">
        <xsd:complexType>
            <xsd:attribute name="name" type="xsd:string" use="required"/>
            <xsd:attribute name="value" type="xsd:string" use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="CodegenPropertyArray">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="1" ref="CodegenProperty"/>
            </xsd:sequence>
            <xsd:attribute name="type" type="xsd:string" use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="ConnectionProperty">
        <xsd:complexType>
            <xsd:attribute name="name" type="xsd:string" use="required"/>
            <xsd:attribute name="value" type="xsd:string" use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="ConnectionPropertyArray">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="1" ref="ConnectionProperty"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="ImportProperty">
        <xsd:complexType>
            <xsd:attribute name="name" type="xsd:string" use="required"/>
            <xsd:attribute name="value" type="xsd:string" use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="ImportPropertyArray">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="1" ref="ImportProperty"/>
            </xsd:sequence>
            <xsd:attribute name="type" type="xsd:string" use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="Platform">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="1" ref="ImportPropertyArray"/>
                <xsd:element maxOccurs="1" minOccurs="0" ref="ConnectionPropertyArray"/>
                <xsd:element maxOccurs="1" minOccurs="0" ref="CodegenPropertyArray"/>
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:string" use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="PlatformArray">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="1" ref="Platform"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Related references

PlatformProperties.xml

ServiceSpecification.xml

ServiceSpecification.xml is one of the options files for batch processing to create enterprise web services descriptions (WSDL) and message converters for CICS and IMS applications.

Purpose

ServiceSpecification.xml has the options for generating specific sets of converters and service definitions. In this file you can also override certain options that you specified in the PlatformProperties.xml file.

Elements

You can use the following elements in the ServiceSpecification.xml document:
- CodegenProperty
- CodegenPropertyArray
- ConnectionProperty
- ConnectionPropertyArray
- ConverterSpecIn
- ConverterSpecOut
- CorrelatorSpec
- DriverSpec
- EISProject
- EISService
- ImportProperty
- ImportPropertyArray
- InputMessage

- InputOutputMessage
- Operation
- OperationProperty
- OperationPropertyArray
- OutputMessage
- RedefinesArray
- RedefineSelection
- ServiceProperty
- ServicePropertyArray
- XsdSpecIn
- XsdSpecOut
- XseSpec

Sample

The ServiceSpecification.xml sample illustrates the use of applicable elements and their attributes.

Schema

The schema for ServiceSpecification.xml provides the structure for the contents of the ServiceSpecification.xml document.

Related concepts

Batch processor

Related tasks

Creating and populating options files
Starting the batch processor

Related references

Container.xml
PlatformProperties.xml

ConverterSpecIn

Use this element of the ServiceSpecification.xml document to specify the generation options for the inbound converter.

Contained by

XseSpec

Contains

None

Attributes

Name	Required?	Values	Default	Description
fileName	No	See Description	Data source file name concatenated with ″I″	Specifies the name of the output file.
overwrite	No	true | false	true	Specifies whether to overwrite the output file if it exists.
programName	No	See Description	Value of CodegenProperty GEN_PROG_NAME concatenated with ″I″	Specifies the program name of the main program entry.

Example

```
<ConverterSpecIn fileName="DFH0CSTDI.cbl" overwrite="true" programName="XCNVI"/>
```

ConverterSpecOut

Use this element of the ServiceSpecification.xml document to specify the generation options for the outbound converter.

Contained by

XseSpec

Contains

None

Attributes

Name	Required?	Values	Default	Description
fileName	No	See Description	Data source file name concatenated with ″O″	Specifies the name of the output file.
overwrite	No	true | false	true	Specifies whether to overwrite the output file if it exists.
programName	No	See Description	Value of CodegenProperty GEN_PROG_NAME concatenated with ″O″	Specifies the program name of the main program entry.

Example

```
<ConverterSpecOut fileName="DFH0CSTDO.cbl" overwrite="true" programName="XCNVO"/>
```

CorrelatorSpec

Use this element of the ServiceSpecification.xml document to specify the generation options for the correlator information. The correlator file is used by the IMS SOAP Gateway.

Contained by

XseSpec

Contains

None

Attributes

Name	Required?	Values	Default	Description
AdapterType	No	See Description	IBM® COBOL XML Adapter	Specifies the adapter that is used for data transformation. You can choose the given value or enter an alphanumeric value up to 8 characters in length.
connectionBundleName	No	See Description	N/A	Provides the name of the connection bundle the web service uses to connect to IMS. Connection bundles are defined in the connection specification XML file maintained by the IMS SOAP Gateway and can be updated using the IMS SOAP Gateway Deployment tool
executionTimeout	No	See Description	0	Specifies the time that the IMS Connect waits for a response from IMS. Valid range: 0-3600000 (in milliseconds).
fileName	No	See Description	Data source file name concatenated with ″xml″	Specifies the name of the output file.
ltermName	No	See Description	None	specifies the IMS Specific property provided by the user. You can set the value of this property if the client application wants to provide an LTERM override name. This name is in the IMS application program’s I/O PCB, with the intent that the IMS application makes logic decisions based on the override value.
overwrite	No	true | false	true	Specifies whether to overwrite the output file if it exists.
soapAction	No	See Description	Data source file name (no extension) prefixed with ″urn″	Specifies the SOAP action.
socketTimeout	No	See Description	0	Specifies the time that the SOAP Gateway waits for a response from IMS Connect. Valid range: Positive integer (in milliseconds).

Example

```
<CorrelatorSpec fileName="IMSPHBK.xml"overwrite="true"soapAction="urn:IMSPHBK" socketTimeout="0" executionTimeout="0" co
```

DriverSpec

Use this element to specify the programming language for the code to be generated and as the container for the code generation properties for the converter. You can use this element in either PlatformProperties.xml or ServiceSpecification.xml. If you use it in both documents, the specification in ServiceSpecification.xml overrides the specification in PlatformProperties.xml.

Contained by

XseSpec

Contains

None

Attributes

Name	Required?	Values	Default	Description
fileName	No	See Description	Data source file name concatenated with ″D″	Specifies the name of the output file.
overwrite	No	true | false	true	Specifies whether to overwrite the output file if it exists.
programName	No	See Description	Value of CodegenProperty GEN_PROG_NAME concatenated with ″D″	Specifies the program name of the main program entry.
businessPgmName	No	See Description	Data source file name up to 8 characters. If the name is longer than 8 characters only the first 8 characters will be used to form the default. The specified name must follow COBOL conventions for the program name.	Specifies the existing business program that the XML converters call. This is the program that you are enabling for processing and/or producing XML messages (to act as a web service, for example.)
driverType	No	BATCH | SOAP_FOR_CICS | WEB_SERVICES_CICS | IMS_SOAP_GATEWAY	SOAP_FOR_CICS	Specifies the type of drivers and converters to generate for a specific subsystem (such as CICS, IMS, TSO). BATCH: Basic converter and driver types running in batch, under TSO or USS SOAP_FOR_CICS: Converters and drivers to be deployed into a SOAP for CICS runtime WEB_SERVICES_CICS: Converters and drivers to be deployed into a CICS Web services runtime IMS_SOAP_GATEWAY: Converters and drivers to be deployed into a IMS SOAP Gateway runtime

Example

```
<DriverSpec fileName="IMSPHBKD.cbl" driverType="IMS_SOAP_GATEWAY" programName="XCNVD" businessPgmName="IMSPHBK" />
```

EISProject

Use this top-level element of the ServiceSpecification.xml document to provide information about the web service.

Contained by

None

Contains

CodegenPropertyArray, EISService, ImportPropertyArray

Attributes

Name	Required?	Values	Default	Description
name	No	See Description	EISProject	Specifies the name of the project that contains the generated files. If the project does not exist, it will be created automatically.

Example

```
<EISProject name="CICSSample">
        <!-- Use the ImportPropertyArray to override values from PlatformProperties.xml -->
        <ImportPropertyArray type="Cobol">
                <ImportProperty name="com.ibm.etools.cobol.COBOL_TRUNC" value="BIN" />
                <ImportProperty name="com.ibm.etools.cobol.COBOL_NSYMBOL" value="DBCS" />
                <ImportProperty name="com.ibm.etools.cobol.COBOL_QUOTE" value="DOUBLE" />
        </ImportPropertyArray>
        <CodegenPropertyArray type="Cobol">
                <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_PROG_NAME" value="XCNV"/>
                <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_AUTH_NAME" value="WSED"/>
                <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_IN_CP_LIST" value="1140"/>
                <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_CP_LIST" value="1140"/>
                <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_OUT_CP_LIST" value="1140"/>
                <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_DEC_COMMA" value=" false"/>
                <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_FLAT_GEN" value=" false"/>
        </CodegenPropertyArray>
        <EISService name="CustomerInfo" type="CICS" targetNameSpace="http://cics.sample"
                generateConverters="true"
                generateSeparateXSD="false"
                generateWSDL="true"
                portType="myPortType" >
        </EISService>
</EISProject>
```

EISService

Use this element of the ServiceSpecification.xml document to provide information about the web service and as a container for the required Operation container and, optionally, for the ConnectionPropertyArray and the ServicePropertyArray.

Contained by

EISProject

Contains

ConnectionPropertyArray, Operation, ServicePropertyArray

Attributes

Name	Values	Default	Description
name	See Description	The default is taken from the xsebatch command line parameter -w.	Specifies the name of the web service. The name of the web services definition file (WSDL) that is generated uses this name.
generateConverters	true | false	false	Specifies whether to generate the converter set (inbound and outbound converters, driver).
generateSeparateXSD	true | false	false	Specifies whether to generate a separate set of XML schemas that define the message (inbound and outbound). Note that the ″false″ setting of this attribute is meaningful only if generateWSDL is set to true. A value of false specifies that the service definition file contains an embedded schema.
generateWSDL	true | false	false	Specifies whether to generate a service definition.
targetNamespace	See Description	svcNS	Specifies the URL of the target namespace.
type	CICS | IMS	CICS	Specifies the relevant subsystem.
targetFilesURI	See Description	Value of targetNamespace, if specified. Otherwise it is set to file://target.files	Specifies the URI of the location where the output files will be generated, relative to the top level project.

Example

```
<EISProject name="CICSSample">
        <EISService name="CustomerInfo" type="CICS" targetNamespace="http://cics.sample"
                targetFilesURI="file://my.files"
                generateConverters="true"
                generateSeparateXSD="false"
                generateWSDL="true" >

        </EISService>
</EISProject>
```

InputMessage

Use this element to define messages for ONE-WAY operations or REQUEST_RESPONSE (SOLICIT_RESPONSE) operations if the input and output messages are of different types. Note that for ONE_WAY operations, you are allowed to specify only the InputMessage. OutputMessage is not allowed and will cause unpredictable results during the generation process.

Contained by

Operation

Contains

ItemSelectionArray

Attributes

Name	Required?	Values	Default	Description
name	No	See Description	esvc	Specifies a name for the messages for the WSDL file.
importDirectory	No	See Description	The input directory	Specifies the directory for the source file.
importFile	Yes (unless the -s command line option is specified)	See Description	The name provided in the xsebatch command-line parameter -s.	Specifies the file name that contains the data definition to be used in creating the web service operation message types. **Note:** Only COBOL data definitions are supported and are subject to the restrictions that are specified in WebSphere Developer documentation.
nativeTypeName	No	See Description	For COBOL, the name of the first available level 01 data item name.	Specifies the name of the data type that is to be imported from the importFile, such as DFHCOMMAREA for a CICS COBOL application. An error message is generated on the console if a parse of the importFile does not identify the nativeTypeName as a valid data type.

Example

```
<InputMessage name="PhoneBookRequest" importDirectory="." importFile="Ex01z.cbl" nativeTypeName="input-msg">
   <RedefinesArray>
      <RedefineSelection redefine="input-msg.redParent.redefd" useRedefinition="input-msg.redParent.redefd2"/>
   </RedefinesArray>
   <ItemSelectionArray>
       <ItemSelection itemName="input-msg.redParent"/>
       <ItemSelection itemName="input-msg.in-extn"/>
       <ItemSelection itemName="input-msg.in-zip"/>
       <ItemSelection itemName="input-msg.in-ll"/>
   </ItemSelectionArray>
</InputMessage>
```

InputOutputMessage

Use this element to define messages for REQUEST-RESPONSE operations if the input and output messages are of the same type.

Contained by

Operation

Contains

ItemSelectionArray

Attributes

Name	Required?	Values	Default	Description
name	No	See Description	esvc	Specifies a name for the messages for the WSDL file.
importDirectory	No	See Description	The input directory	Specifies the directory for the source file.

Name	Required?	Values	Default	Description
importFile	Yes (unless the -s command line option is specified)	See Description	The name provided in the xsebatch command-line parameter -s.	Specifies the file name that contains the data definition to be used in creating the web service operation message types. **Note:** Only COBOL data definitions are supported and are subject to the restrictions that are specified in WebSphere Developer documentation.
nativeTypeName	No	See Description	For COBOL, the name of the first available level 01 data item name.	Specifies the name of the data type that is to be imported from the importFile, such as DFHCOMMAREA for a CICS COBOL application. An error message is generated on the console if a parse of the importFile does not identify the nativeTypeName as a valid data type.

Example

```
<InputOutputMessage name="PhoneBookMsg" importDirectory="." importFile="Ex01z.cbl" nativeTypeName="input-msg">
    <RedefinesArray>
       <RedefineSelection redefine="input-msg.redParent.redefd" useRedefinition="input-msg.redParent.redefd2"/>
    </RedefinesArray>
    <ItemSelectionArray>
       <ItemSelection itemName="input-msg.redParent"/>
       <ItemSelection itemName="input-msg.in-extn"/>
       <ItemSelection itemName="input-msg.in-zip"/>
       <ItemSelection itemName="input-msg.in-ll"/>
    </ItemSelectionArray>
</InputOutputMessage>
```

ItemSelection

Use this element to select individual data item in the COBOL structure.

The selection rules apply as follows:
- When any item is selected, all parent group items containing the selected item are automatically selected
- If a selected item contains the REDEFINES clause, it must also be specified in the RedefineSelection element for this message specification
- When any group item is selected all its children are automatically selected (any items that contain the REDEFINES clause are also subject to the preceding rule)
- When an ODO (variable size repeating) item is selected, its ODO object (count variable) must also be selected. Otherwise the results produced by the generated artifacts are undefined
- ItemSelection elements can appear in any order within ItemSelectionArray
- The value of the ItemSelection attribute must specify the data item name prefixed by dot-separated parent group names.
- The values of the ItemSelection attributes are not case-sensitive. For example, specifying *itemName="FoO.bAr"* is equivalent to specifying *itemName="fOo.BaR"*
- If none of the selected items appear in the data structure, the results produced by the generated artifacts are undefined.

Contained by

ItemSelectionArray

Contains

None

Attributes

Name	Required?	Values	Default	Description
name	itemName	Yes		Specifies the name of the data item to be selected.
value			none	

The value of the attribute must specify the item name prefixed a dot-separated parent names as shown in the example.

Example

For the following COBOL data structure

```
01 INPUT-MSG.
    02 IN-LL           PICTURE S9(3) COMP.
    02 IN-ZZ           PICTURE S9(3) COMP.
    02 IN-TRCD         PICTURE X(10).
    02 IN-CMD          PICTURE X(8).
    02 IN-NAME1        PICTURE X(10).
    02 redParent.
       03 redefd.
          04 IN-NAME2   PICTURE X(10).
       03 redefd2 redefines redefd.
          04 IN-NAME2R  PICTURE X(10).
    02 IN-EXTN         PICTURE X(10).
    02 IN-EXTNR redefines in-extn PICTURE X(10).
    02 IN-ZIP          PICTURE X(7).
```

you can specify the following item selections:

```
<ItemSelection>
   <ItemSelection itemName="input-msg.redParent"/>
   <ItemSelection itemName="input-msg.in-extn"/>
   <ItemSelection itemName="input-msg.in-zip"/>
   <ItemSelection itemName="input-msg.in-ll"/>
</ItemSelectionArray>
```

Note that if you want to specify *redefd2* redefinition in the *redParent* group, you will need to specify the following RedefinesArray:

```
<RedefinesArray>
   <RedefineSelection redefine="input-msg.redParent.redefd" useRedefinition="input-msg.redParent.redefd2"/>
</RedefinesArray>
```

otherwise *redefd* items will be selected by default.

ItemSelectionArray

Use this element as the container for individual data item selections in the COBOL structure.

If the this element is not specified the default element selection rules apply as follows:

- If nativeTypeName of the message specification is not given, the first available LEVEL 01 structure and all its children are automatically selected according to any rules specified in the RedefineSelection section and ItemSelection section.

- If nativeTypeName of the message specification is given, it is assumed to be a LEVEL 01 structure and all its children are automatically selected according to any rules specified in the RedefineSelection section and ItemSelection section.
- If the RedefineSelectionArray is given, the items from that array will override the default REDEFINES behavior (See RedefineSelection and ItemSelection section for more information)

Contained by

InputMessage, InputOutputMessage, OutputMessage

Contains

ItemSelection

Attributes

None

Example

```
<ItemSelectionArray>
   <ItemSelection itemName="input-msg.redParent"/>
   <ItemSelection itemName="input-msg.in-extn"/>
   <ItemSelection itemName="input-msg.in-zip"/>
   <ItemSelection itemName="input-msg.in-ll"/>
</ItemSelectionArray>
```

Operation

Use this element to specify the type of operation and as the container for operation properties. Note that only one operation is allowed per service. Specifying more than one operation may cause an invalid WSDL to be generated.

Contained by

EISService

Contains

OperationPropertyArray, InputOutputMessage, InputMessage, OutputMessage, XseSpec

Attributes

Name	Required?	Values	Default	Description
name	Yes	See Description	The name attribute for the EISService element concatenated with ″Operation″	Specifies the name of the operation in the WSDL file.
type	Yes	REQUEST_RESPONSE | SOLICIT_RESPONSE | ONE_WAY | NOTIFICATION	REQUEST_RESPONSE	Specifies whether to generate the converter set (inbound, outbound, converters, and driver).

REQUEST_RESPONSE and SOLICIT_RESPONSE operations cause an inbound and an outbound converter, a driver (if the generateConverters option is in effect), and an inbound and outbound schema (if the generateSeparateXSD option is in effect) to be generated. A NOTIFICATION operation causes an outbound converter, a driver (if the generateConverters option is in effect), and an outbound schema (if the generateSeparateXSD option is in effect) to be generated. A ONE_WAY operation causes an inbound converter, a driver (if the generateConverters option is in effect), and an inbound schema (if the generateSeparateXSD option is in effect) to be generated.

Example

```
<Operation name="getCustomerInfo" type="REQUEST_RESPONSE">
        <OperationPropertyArray>
                <OperationProperty name="soapOpStyle" value="document" />
                <OperationProperty name="soapBindingStyle" value="document" />
                <OperationProperty name="soapBodyUse" value="literal" />
        </OperationPropertyArray>
        <InputOutputMessage name="CustomerDetails" importDirectory="." importFile="DFH0ACTD.cbl"
                nativeTypeName="DFHCOMMAREA">
                <RedefinesArray>
                        <RedefineSelection redefine="name.info" useRedefinition="name.last-name"/>
                        <RedefineSelection redefine="address.zip-code" useRedefinition="province"/>
                </RedefinesArray>
        </InputOutputMessage>
        <XseSpec>
                <DriverSpec fileName=""DFH0CSTDD.cbl" driverType="IMS SOAP Gateway" programName="XCNVD" businessPgmName=
                <ConverterSpecIn fileName="DFH0CSTDI.cbl" overwrite="true"
                        programName="XCNVI"/>
                <ConverterSpecOut fileName="DFH0CSTDO.cbl" overwrite="true"
                        programName="XCNVO"/>
                <XsdSpecIn fileName="DFH0CSTDI.xsd overwrite="true"
                        targetNamespace="http://www.DFH0CSTDI.com/schemas/DFH0CSTDIInterface"
                        xsdNamespace="http://www.w3.org/2001/XMLSchema"
                        localNamespace="http://www.DFH0CSTDI.com/schemas/DFH0CSTDIInterface"
                        xsdPrefix="cbl"
                        xsdElemName="dfhcommarea"/>
                <XsdSpecOut fileName="DFH0CSTDO.xsd" overwrite="true"
                        targetNamespace="http://www.DFH0CSTDO.com/schemas/DFH0CSTDOInterface"
                        xsdNamespace="http://www.w3.org/2001/XMLSchema"
                        localNamespace="http://www.DFH0CSTDO.com/schemas/DFH0CSTDOInterface"
                        xsdPrefix="cbl"
                        xsdElemName="dfhcommarea"/>
        </XseSpec>
</Operation>
```

OperationProperty

Use this element to specify the properties for an operation.

Contained by

OperationPropertyArray

Contains

None

Attributes

Name	Values	Required?	Default	Description
name	soapOpStyle	No		Specifies the SOAP operation style. The value must match the soapBindingStyle in ServiceSpecification.xml.
value	document | rpc		document	
name	soapBodyUse	No		Specifies use of encoding style for SOAP body.
value	literal | encoded		literal	

Name	Values	Required?	Default	Description
name	soapAction	No		Specifies the URI for the optional SOAP action.
value	URI			none

Note: If you are generating artifacts for IMS SOAP Gateway you must specify soapAction property, for example ″urn:IMSPHBK″. It should match the soapAction value in the CorrelatorSpec property. If you don't specify this property or if it does not match the CorrelatorSpec, it will cause a runtime error.

Example

```
<OperationPropertyArray>
                <OperationProperty name="soapOpStyle" value="document" />
                <OperationProperty name="soapBodyUse" value="literal" />
                <OperationProperty name="soapAction" value="urn:myaction" />
</OperationPropertyArray>
```

OperationPropertyArray

Use this element as the container for operation properties.

Contained by

Operation

Contains

OperationProperty

Attributes

None

Example

```
<OperationPropertyArray>
        <OperationProperty name="soapOpStyle" value="document" />
        <OperationProperty name="soapBindingStyle" value="document" />
        <OperationProperty name="soapBodyUse" value="literal" />
</OperationPropertyArray>
```

OutputMessage

Use this element to define messages for NOTIFICATION operations or REQUEST_RESPONSE (SOLICIT_RESPONSE) operations if the input and output messages are of different types. Note that for NOTIFICATION operations, you are allowed to specify only the OutputMessage. InputMessage is not allowed and will cause unpredictable results during the generation process.

Contained by

Operation

Contains

ItemSelectionArray

Attributes

Name	Required?	Values	Default	Description
name	No	See Description	esvc	Specifies a name for the messages for the WSDL file.
importDirectory	No	See Description	The input directory	Specifies the directory for the source file.
importFile	Yes (unless the -s command line option is specified)	See Description	The name provided in the xsebatch command-line parameter -s.	Specifies the file name that contains the data definition to be used in creating the web service operation message types. **Note:** Only COBOL data definitions are supported and are subject to the restrictions that are specified in WebSphere Developer documentation.
nativeTypeName	No	See Description	For COBOL, the name of the first available level 01 data item name.	Specifies the name of the data type that is to be imported from the importFile, such as DFHCOMMAREA for a CICS COBOL application. An error message is generated on the console if a parse of the importFile does not identify the nativeTypeName as a valid data type.

Example

```
<OutputMessage name="PhoneBookReply" importDirectory="." importFile="Ex01z.cbl" nativeTypeName="output-msg">
   <RedefinesArray>
      <RedefineSelection redefine="output-msg.redParent.redefd" useRedefinition="output-msg.redParent.redefd2"/>
   </RedefinesArray>
   <ItemSelectionArray>
      <ItemSelection itemName="output-msg.redParent"/>
      <ItemSelection itemName="output-msg.in-extn"/>
      <ItemSelection itemName="output-msg.in-zip"/>
      <ItemSelection itemName="output-msg.in-ll"/>
   </ItemSelectionArray>
</OutputMessage>
```

RedefinesArray

Use this element as the container for REDEFINE item selections for COBOL.

Note: The default behavior for redefinitions when there are REDEFINES in the data but the RedefinesArray element is omitted is to use the original definition of the items.

Contained by

InputMessage, InputOutputMessage, OutputMessage

Contains

RedefineSelection

Attributes

None

Example

```
<RedefinesArray>
 <RedefineSelection redefine="name.info" useRedefinition="name.last-name"/>
 <RedefineSelection redefine="address.zip-code" useRedefinition="province"/>
</RedefinesArray>
```

RedefineSelection

Use this element to select redefinitions for an item.

Contained by

RedefinesArray

Contains

None

Attributes

If necessary for uniqueness, qualify the data item names using the period-separated notation. For example, use name info from the following data structure.

```
1 name.
  2 info pic x(100).
  2 last-name redefines info pic x(100).
```

Name	Values	Required?	Default	Description
name	redefine	No		Specifies the name of the data item to be redefined. **Note:** Redefinitions of redefined items are not supported.
value			none	
name	useRedefinition	No		Specifies the name of the data item to be used.
value			none	

Example

```
<RedefinesArray>
        <RedefineSelection redefine="name.info" useRedefinition="name.last-name"/>
        <RedefineSelection redefine="address.zip-code" useRedefinition="province"/>
</RedefinesArray>
```

ServiceProperty

Use this element to specify the properties for a service.

Contained by

ServicePropertyArray

Contains

None

Attributes

Name	Values	Required?	Default	Description
name	portName	No		Specifies the name of the port element in the WSDL file.
value	QNAME		The name attribute for the EISService element (or if it is not specified, the value of -w parameter from the xsebatch command line invocation) concatenated with the upper-cased value of the type attribute of the EISService element.	
name	portType	No		Specifies the name of the port type element in the WSDL file.
value	QNAME		The name attribute for the EISService element (or if it is not specified, the value of -w parameter from the xsebatch command line invocation) concatenated with the lower-cased value of the type attribute of the EISService element.	
name	bindingName	No		Specifies the name of the binding element in the WSDL file.
value	QNAME		The name attribute for the EISService element (or if it is not specified, the value of -w parameter from the xsebatch command line invocation) concatenated with the upper-cased value of the type attribute of the EISService element concatenated with the word ″Binding″.	
name	soapTransport	No		Specifies the value for the transport attribute of the binding element in the WSDL file.
value	URI		http://schemas.xmlsoap.org/soap/http	
name	soapBindingStyle	No		Specifies the value of the style attribute of binding element in the WSDL file. The value must match the soapOpStyle.in Operation properties.
value	document | rpc		document	

Name	Values	Required?	Default	Description
name	serviceName	No		Specifies the value of the name attribute of service element in the WSDL file.
value	QNAME		The name attribute for the EISService element (or if it is not specified, the value of -w parameter from the xsebatch command line invocation) concatenated with the upper-cased value of the type attribute of the EISService element concatenated with the word ″Service″.	

Example

```
<ServicePropertyArray>
        <ServiceProperty name="bindingName" value="myBinding" />
        <ServiceProperty name="bindingType" value="SOAP" />
        <ServiceProperty name="soapTransport" value="http://schemas.xmlsoap.org/soap/http" />
        <ServiceProperty name="soapBindingStyle" value="document" />
</ServicePropertyArray>
```

ServicePropertyArray

Use this element as a container for the properties for a service.

Contained by

EISService

Contains

ServiceProperty

Attributes

None

Example

```
<ServicePropertyArray>
 <ServiceProperty name="bindingName" value="myBinding" />
 <ServiceProperty name="bindingType" value="SOAP" />
 <ServiceProperty name="soapTransport" value="http://schemas.xmlsoap.org/soap/http" />
 <ServiceProperty name="soapBindingStyle" value="document" />
</ServicePropertyArray>
```

WSBindSpec

Use this element of the ServiceSpecification.xml document to specify the generation options for the Vendor WSBind file.

The Vendor WSBind file is used to install a new web service under CICS Transaction Server version 3.1 (and later). In order to generate a WSBind file, the **driverType** attribute in the DriverSpec element should be set to ’WEB_SERVICES_CICS’.

Contained by

XseSpec

Contains

None

Attributes

Name	Required?	Values	Default	Description
fileName	No	See Description	Data source file name concatenated with ″.wsbind″	Specifies the name of the output file.
overwrite	No	true ǀ false	true	Specifies whether to overwrite the output file if it exists.
pgmint	No	2 ǀ 1	2 (DFHCOMMAREA)	Specify whether the CICS application program communicates via a DFHCOMMAREA (2) or CONTAINER (1).
contid	No (Yes, if pgmint is set to CONTAINER)	See Description	None	If the CICS application program (specified in businessPgmName attribute of the DriverSpec element) communicates via a CONTAINER, specify the name of the CONTAINER expected by program.
uri	No	See Description	See Description	Desired local URI to for the web service, for example, ″/exampleApp/InquireSingle″. Note: this is different that the location of the web service for example, http://server:port[local URI]. If you do not specify this property it will have to be defined at install time during manual creation of the web service resource definitions in CICS.
pipeline	No	See Description	See Description	The name of the CICS PIPELINE resource under which this web service should be installed. If you do not specify this property it will have to be defined at install time during manual creation of the web service resource definitions in CICS.
wsdlloc	No	See Description	See Description	Full HFS path to the WSDL file that CICS should use for validation of SOAP request and response messages, for example, ″/u/svltest/pickup/inquireSingle.wsdl″. If you do not specify this property it can be defined at install time during manual creation of the web service resource definitions in CICS.

Example

```
<WSBindSpec fileName="DFH0XCMN.wsbind" overwrite="true" pgmint="2" uri="/exampleApp/InquireSingle " pipeline="PIPE8070" 
```

XsdSpecIn

Use this element of the ServiceProperties.xml document to specify the generation options for the XML schema that corresponds to the input data structure.

Contained by

XseSpec

Contains

None

Attributes

Name	Required?	Values	Default	Description
fileName	No	See Description	Data source file name concatenated with ″I″	Specifies the name of the output file.
overwrite	No	true | false	true	Specifies whether to overwrite the output file if it exists.
localNamespace	No	See Description	http://www.w3.org/2001/XMLSchema **Note:** Inbound namespaces have no effect on the code generated in the inbound converter.	Specifies the local namespace.
targetNamespace	No	See Description	For a data source file name foo: http://www.fooI.com/schemas/fooIInterface	Specifies the target namespace.
xsdElemName	No	See Description	Value of the nativeTypeName attribute in the message specification	Specifies the global element name for the schema.
xsdNamespace	No	See Description	Value of the nativeTypeName attribute in the message specification	Specifies the xsd namespace.
xsdPrefix	No	See Description	cbl	Specifies the xsd namespace prefix.

Example

```
<XsdSpecIn fileName="DFH0CSTDI.xsd" overwrite="true"
        targetNamespace="http://www.DFH0CSTDI.com/schemas/DFH0CSTDIInterface"
        xsdNamespace="http://www.w3.org/2001/XMLSchema"
        localNamespace="http://www.DFH0CSTDI.com/schemas/DFH0CSTDIInterface"
        xsdPrefix="cbl"
        xsdElemName="dfhcommarea"/>
```

XsdSpecOut

Use this element of the ServiceProperties.xml document to specify the generation options for the XML schema that corresponds to the output data structure.

Contained by

XseSpec

Contains

None

Attributes

Name	Required?	Values	Default	Description
fileName	No	See Description	Data source file name concatenated with ″O″	Specifies the name of the output file.
overwrite	No	true | false	true	Specifies whether to overwrite the output file if it exists.
localNamespace	No	See Description	http://www.w3.org/2001/XMLSchema	Specifies the local namespace.
targetNamespace	No	See Description	For a data source file name foo: http://www.fooO.com/schemas/fooIInterface	Specifies the target namespace.
xsdElemName	No	See Description	Value of the nativeTypeName attribute in the message specification	Specifies the global element name for the schema.
xsdNamespace	No	See Description	Value of the nativeTypeName attribute in the message specification	Specifies the xsd namespace.
xsdPrefix	No	See Description	cbl	Specifies the xsd namespace prefix.

Example

```
<XsdSpecOut fileName="DFH0CSTDO.xsd" overwrite="true"
        targetNamespace="http://www.DFH0CSTDO.com/schemas/DFH0CSTDOInterface"
        xsdNamespace=http://www.w3.org/2001/XMLSchema
        localNamespace=http://www.DFH0CSTDO.com/schemas/DFH0CSTDOInterface
        xsdPrefix="cbl"
        xsdElemName="dfhcommarea"/>
```

XseSpec

Use this element as the container for specifying generation options for the set of converters, driver, and XSD files.

Contained by

Operation

Contains

ConverterSpecIn, ConverterSpecOut, XsdSpecIn, XsdSpecOut, CorrelatorSpec, DriverSpec

Attributes

None

Example

```
<XseSpec>
        <DriverSpec fileName=""DFH0CSTDD.cbl" driverType="IMS SOAP Gateway" programName="XCNVD" businessPgmName="Ex01" /
        <ConverterSpecIn fileName="DFH0CSTDI.cbl" overwrite="true" programName="XCNVI"/>
        <ConverterSpecOut fileName="DFH0CSTDO.cbl" overwrite="true" programName="XCNVO"/>
        <XsdSpecIn fileName="DFH0CSTDI.xsd" overwrite="true"
                targetNamespace="http://www.DFH0CSTDI.com/schemas/DFH0CSTDIInterface"
                xsdNamespace="http://www.w3.org/2001/XMLSchema"
                localNamespace="http://www.DFH0CSTDI.com/schemas/DFH0CSTDIInterface"
                xsdPrefix="cbl"
                xsdElemName="dfhcommarea" />
        <XsdSpecOut fileName="DFH0CSTDO.xsd" overwrite="true"
                targetNamespace="http://www.DFH0CSTDO.com/schemas/DFH0CSTDOInterface"
                xsdNamespace="http://www.w3.org/2001/XMLSchema"
                localNamespace="http://www.DFH0CSTDO.com/schemas/DFH0CSTDOInterface"
                xsdPrefix="cbl"
                xsdElemName="dfhcommarea"       />
</XseSpec>
```

ServiceSpecification.xml sample

```
<EISProject name="CICSSample">
        <!-- Use the ImportPropertyArray to override values from PlatformProperties.xml -->
        <ImportPropertyArray type="Cobol">
                <ImportProperty name="com.ibm.etools.cobol.COBOL_TRUNC" value="STD" />
                <ImportProperty name="com.ibm.etools.cobol.COBOL_NSYMBOL" value="DBCS" />
                <ImportProperty name="com.ibm.etools.cobol.COBOL_QUOTE" value="DOUBLE" />
        </ImportPropertyArray>
    <CodegenPropertyArray type="Cobol">
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_PROG_NAME" value="XCNV"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_AUTH_NAME" value="WSED"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_IN_CP_LIST" value="1140"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_CP_LIST" value="1140"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_OUT_CP_LIST" value="1140"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_DEC_COMMA" value=" false"/>
      <CodegenProperty name="com.ibm.etools.xmlent.ui.GEN_FLAT_GEN" value=" false"/>
    </CodegenPropertyArray>
        <EISService name="CustomerInfo" type="CICS" targetNamespace="http://cics.sample" generateConverters="true" generate
                <!-- Use the ConnectionPropertyArray to override values from PlatformProperties.xml -->
                <ConnectionPropertyArray>

                        <ConnectionProperty name="connectionURI" value="http://winmvsn0.cpit.hursley.ibm.com:8888/CICS/XMLS
                </ConnectionPropertyArray>
                <ServicePropertyArray>

                                <ServiceProperty name="bindingName" value="myBinding" />
                                <ServiceProperty name="soapTransport" value="http://schemas.xmlsoap.org/soap/http" />
                                <ServiceProperty name="soapBindingStyle" value="document" />
                </ServicePropertyArray>
                <Operation name="getCustomerInfo" type="REQUEST_RESPONSE">
                        <OperationPropertyArray>

                                <OperationProperty name="soapOpStyle" value="document" />
                                <OperationProperty name="soapBodyUse" value="literal" />
                                <OperationProperty name="soapAction" value="http://example.com/getCustomerInfo" />
                        </OperationPropertyArray>
                        <InputOutputMessage name="CustomerDetails" importDirectory="." importFile="DFH0ACTD.cbl" nativeType
                          <RedefinesArray>
                             <RedefineSelection redefine="name.info" useRedefinition="name.last-name"/>
                             <RedefineSelection redefine="address.zip-code" useRedefinition="province"/>
                          </RedefinesArray>
                        </InputOutputMessage>
                        <XseSpec>
                          <DriverSpec fileName=""DFH0CSTDD.cbl" driverType="IMS SOAP Gateway" programName="XCNVD" businessP
                          <ConverterSpecIn fileName="DFH0CSTDI.cbl" overwrite="true" programName="XCNVI"/>
                          <ConverterSpecOut fileName="DFH0CSTDO.cbl" overwrite="true" programName="XCNVO"/>
                          <XsdSpecIn fileName="DFH0CSTDI.xsd"
                                  overwrite="true"
                                  targetNamespace="http://www.DFH0CSTDI.com/schemas/DFH0CSTDIInterface"
                                  xsdNamespace="http://www.w3.org/2001/XMLSchema"
                                  localNamespace="http://www.DFH0CSTDI.com/schemas/DFH0CSTDIInterface"
```

```
                                   xsdPrefix="cbl"
                                   xsdElemName="dfhcommarea"
                                   />
                        <XsdSpecOut fileName="DFH0CSTDO.xsd"
                                   overwrite="true"
                                   targetNamespace="http://www.DFH0CSTDO.com/schemas/DFH0CSTDOInterface"
                                   xsdNamespace="http://www.w3.org/2001/XMLSchema"
                                   localNamespace="http://www.DFH0CSTDO.com/schemas/DFH0CSTDOInterface"
                                   xsdPrefix="cbl"
                                   xsdElemName="dfhcommarea"
                                   />
                      </XseSpec>
              </Operation>
      </EISService>
</EISProject>
```

Related references

ServiceSpecification.xml

Schema for ServiceSpecification.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="ConnectionProperty">
        <xsd:complexType>
            <xsd:attribute name="name" type="xsd:string" use="required"/>
            <xsd:attribute name="value" type="xsd:string" use="required"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="ConnectionPropertyArray">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element maxOccurs="unbounded" minOccurs="1" ref="ConnectionProperty"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="ConverterSpecIn">
        <xsd:complexType>
            <xsd:attribute name="fileName" type="xsd:string" use="optional"/>
            <xsd:attribute name="overwrite" type="xsd:boolean" use="optional"/>
            <xsd:attribute name="programName" type="xsd:string" use="optional"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="ConverterSpecOut">
        <xsd:complexType>
            <xsd:attribute name="fileName" type="xsd:string" use="optional"/>
            <xsd:attribute name="overwrite" type="xsd:boolean" use="optional"/>
            <xsd:attribute name="programName" type="xsd:string" use="optional"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="CorrelatorSpec">
        <xsd:complexType>
            <xsd:attribute name="fileName" type="xsd:string" use="optional"/>
            <xsd:attribute name="overwrite" type="xsd:boolean" use="optional"/>
            <xsd:attribute name="soapAction" type="xsd:string" use="optional"/>
                        <xsd:attribute name="adapterType" type="xsd:string" use="optional"/>
                        <xsd:attribute name="connectionBundleName" use="optional">
                                <xsd:simpleType>
                                        <xsd:restriction base="xsd:string">
                                                <xsd:maxLength value="20"/>
                                        </xsd:restriction>
                                </xsd:simpleType>
                        </xsd:attribute>
            <xsd:attribute name="socketTimeout" type="xsd:nonNegativeInteger" use="optional"/>
                        <xsd:attribute name="executionTimeout" use="optional">
```

```
                                <xsd:simpleType>
                                        <xsd:restriction base="xsd:nonNegativeInteger">
                                                <xsd:maxInclusive value="3600000"/>
                                        </xsd:restriction>
                                </xsd:simpleType>
                        </xsd:attribute>
                        <xsd:attribute name="ltermName" use="optional">
                                <xsd:simpleType>
                                        <xsd:restriction base="xsd:string">
                                                <xsd:maxLength value="8"/>
                                        </xsd:restriction>
                                </xsd:simpleType>
                        </xsd:attribute>
    </xsd:complexType>
</xsd:element>
<xsd:element name="WSBindSpec">
    <xsd:complexType>
        <xsd:attribute name="fileName" type="xsd:string" use="optional"/>
        <xsd:attribute name="overwrite" type="xsd:boolean" use="optional"/>
        <xsd:attribute name="pgmint" type="xsd:int" use="optional"/>
        <xsd:attribute name="contid" type="xsd:string" use="optional"/>
        <xsd:attribute name="uri" type="xsd:string" use="optional"/>
        <xsd:attribute name="pipeline" type="xsd:string" use="optional"/>
        <xsd:attribute name="wsdlloc" type="xsd:string" use="optional"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name="DriverSpec">
    <xsd:complexType>
            <xsd:attribute name="driverType" type="xsd:string" use="optional"/>
        <xsd:attribute name="fileName" type="xsd:string" use="optional"/>
        <xsd:attribute name="overwrite" type="xsd:boolean" use="optional"/>
        <xsd:attribute name="programName" type="xsd:string" use="optional"/>
                        <xsd:attribute name="businessPgmName" type="xsd:string" use="optional"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name="EISProject">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="0" ref="ImportPropertyArray"/>
            <xsd:element maxOccurs="1" minOccurs="0" ref="CodegenPropertyArray"/>
            <xsd:element maxOccurs="unbounded" minOccurs="1" ref="EISService"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name="EISService">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="0" ref="ServicePropertyArray"/>
            <xsd:element maxOccurs="1" minOccurs="0" ref="ConnectionPropertyArray"/>
            <xsd:element ref="Operation"/>
        </xsd:sequence>
        <xsd:attribute name="generateConverters" type="xsd:boolean" use="optional"/>
        <xsd:attribute name="generateSeparateXSD" type="xsd:boolean" use="optional"/>
        <xsd:attribute name="generateWSDL" type="xsd:boolean" use="optional"/>
        <xsd:attribute name="name" type="xsd:string" use="optional"/>
        <xsd:attribute name="targetNamespace" type="xsd:string" use="optional"/>
        <xsd:attribute name="type" type="xsd:string" use="optional"/>
        <xsd:attribute name="targetFilesURI" type="xsd:string" use="optional"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name="ImportProperty">
    <xsd:complexType>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="value" type="xsd:string" use="required"/>
    </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="ImportPropertyArray">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" ref="ImportProperty"/>
        </xsd:sequence>
        <xsd:attribute name="type" type="xsd:string" use="optional"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name="ServiceProperty">
    <xsd:complexType>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="value" type="xsd:string" use="required"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name="ServicePropertyArray">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" ref="ServiceProperty"/>
        </xsd:sequence>
        <xsd:attribute name="type" type="xsd:string" use="optional"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name="CodegenProperty">
    <xsd:complexType>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="value" type="xsd:string" use="required"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name="CodegenPropertyArray">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" ref="CodegenProperty"/>
        </xsd:sequence>
        <xsd:attribute name="type" type="xsd:string" use="optional"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name="InputMessage">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="0" ref="RedefinesArray"/>
            <xsd:element maxOccurs="1" minOccurs="0" ref="ItemSelectionArray"/>
        </xsd:sequence>
        <xsd:attribute name="importDirectory" type="xsd:string" use="optional"/>
        <xsd:attribute name="importFile" type="xsd:string" use="required"/>
        <xsd:attribute name="name" type="xsd:string" use="optional"/>
        <xsd:attribute name="nativeTypeName" type="xsd:string" use="optional"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name="InputOutputMessage">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="0" ref="RedefinesArray"/>
            <xsd:element maxOccurs="1" minOccurs="0" ref="ItemSelectionArray"/>
        </xsd:sequence>
        <xsd:attribute name="importDirectory" type="xsd:string" use="optional"/>
        <xsd:attribute name="importFile" type="xsd:string" use="required"/>
        <xsd:attribute name="name" type="xsd:string" use="optional"/>
        <xsd:attribute name="nativeTypeName" type="xsd:string" use="optional"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name="ItemSelection">
    <xsd:complexType>
        <xsd:attribute name="itemName" type="xsd:string" use="required"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name="ItemSelectionArray">
    <xsd:complexType>
```

```
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" ref="ItemSelection"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="Operation">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="0" ref="OperationPropertyArray"/>
            <xsd:choice>
                <xsd:sequence>
                    <xsd:element ref="InputOutputMessage"/>
                    <xsd:element maxOccurs="1" minOccurs="0" ref="XseSpec"/>
                </xsd:sequence>
                <xsd:sequence>
                    <xsd:element ref="InputMessage"/>
                    <xsd:element maxOccurs="1" minOccurs="0" ref="XseSpec"/>
                </xsd:sequence>
                <xsd:sequence>
                    <xsd:element ref="OutputMessage"/>
                    <xsd:element maxOccurs="1" minOccurs="0" ref="XseSpec"/>
                </xsd:sequence>
            </xsd:choice>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="type" type="xsd:string" use="required"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name="OperationProperty">
    <xsd:complexType>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="value" type="xsd:string" use="required"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name="OperationPropertyArray">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" ref="OperationProperty"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="OutputMessage">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="0" ref="RedefinesArray"/>
            <xsd:element maxOccurs="1" minOccurs="0" ref="ItemSelectionArray"/>
        </xsd:sequence>
        <xsd:attribute name="importDirectory" type="xsd:string" use="optional"/>
        <xsd:attribute name="importFile" type="xsd:string" use="required"/>
        <xsd:attribute name="name" type="xsd:string" use="optional"/>
        <xsd:attribute name="nativeTypeName" type="xsd:string" use="optional"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name="RedefineSelection">
    <xsd:complexType>
        <xsd:attribute name="redefine" type="xsd:string" use="required"/>
        <xsd:attribute name="useRedefinition" type="xsd:string" use="required"/>
    </xsd:complexType>
</xsd:element>
<xsd:element name="RedefinesArray">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" ref="RedefineSelection"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="XsdSpecIn">
```

```
        <xsd:complexType>
            <xsd:attribute name="fileName" type="xsd:string" use="optional"/>
            <xsd:attribute name="localNamespace" type="xsd:string" use="optional"/>
            <xsd:attribute name="overwrite" type="xsd:boolean" use="optional"/>
            <xsd:attribute name="targetNamespace" type="xsd:string" use="optional"/>
            <xsd:attribute name="xsdElemName" type="xsd:string" use="optional"/>
            <xsd:attribute name="xsdNamespace" type="xsd:string" use="optional"/>
            <xsd:attribute name="xsdPrefix" type="xsd:string" use="optional"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="XsdSpecOut">
        <xsd:complexType>
            <xsd:attribute name="fileName" type="xsd:string" use="optional"/>
            <xsd:attribute name="localNamespace" type="xsd:string" use="optional"/>
            <xsd:attribute name="overwrite" type="xsd:boolean" use="optional"/>
            <xsd:attribute name="targetNamespace" type="xsd:string" use="optional"/>
            <xsd:attribute name="xsdElemName" type="xsd:string" use="optional"/>
            <xsd:attribute name="xsdNamespace" type="xsd:string" use="optional"/>
            <xsd:attribute name="xsdPrefix" type="xsd:string" use="optional"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="XseSpec">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="DriverSpec"/>
                <xsd:element maxOccurs="1" minOccurs="0" ref="ConverterSpecIn"/>
                <xsd:element maxOccurs="1" minOccurs="0" ref="ConverterSpecOut"/>
                <xsd:element maxOccurs="1" minOccurs="0" ref="XsdSpecIn"/>
                <xsd:element maxOccurs="1" minOccurs="0" ref="XsdSpecOut"/>
                <xsd:element maxOccurs="1" minOccurs="0" ref="CorrelatorSpec"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Related references

ServiceSpecification.xml

Chapter 8. Mapping reference

The following mapping reference topics are included in this section:
- Mapping reference information
- COBOL language types
- XML and COBOL type compatibility
- XML and COBOL structure compatibility
- XML types derived from COBOL
- Isomorphic and non-isomorphic element mapping

Related concepts

Introduction to XML Services for the Enterprise
XML to COBOL mapping tools

Related tasks

Creating a web service interface with the Web Services Enablement wizard
Setting preferences for XML Services for the Enterprise
Mapping XML to COBOL
Creating a mapping session file
Editing a mapping session file

Mapping reference information

Table 1. The values of the fundamental facets for each •built-in• XML Schema datatype.

	Datatype	**ordered**	**bounded**	**cardinality**	**numeric**
primitive	string	false	false	countably infinite	false
	boolean	false	false	finite	false
	float	total	true	finite	true
	double	total	true	finite	true
	decimal	total	false	countably infinite	true
	duration	partial	false	countably infinite	false
	dateTime	partial	false	countably infinite	false
	time	partial	false	countably infinite	false
	date	partial	false	countably infinite	false
	gYearMonth	partial	false	countably infinite	false
	gYear	partial	false	countably infinite	false
	gMonthDay	partial	false	countably infinite	false
	gDay	partial	false	countably infinite	false
	gMonth	partial	false	countably infinite	false
	hexBinary	false	false	countably infinite	false
	base64Binary	false	false	countably infinite	false
	anyURI	false	false	countably infinite	false
	QName	false	false	countably infinite	false
	NOTATION	false	false	countably infinite	false

Table 1. The values of the fundamental facets for each •built-in• XML Schema datatype. (continued)

	Datatype	ordered	bounded	cardinality	numeric
derived	normalizedString	false	false	countably infinite	false
	token	false	false	countably infinite	false
	language	false	false	countably infinite	false
	IDREFS	false	false	countably infinite	false
	ENTITIES	false	false	countably infinite	false
	NMTOKEN	false	false	countably infinite	false
	NMTOKENS	false	false	countably infinite	false
	Name	false	false	countably infinite	false
	NCName	false	false	countably infinite	false
	ID	false	false	countably infinite	false
	IDREF	false	false	countably infinite	false
	ENTITY	false	false	countably infinite	false
	integer	total	false	countably infinite	true
	nonPositiveInteger	total	false	countably infinite	true
	negativeInteger	total	false	countably infinite	true
	long	total	true	finite	true
	int	total	true	finite	true
	short	total	true	finite	true
	byte	total	true	finite	true
	nonNegativeInteger	total	false	countably infinite	true
	unsignedLong	total	true	finite	true
	unsignedInt	total	true	finite	true
	unsignedShort	total	true	finite	true
	unsignedByte	total	true	finite	true
	positiveInteger	total	false	countably infinite	true

COBOL language types

The COBOL language types are described in the Enterprise COBOL Language reference manual and are comprised of the following classes and categories:

Table 2. COBOL Language type classes and categories

Level of item	Class	Category
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
		Internal floating-point
		External floating-point
	Alphanumeric	Numeric-edited
		Alphanumeric-edited
		Alphanumeric
		DBCS
	National	National
Group	Alphanumeric	Alphabetic
		Numeric
		Internal floating-point
		External floating-point
		Numeric-edited
		Alphanumeric-edited
		Alphanumeric
		DBCS
		National

XML and COBOL type compatibility

In order for your mapping selection to be valid, the COBOL elementary items and XML elements you are trying to match must have similar (compatible) types. In other words, the COBOL data class or category of the mapped item should match the data category of the XML elements. For example, a COBOL numeric item should only be matched with an XML element which could be described by a built-in numeric XML schema type. The following table describes relationship between COBOL data classes and categories and the XML Schema types. Note that you cannot map COBOL data items that do not have category and class (for example, PROCEDURE POINTER).

Table 3. Mapping of built-in XML types to COBOL data classes and categories

	XML Datatype	COBOL data Class or Category	Default Inbound Conversion	Default Outbound Conversion[1]
primitive	string	Alphabetic, Alphanumeric, National, DBCS, Numeric, Numeric edited[2]	MOVE, NUMVAL	MOVE
	boolean	Alphabetic, Numeric	MOVE, NUMVAL	MOVE
	float	Numeric, Numeric-edited	MOVE, NUMVAL	MOVE
	double	Numeric, Numeric-edited	MOVE, NUMVAL	MOVE
	decimal	Numeric, Numeric-edited	MOVE, NUMVAL	MOVE
	duration	Alphanumeric, National	N/A	N/A
	dateTime	Alphanumeric, National	N/A	N/A
	time	Alphanumeric, National	N/A	N/A
	date	Alphanumeric, National	N/A	N/A
	gYearMonth	Alphanumeric, National	N/A	N/A
	gYear	Alphanumeric, National	N/A	N/A
	gMonthDay	Alphanumeric, National	N/A	N/A
	gDay	Alphanumeric, National	N/A	N/A
	gMonth	Alphanumeric, National	N/A	N/A
	hexBinary	Alphanumeric, National	N/A	N/A
	base64Binary	Alphanumeric, National	N/A	N/A
	anyURI	Alphanumeric, National	N/A	N/A
	QName	Alphanumeric, National	N/A	N/A
	NOTATION	Alphanumeric, National	N/A	N/A

Table 3. Mapping of built-in XML types to COBOL data classes and categories (continued)

	XML Datatype	COBOL data Class or Category	Default Inbound Conversion	Default Outbound Conversion[1]
derived	normalizedString	Alphabetic, Alphanumeric, National	N/A	N/A
	token	Alphabetic, Alphanumeric, National	N/A	N/A
	language	Alphabetic, Alphanumeric, National	N/A	N/A
	IDREFS	Alphabetic, Alphanumeric, National	N/A	N/A
	ENTITIES	Alphabetic, Alphanumeric, National	N/A	N/A
	NMTOKEN	Alphabetic, Alphanumeric, National	N/A	N/A
	NMTOKENS	Alphabetic, Alphanumeric, National	N/A	N/A
	Name	Alphabetic, Alphanumeric, National	N/A	N/A
	NCName	Alphabetic, Alphanumeric, National	N/A	N/A
	ID	Alphabetic, Alphanumeric, National	N/A	N/A
	IDREF	Alphabetic, Alphanumeric, National	N/A	N/A
	ENTITY	Alphabetic, Alphanumeric, National	N/A	N/A
	integer	Numeric, Alphanumeric[2], Numeric-edited	MOVE, NUMVAL	MOVE
	nonPositiveInteger	Numeric, Alphanumeric[2], Numeric-edited	MOVE, NUMVAL	MOVE
	negativeInteger	Numeric, Alphanumeric[2], Numeric-edited	MOVE, NUMVAL	MOVE
	long	Numeric, Alphanumeric[2], Numeric-edited	MOVE, NUMVAL	MOVE
	int	Numeric, Alphanumeric[2], Numeric-edited	MOVE, NUMVAL	MOVE
	short	Numeric, Alphanumeric[2], Numeric-edited	MOVE, NUMVAL	MOVE
	byte	Numeric, Alphanumeric[2], Numeric-edited	MOVE, NUMVAL	MOVE
	nonNegativeInteger	Numeric, Alphanumeric[2], Numeric-edited	MOVE, NUMVAL	MOVE
	unsignedLong	Numeric, Alphanumeric[2], Numeric-edited	MOVE, NUMVAL	MOVE
	unsignedInt	Numeric, Alphanumeric[2], Numeric-edited	MOVE, NUMVAL	MOVE
	unsignedShort	Numeric, Alphanumeric[2], Numeric-edited	MOVE, NUMVAL	MOVE
	unsignedByte	Numeric, Alphanumeric[2], Numeric-edited	MOVE, NUMVAL	MOVE

Table 3. Mapping of built-in XML types to COBOL data classes and categories (continued)

	XML Datatype	COBOL data Class or Category	Default Inbound Conversion	Default Outbound Conversion[1]
positiveInteger	Numeric, Alphanumeric[2], Numeric-edited	MOVE, NUMVAL	MOVE	

[1]The mapping tools will not enforce matching rules for user-defined simple XML schema types that are derived by constraint. For example, if for a base=″int″, the user defined type has a constraint of minInclusive value=″-99″ we may not be able to enforce the minInclusive constraint.

[2] Valid MOVEs between numeric and non-numeric types in XML and COBOL follow the rules described in the COBOL Language Reference Manual

XML and COBOL structure compatibility

You can match subordinate elementary COBOL group items to simple type elements of XML complex types.

Table 3 shows XML structure derivation from COBOL groups.

Table 4. XML and COBOL structure matching

COBOL structure	Matching XML structure

COBOL Redefine	Will always be defined as a group. If it is the outermost type then it will also be typed.
	`<complexType name="RedefinedElementName+_">`
	`   <xsd:group ref="RedefinedElementName+_"/>`
	`</complexType>`
	If it is a contained redefine then
	`<xsd:group name="RedefinedElementName+_">`
	`   <xsd:choice>`
	`   </xsd:choice>`
	`</xsd:complexType>`
	If the contained group is an array then it will also be typed.

COBOL Group	Will always be defined as a group. If it is the outermost type then it will also be typed.
	`<complexType name="DFHCOMMAREA">`
	`   <xsd:group ref="DFHCOMMAREA"/>`
	`</complexType>`
	If it is a contained group then
	`<xsd:group name="Mygroup">`
	`   <xsd:sequence>`
	`   </xsd:sequence>`
	`</xsd:complexType>`
	If the contained group is an array then it will also be typed.

Table 4. XML and COBOL structure matching (continued)

COBOL structure	Matching XML structure

COBOL OCCURS - fixed length array
 	
05 STOCKQUOTES COMP-1 OCCURS 5 TIMES.
	For web services support:
	`<complexType name="ArrayOfXXXXX">`
	   `<complexContent>`
	      `<restriction base="soapenc:Array">`
	         `<attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:float[5]"/>`
	     `</restriction>`
	   `</complexContent>`
	`</complexType>`
	
	For general support:
	`<element name="STOCKQUOTES" minOccurs="5" maxOccurs="5" type="xsd:float"/>`

COBOL OCCURS DEPENDING ON - variable length array (ODO)	`<element name="numStockQuotes">`
	`<xsd:simpleType>`
	`<xsd:restriction base="xsd:short">`
	   `<xsd:minInclusive value="00"/>`
	   `<xsd:maxInclusive value="99"/>`
	`</xsd:restriction>`
	`</xsd:simpleType>`
	`</element>`
	`<xsd:element name="stockQuotes">`
	
	For web services support:
	`<complexType name="ArrayOfXXXXX">`
	    `<complexContent>`
	      `<xsd:annotation>`
	         `<xsd:appinfo> <dependingOn>numStockQuotes</dependingOn>`
	            `</xsd:appinfo>`
	         `</xsd:annotation>`
	         `<restriction base="soapenc:Array">`
	            `<attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:float[]"/>`
	        `</restriction>`
	      `</complexContent>`
	   `</complexType>`
	`</element>`
	
	For general support:
	`<element name="STOCKQUOTES" minOccurs="1" maxOccurs="5" type="xsd:float">`
	`<xsd:annotation>`
	      `<xsd:appinfo> <dependingOn>numStockQuotes</dependingOn>`
	      `</xsd:appinfo>`
	   `</xsd:annotation>`
	`</element>`

XML types derived from COBOL

The table below shows how XML types are derived from COBOL types by the XML converter generators.

Table 5. COBOL to XML type derivation

COBOL Type	COBOLUsageValue+ COBOL ModelType properties	Corresponding XSD Type

COBOL Alphabetic Type
 	
05 Fname PIC A(20).)
		`<xsd:simpleType>`
		   `<restriction base="xsd:string">`
		      `<length value="n"/>`
		   `</restriction>`
		`</simpleType>`

Table 5. COBOL to XML type derivation (continued)

COBOL Type	**COBOLUsageValue+ COBOL ModelType properties**	**Corresponding XSD Type**

COBOL Alphanumeric Type		`<xsd:simpleType>`
		`   <restriction base="xsd:string">`
		`      <length value="n"/>`
		`   </restriction>`
		`</simpleType>`

COBOL Numeric Type		

display, binary, comp, comp-4, comp-5 ->display/binary	Display/binary +decimal	`<xsd:simpleType>`
		`<xsd:restriction base="xsd:decimal">`
		`   <xsd:minInclusive value="xx.x"/>`
		`   <xsd:maxInclusive value="yy.y"/>`
		`</xsd:restriction>`
		`</xsd:simpleType>`

	Display/binary +~decimal + number of nines <= 4 + sign	`<xsd:simpleType>`
		`<xsd:restriction base="xsd:short">`
		`   <xsd:minInclusive value="xx"/>`
		`   <xsd:maxInclusive value="yy"/>`
		`</xsd:restriction>`
		`</xsd:simpleType>`

	Display/binary +~decimal + 4 <number of nines <= 9+ sign	`<xsd:simpleType>`
		`<xsd:restriction base="xsd:int">`
		`   <xsd:minInclusive value="xx"/>`
		`   <xsd:maxInclusive value="yy"/>`
		`</xsd:restriction>`
		`</xsd:simpleType>`

COBOL Numeric Type		

Display, binary, comp, comp-4, comp-5 ->display/binary	Display/binary +~decimal + 9 <number of nines +sign	`<xsd:simpleType>`
		`<xsd:restriction base="xsd:long">`
		`   <xsd:minInclusive value="xx"/>`
		`   <xsd:maxInclusive value="yy"/>`
		`</xsd:restriction>`
		`</xsd:simpleType>`

	Display/binary +~decimal + number of nines <= 4 + no sign	`<xsd:simpleType>`
		`<xsd:restriction base="xsd:short">`
		`   <xsd:minInclusive value="xx"/>`
		`   <xsd:maxInclusive value="yy"/>`
		`</xsd:restriction>`
		`</xsd:simpleType>`

	Display/binary +~decimal + 4 <number of nines <= 9+ no sign	`<xsd:simpleType>`
		`<xsd:restriction base="xsd:int">`
		`   <xsd:minInclusive value="xx"/>`
		`   <xsd:maxInclusive value="yy"/>`
		`</xsd:restriction>`
		`</xsd:simpleType>`

	Display/binary +~decimal + 9 <number of nines + nosign	`<xsd:simpleType>`
		`<xsd:restriction base="xsd:long">`
		`   <xsd:minInclusive value="xx"/>`
		`   <xsd:maxInclusive value="yy"/>`
		`</xsd:restriction>`
		`</xsd:simpleType>`

packed-decimal, comp-3 -> packedDecimal	packedDecimal	`<xsd:simpleType>`
		`<xsd:restriction base="xsd:decimal">`
		`   <xsd:minInclusive value="xx.x"/>`
		`   <xsd:maxInclusive value="yy.y"/>`
		`</xsd:restriction>`
		`</xsd:simpleType>`

Table 5. COBOL to XML type derivation (continued)

COBOL Type	COBOLUsageValue+ COBOL ModelType properties	Corresponding XSD Type

comp-1 -> float	float	`<xsd:simpleType>`
		`<xsd:restriction base="xsd:float">`
		`   <xsd:minInclusive value="xx.x"/>`
		`   <xsd:maxInclusive value="yy.y"/>`
		`</xsd:restriction>`
		`</xsd:simpleType>`

comp-2 -> double	double	`xsd:simpleType>`
		`<xsd:restriction base="xsd:double">`
		`   <xsd:minInclusive value="xx.x"/>`
		`   <xsd:maxInclusive value="yy.y"/>`
		`</xsd:restriction>`
		`</xsd:simpleType>`

COBOL Alphanumeric-edited Type		`<xsd:simpleType>`
		`   <restriction base="string">`
		`      <length value="n"/>`
		`   </restriction>`
		`</simpleType>`

COBOL Numeric-edited Type		`<xsd:simpleType>`
		`   <restriction base="string">`
		`      <length value="n"/>`
		`   </restriction>`
		`</simpleType>`

COBOL DBCS Type	DBCS	`<xsd:simpleType>`
		`   <restriction base="string">`
		`      <length value="n"/>`
		`   </restriction>`
		`</simpleType>`

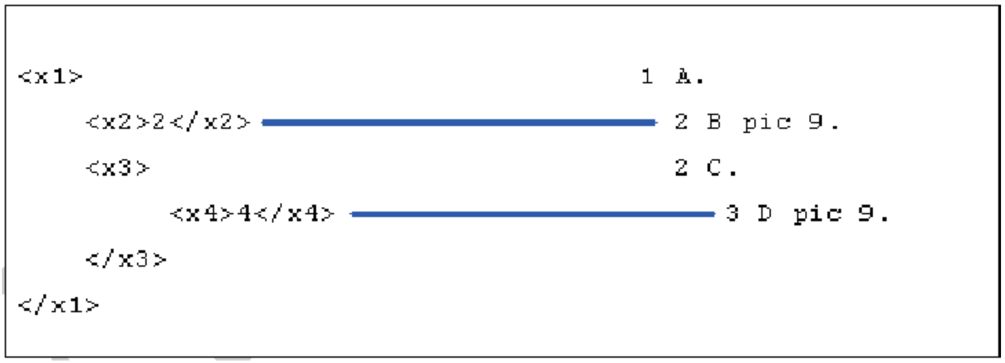
COBOL External floating point Type		`<xsd:simpleType>`
		`   <restriction base="string">`
		`      <length value="n"/>`
		`   </restriction>`
		`</simpleType>`

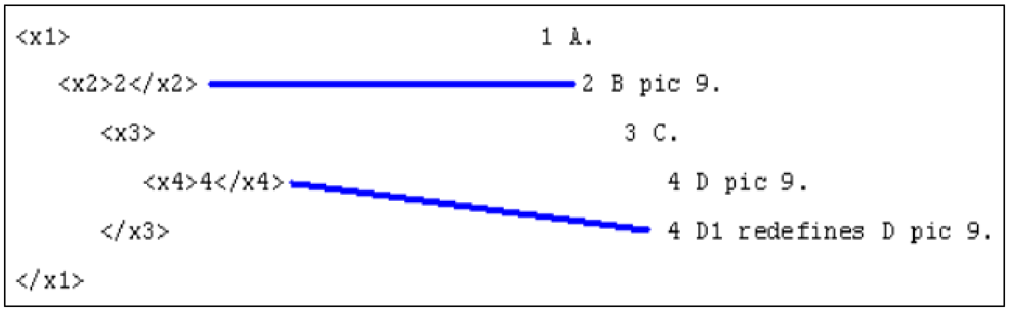
COBOL National (Unicode) Type	Data stored in Unicode format	
COBOL Address Type	- not supported -	
COBOL Object reference Type	- not supported -	

COBOL Level 88
 	
05 TXN-Resp-Code PIC X(3)
88 Business-Code value ″AAA″ THRU ″XXX″
88 Business-Error value ″XYX″ THRU ″ZYX″
88 Completed-Code value ″COM″
		`<xsd:element name="TXN_Resp_Code">`
		`   <xsd:annotation>`
		`      <xsd:appinfo>`
		`<level88>Business_Code value "AAA" THRU "XXX"</level88>`
		`<level88>Business_Error value "XYX" THRU "ZYX"</level88>`
		`<level88>Completed_Code value "COM"</level88>`
		`<level88></level88>`
		`<level88></level88>`
		`         </xsd:appinfo>`
		`     </xsd:annotation>`
		`   <xsd:simpleType>`
		`   <xsd:restriction base="xsd:string">`
		`      <xsd:length value="3"/>`
		`   </xsd:restriction>`
		`   </xsd:simpleType>`
		`</xsd:element>`

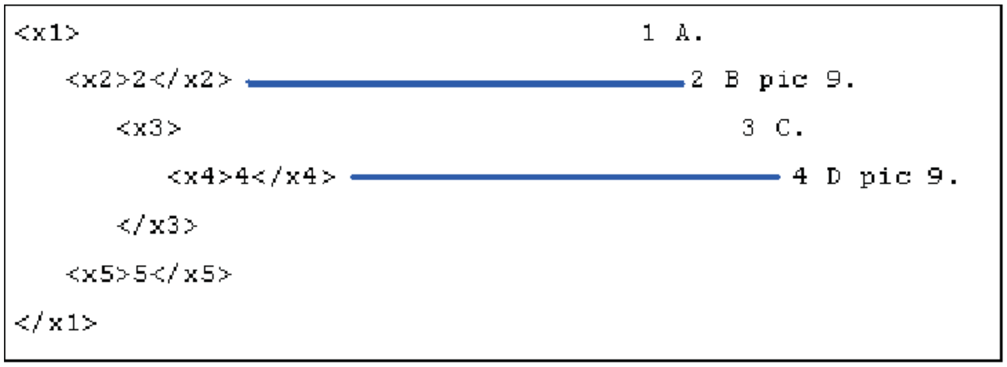
Isomorphic and non-isomorphic element mapping

The following two sets are isomorphic structures (an XML instance document and a COBOL data structure) and both have isomorphic element mapping:

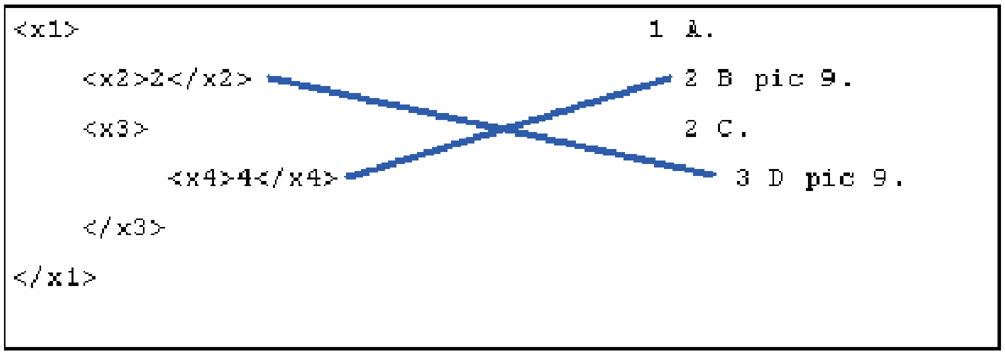




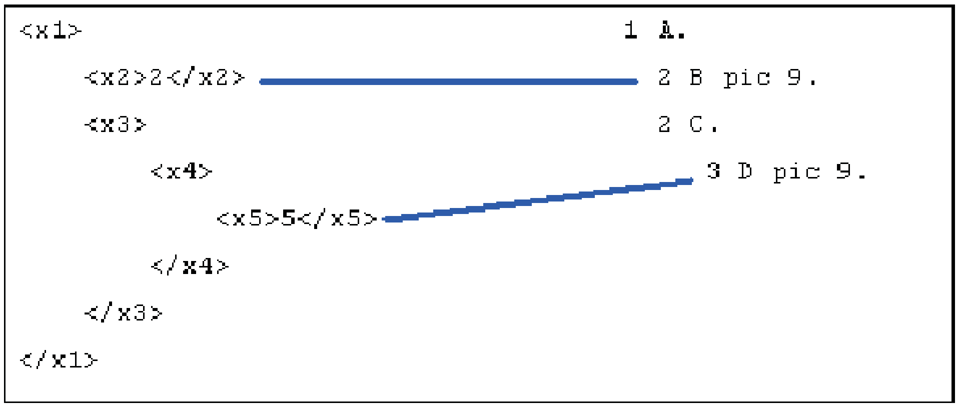
The following two non-isomorphic structures (an XML instance document and a COBOL data structure) have isomorphic subsets and show isomorphic element mapping:



The following two isomorphic structures (an XML instance document and a COBOL data structure) have non-isomorphic element mapping:



The following two non-isomorphic structures (an XML instance document and a COBOL data structure) have non-isomorphic element mapping:



Chapter 9. The CICS catalog manager example application

The CICS catalog example application is a working COBOL application that is designed to illustrate best practice when connecting CICS applications to external clients and servers.

The example is constructed around a simple sales catalog and order processing application, in which the end user can perform these functions:
- List the items in a catalog.
- Inquire on individual items in the catalog.(inquireSingle)
- Order items from the catalog.

The catalog is implemented as a VSAM file.

Installation of the base catalog manager application is covered in the CICS 3.1 Transaction Server documentation. A web client front end is provided with the catalog manager application. Configuration of the web client is also described in the CICS 3.1 Transaction Server documentation. The web client calls multiple web services that are provided by the base catalog example application after it has been enabled for web services.

The base application

The base application, with its 3270 user interface, provides functions with which you can list the contents of a stored catalog, select an item from the list, and enter a quantity to order. The application has a modular design which makes it simple to extend the application to support newer technology, such as Web services.

Figure 1 shows the structure of the base application.

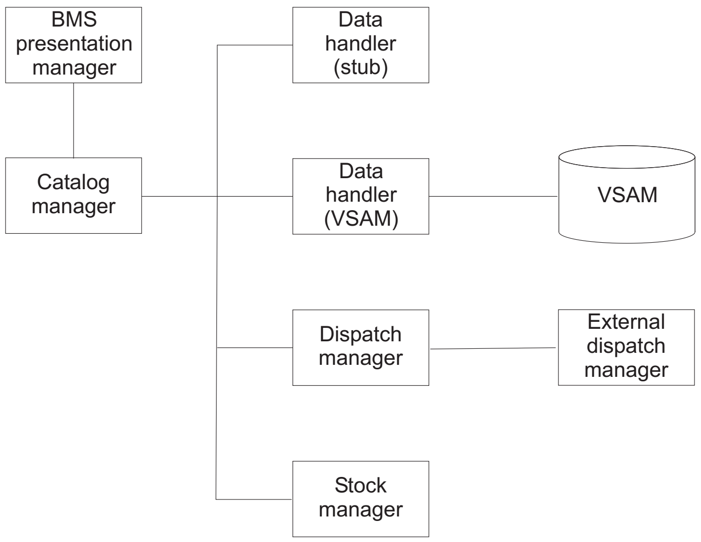


Figure 1. Structure of the base application

Installing and setting up the base application

Before you can run the base application you must define and populate two VSAM data sets, and install two transaction definitions.

Creating and defining the VSAM data sets

The example application uses two KSDS VSAM data sets to be defined and populated. One data set contains configuration information for the example application. The other contains the sales catalog.

1. Locate the JCL to create the VSAM data sets. During CICS installation, the JCL is placed in the *hlq*.SDFHSAMP library:
   - Member DFH$ECNF contains the JCL to generate the configuration data set.
   - Member DFH$ECAT contains the JCL to generate the catalog data set.
2. Modify the JCL and access method services commands.
   a. Supply a valid JOB card.
   b. Supply a suitable high level qualifier for the data set names in the access method services commands. As supplied, the JCL uses a high level qualifier of `HLQ`.

      The following command defines the catalog file:

      ```
      DEFINE CLUSTER (NAME(hlq.EXMPLAPP.catname)-
             TRK(1 1) -
             KEYS(4 0) -
             RECORDSIZE(80,80) -
             SHAREOPTIONS(2 3) -
             INDEXED -
             ) -
             DATA (NAME(hlq.EXMPLAPP.catname.DATA) -
             ) -
             INDEX (NAME(hlq.EXMPLAPP.catname.INDEX) -
             )
      ```

      where
      - *hlq* is a high level qualifier of your choice
      - *catname* is a name of your choice. The name used in the example application as supplied is `EXMPCAT`.

      .

      The following command defines the configuration file:

      ```
      DEFINE CLUSTER (NAME(hlq.EXMPLAPP.EXMPCONF)-
                TRK(1 1) -
                KEYS(9 0) -
                RECORDSIZE(350,350) -
                SHAREOPTIONS(2 3) -
                INDEXED -
                ) -
                DATA (NAME(hlq.EXMPLAPP.EXMPCONF.DATA) -
                ) -
                INDEX (NAME(hlq.EXMPLAPP.EXMPCONF.INDEX) -
                )
      ```

      where *hlq* is a high level qualifier of your choice.
3. Run both jobs to create and populate the data sets.
4. Use the CEDA transaction to create a FILE definition for the catalog file.
   a. Enter the following: `CEDA DEF FILE(EXMPCAT)G(EXAMPLE)`. Alternatively, you can copy the FILE definition from CICS supplied group DFH$EXBS.
   b. Enter the following additional attributes:

      ```
      DSNAME(hlq.EXMPLAPP.EXMPCAT)
      ADD(YES)
      BROWSE(YES)
      DELETE(YES)
      READ(YES)
      UPDATE(YES)
      ```
   c. Use the default values for all other attributes.

5. Use the CEDA transaction to create a FILE definition for the configuration file.
   a. Enter the following: `CEDA DEF FILE(EXMPCONF) G(EXAMPLE)`. Alternatively, you can copy the FILE definition from CICS supplied group DFH$EXBS.
   b. Enter the following additional attributes:
      ```
      DSNAME(hlq.EXMPLAPP.EXMPCONF)
      ADD(YES)
      BROWSE(YES)
      DELETE(YES)
      READ(YES)
      UPDATE(YES)
      ```
   c. Use the default values for all other attributes.

Defining the 3270 interface

The example application is supplied with a 3270 user interface to run the application and to customize it. The user interface consists of two transactions, EGUI and ECFG.

Use the CEDA transaction to create TRANSACTION definitions for both transactions.
1. To define transaction EGUI, enter the following: `CEDA DEF TRANS (EGUI) G(EXAMPLE) PROG(DFH0XGUI)`.
2. To define transaction ECFG, enter the following: `CEDA DEF TRANS (ECFG) G(EXAMPLE) PROG(DFH0XCFG)`

Use the default values for all other attributes.

Note: The correct operation of the example application does not depend on the names of the transactions, so you can use different names if you wish.
Alternatively, you can copy the TRANSACTION definitions from CICS supplied group DFH$EXBS.

Completing the installation

To complete the installation, install the RDO group that contains your resource definitions.

Enter the following command at a CICS terminal: `CEDA I G(EXAMPLE)`.

The application is now ready for use.

Web service support for the example application

The Web service support extends the example application, providing a Web client front end and two versions of a Web service endpoint for the order dispatcher component.

The Web client front end and one version of the Web service endpoint are supplied as enterprise archives (EARs) that will run in the following environments:
- WebSphere Application Server Version 5 Release 1 or later
- WebSphere Studio Application Developer Version 5 Release 1 or later with a WebSphere unit test environment
- WebSphere Studio Enterprise Developer Version 5 Release 1 or later with a WebSphere unit test environment

The second version of the Web service endpoint is supplied as a CICS service provider application program (DFH0XODE).

Configuring code page support

As supplied, the example application uses two coded character sets. You must configure your system to enable data conversion between the two character sets.

The coded character sets used in the example application are:

CCSID	Description
037	EBCDIC Group 1: USA, Canada (z/OS), Netherlands, Portugal, Brazil, Australia, New Zealand)
1208	UTF-8 Level 3

Add the following statements to the conversion image for your z/OS system:

```
CONVERSION 037,1208;
CONVERSION 1208,037;
```

Installing Web service support

Before you can run the Web service support for the example application, you must create two HFS directories, and create and install a number of CICS resource definitions.

Creating the HFS directories

Web service support for the example application requires a shelf directory and a pickup directory in the Hierarchical File System (HFS).

The shelf directory is used to store the Web service binding files that are associated with WEBSERVICE resources. Each WEBSERVICE resource is, in turn, associated with a PIPELINE. The shelf directory is managed by the PIPELINE resource and you should not modify its contents directly. Several PIPELINES can use the same shelf directory, as CICS ensures a unique directory structure beneath the shelf directory for each PIPELINE.

The pickup directory is the directory that contains the Web service binding files associated with a PIPELINE. When a PIPELINE is installed, or in response to a PERFORM PIPELINE SCAN command, information in the binding files is used to dynamically create the WEBSERVICE and URIMAP definitions associated with the PIPELINE.

The example application uses `/var/cicsts` for the shelf directory.

A pipeline will read in an XML pipeline configuration file at install time. It is therefore also useful to define a directory in which to store these.

Creating the PIPELINE definition

The complete definition of a pipeline consists of a PIPELINE resource and a pipeline configuration file. The file contains the details of the message handlers that will act on Web service requests and responses as they pass through the pipeline.

The example application uses the CICS-supplied SOAP 1.1 handler to deal with the SOAP envelopes of inbound and outbound requests. CICS provides sample pipeline configuration files which you can use in your service provider and service requester.

More than one WEBSERVICE can share a single PIPELINE, therefore you need define only one pipeline for the inbound requests of the example application. You must, however, define a second PIPELINE for the outbound requests as a single PIPELINE cannot be configured to be both a provider and requester pipeline at the same time.

1. Use the CEDA transaction to create a PIPELINE definition for the service provider.

   a. Enter the following: `CEDA DEF PIPE(EXPIPE01) G(EXAMPLE)`. Alternatively, you can copy the PIPELINE definition from CICS supplied group DFH$EXWS.
   b. Enter the following additional attributes:

      ```
      STATUS(Enabled)
      CONFIGFILE(/usr/lpp/cicsts
                /samples/pipelines/basicsoap11provider.xml)
      SHELF(var/cicsts)
      WSDIR(/usr/lpp/cicsts/samples/webservices/wsbind/provider/)
      ```

      Note that the HFS entries are case sensitive and assume a default CICS HFS install root of `/usr/lpp/cicsts`.
2. Use the CEDA transaction to create a PIPELINE definition for the service requester.
   a. Enter the following: `CEDA DEF PIPE(EXPIPE02) G(EXAMPLE)`. Alternatively, you can copy the PIPELINE definition from CICS supplied group DFH$EXWS.
   b. Enter the following additional attributes:

      ```
      STATUS(Enabled)
      CONFIGFILE(/usr/lpp/cicsts
                /samples/pipelines/basicsoap11requester.xml)
      SHELF(var/cicsts)
      WSDIR(/usr/lpp/cicsts/samples/webservices/wsbind/requester/)
      ```

      Note that the HFS entries are case sensitive and assume a default CICS HFS install root of `/usr/lpp/cicsts`.

Creating a TCPIPSERVICE

As the client connects to your Web services over an HTTP transport you must define a TCPIPSERVICE to receive the inbound HTTP traffic.

Use the CEDA transaction to create a TCPIPSERVICE definition to handle inbound HTTP requests.

1. Enter the following: `CEDA DEF TCPIPSERVICE(EXMPPPORT) G(EXAMPLE)`. Alternatively, you can copy the TCPIPSERVICE definition from CICS supplied group DFH$EXWS.
2. Enter the following additional attributes:

   `URM(NONE)`

   `PORTNUMBER(`*`port`*`)` where *`port`* is an unused port number in your CICS system.

   `PROTOCOL(HTTP)`

   `TRANSACTION(CWXN)`
3. Use the default values for all other attributes.

Dynamically installing the WEBSERVICE and URIMAP resources

Each function exposed as a Web service requires a WEBSERVICE resource to map between the incoming XML of the SOAP BODY and the COMMAREA interface of the program, and a URIMAP resource that routes incoming requests to the correct PIPELINE and WEBSERVICE. Although you can use RDO to define and install your WEBSERVICE and URIMAP resources, you can also have CICS create them dynamically when you install a PIPELINE resource.

Install the PIPELINE resources. Use the following commands:

```
CEDA INSTALL PIPELINE(EXPIPE01) G(EXAMPLE)
CEDA INSTALL PIPELINE(EXPIPE02) G(EXAMPLE)
```

When you install each PIPELINE resource, CICS scans the directory specified in the PIPELINE's WSDIR attribute (the pickup directory). For each Web service binding file in the directory, that is for each file with the `.wsbind` suffix, CICS installs a WEBSERVICE and a URIMAP if one does not already exist. Existing resources are replaced if the information in the binding file is newer than the existing resources.

When the PIPELINE is later disabled and discarded all associated WEBSERVICE and URIMAP resources will also be discarded.

If you have already installed the PIPELINE, use the PERFORM PIPELINE SCAN command to initiate the scan of the PIPELINE's pickup directory.

When you have installed the PIPELINEs, the following WEBSERVICEs and their associated URIMAPs will be installed in your system:

```
dispatchOrder
dispatchOrderEndpoint
inquireCatalog
inquireSingle
placeOrder
```

The names of the WEBSERVICEs are derived from the names of the Web service binding files; the names of the URIMAPs are generated dynamically. You can view the resources with a CEMT INQUIRE WEBSERVICE command:

```
I WEBS
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Webs(dispatchOrder                  ) Pip(EXPIPE02)
    Ins                                                    Dat(20041203)
 Webs(dispatchOrderEndpoint          ) Pip(EXPIPE01)
    Ins Uri(£539140 ) Pro(DFH0XODE) Com                    Dat(20041203)
 Webs(inquireCatalog                 ) Pip(EXPIPE01)
    Ins Uri(£539141 ) Pro(DFH0XCMN) Com                    Dat(20041203)
 Webs(inquireSingle                  ) Pip(EXPIPE01)
    Ins Uri(£539142 ) Pro(DFH0XCMN) Com                    Dat(20041203)
 Webs(placeOrder                     ) Pip(EXPIPE01)
    Ins Uri(£539150 ) Pro(DFH0XCMN) Com                    Dat(20041203)
```

The display shows the names of the PIPELINE, the URIMAP, and the target program that is associated with each WEBSERVICE. Note that in this example, there is no URIMAP or target program displayed for WEBSERVICE(dispatchOrder) because the WEBSERVICE is for an outbound request.

WEBSERVICE(dispatchOrderEndpoint) represents the local CICS implementation of the dispatch order service.

Creating the WEBSERVICE resources with RDO

As an alternative to using the PIPELINE scanning mechanism to install WEBSERVICE resources, you can create and install them using Resource Definition Online (RDO).

Important: If you use RDO to define the WEBSERVICE and URIMAP resources, you must ensure that their Web service binding files are **not** in the PIPELINE's pickup directory.

1. Use the CEDA transaction to create a WEBSERVICE definition for the inquire catalog function of the example application.
   a. Enter the following: `CEDA DEF WEBSERVICE(EXINQCWS) G(EXAMPLE)`.
   b. Enter the following additional attributes:

   ```
   PIPELINE(EXPIPE01)
   WSBIND(/usr/lpp/cicsts/samples
          /webservices/wsbind/inquireCatalog.wsbind)
   ```

2. Repeat the preceding step for each of the following functions of the example application.

Function	WEBSERVICE name	PIPELINE attribute	WSBIND attribute
INQUIRE SINGLE ITEM	EXINQSWS	EXPIPE01	`/usr/lpp/cicsts/samples /webservices/wsbind /provider/inquireSingle.wsbind`
PLACE ORDER	EXORDRWS	EXPIPE01	`/usr/lpp/cicsts/samples /webservices/wsbind /provider/placeOrder.wsbind`
DISPATCH STOCK	EXODRQWS	EXPIPE02	`/usr/lpp/cicsts/samples /webservices/wsbind /requester/dispatchOrder.wsbind`
DISPATCH STOCK endpoint (optional)	EXODEPWS	EXPIPE01	`/usr/lpp/cicsts/samples /webservices/wsbind /provider/dispatchOrderEndpoint.wsbind`

Creating the URIMAP resources with RDO

As an alternative to using the PIPELINE scanning mechanism to install URIMAP resources, you can create and install them using Resource Definition Online (RDO).

Important: If you use RDO to define the WEBSERVICE and URIMAP resources, you must ensure that their Web service binding files are **not** in the PIPELINE's pickup directory.

1. Use the CEDA transaction to create a URIMAP definition for the inquire catalog function of the example application.
   a. Enter the following: `CEDA DEF URIMAP(INQCURI) G(EXAMPLE)`.
   b. Enter the following additional attributes:

```
USAGE(PIPELINE)
HOST(*)
PATH(/exampleApp/inquireCatalog)
TCPIPSERVICE(SOAPPORT)
PIPELINE(EXPIPE01)
WEBSERVICE(EXINQCWS)
```

2. Repeat the preceding step for each of the remaining functions of the example application. Use the following names for your URIMAPs:

Function	URIMAP name
INQUIRE SINGLE ITEM	INQSURI
PLACE ORDER	ORDRURI
DISPATCH STOCK	Not required
DISPATCH STOCK endpoint (optional)	ODEPURI

   a. Specify the following distinct attributes for each URIMAP:

Function	URIMAP name	PATH	WEBSERVICE
INQUIRE SINGLE ITEM	INQSURI	/exampleApp/inquireSingle	EXINQSWS
PLACE ORDER	ORDRURI	/exampleApp/placeOrder	EXORDRWS
DISPATCH STOCK endpoint (optional)	ODEPURI	/exampleApp/dispatchOrder	EXODEPWS

   b. Enter the following additional attributes, which are the same for all the URIMAPs:

```
USAGE(PIPELINE)
```

```
HOST(*)
TCPIPSERVICE(SOAPPORT)
PIPELINE(EXPIPE01)
```

Completing the installation

To complete the installation, install the RDO group that contains your resource definitions.

Enter the following command at a CICS terminal: `CEDA I G(EXAMPLE)`.

The application is now ready for use.

Configuring the example application

The base application includes a transaction (ECFG) that you can use to configure the example application.

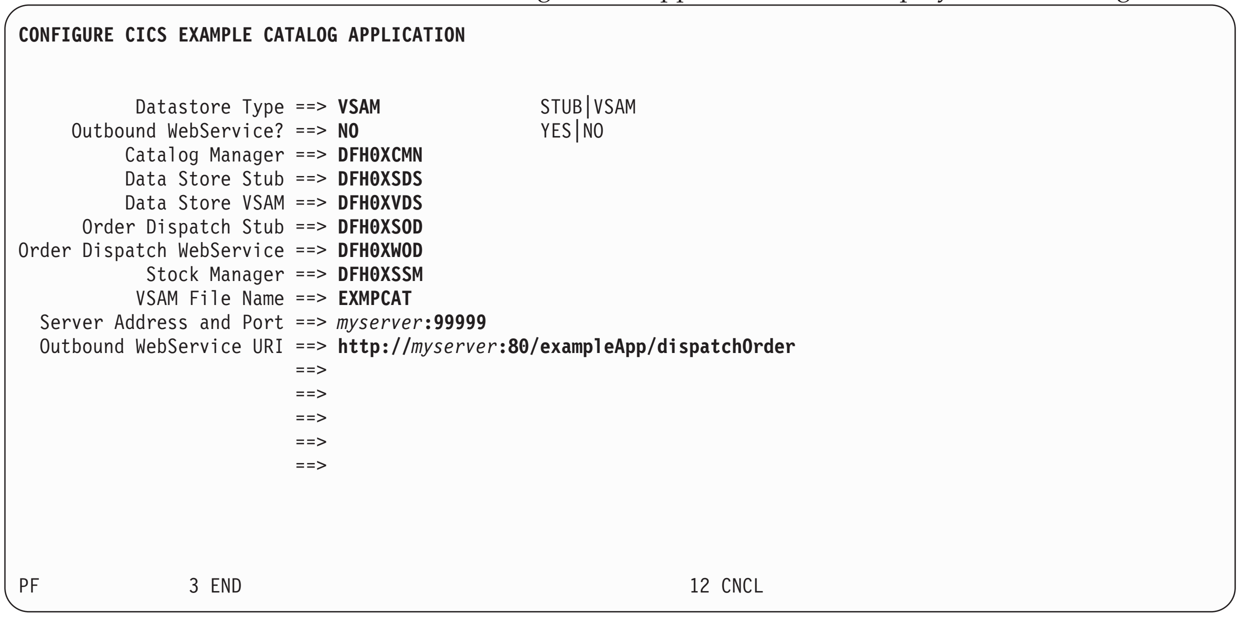
The configuration transaction uses mixed case information. You must use a terminal that can handle mixed case information correctly.

The transaction lets you specify a number of aspects of the example application. These include:

- The overall configuration of the application, such as the use of Web services
- The network addresses used by the Web services components of the application
- The names of resources, such as the file used for the data store
- The names of programs used for each component of the application

The configuration transaction lets you replace CICS-supplied components of the example application with your own, without restarting the application.

1. Enter the transaction ECFG to start the configuration application. CICS displays the following screen:



```
CONFIGURE CICS EXAMPLE CATALOG APPLICATION


          Datastore Type ==> VSAM              STUB|VSAM
    Outbound WebService? ==> NO                YES|NO
         Catalog Manager ==> DFH0XCMN
          Data Store Stub ==> DFH0XSDS
          Data Store VSAM ==> DFH0XVDS
      Order Dispatch Stub ==> DFH0XSOD
Order Dispatch WebService ==> DFH0XWOD
            Stock Manager ==> DFH0XSSM
           VSAM File Name ==> EXMPCAT
  Server Address and Port ==> myserver:99999
  Outbound WebService URI ==> http://myserver:80/exampleApp/dispatchOrder
                          ==>
                          ==>
                          ==>
                          ==>
                          ==>




PF             3 END                                      12 CNCL
```

2. Complete the fields.

   Datastore Type
   Specify STUB if you want to use the Data Store Stub program.

   Specify VSAM if you want to use the VSAM data store program.

Outbound WebService
Specify YES if you want to use a remote Web service for your Order Dispatch function, that is, if you want the catalog manager program to link to the Order Dispatch Web service program.

Specify NO if you want to use a stub program for your Order Dispatch function, that is, if you want the catalog manager program to link to the Order Dispatch Stub program.

Catalog Manager
Specify the name of the Catalog Manager program. The program supplied with the example application is DFH0XCMN.

Data Store Stub
If you specified STUB in the **Datastore Type** field, specify the name of the Data Store Stub program. The program supplied with the example application is DFH0XSDS.

Data Store VSAM
If you specified VSAM in the **Datastore Type** field, specify the name of the VSAM data store program. The program supplied with the example application is DFH0XVDS.

Order Dispatch Stub
If you specified NO in the **Outbound WebService** field, specify the name of the Order Dispatch Stub program. The program supplied with the example application is DFH0XSOD.

Order Dispatch WebService
If you specified YES in the **Outbound WebService** field, specify the name of the program that functions as a service requester. The program supplied with the example application is DFH0XWOD.

Stock Manager
Specify the name of the Stock Manager program. The program supplied with the example application is DFH0XSSM.

VSAM File Name
If you specified VSAM in the **Datastore Type** field, specify the name of the CICS FILE definition. The name used in the example application as supplied is EXMPCAT.

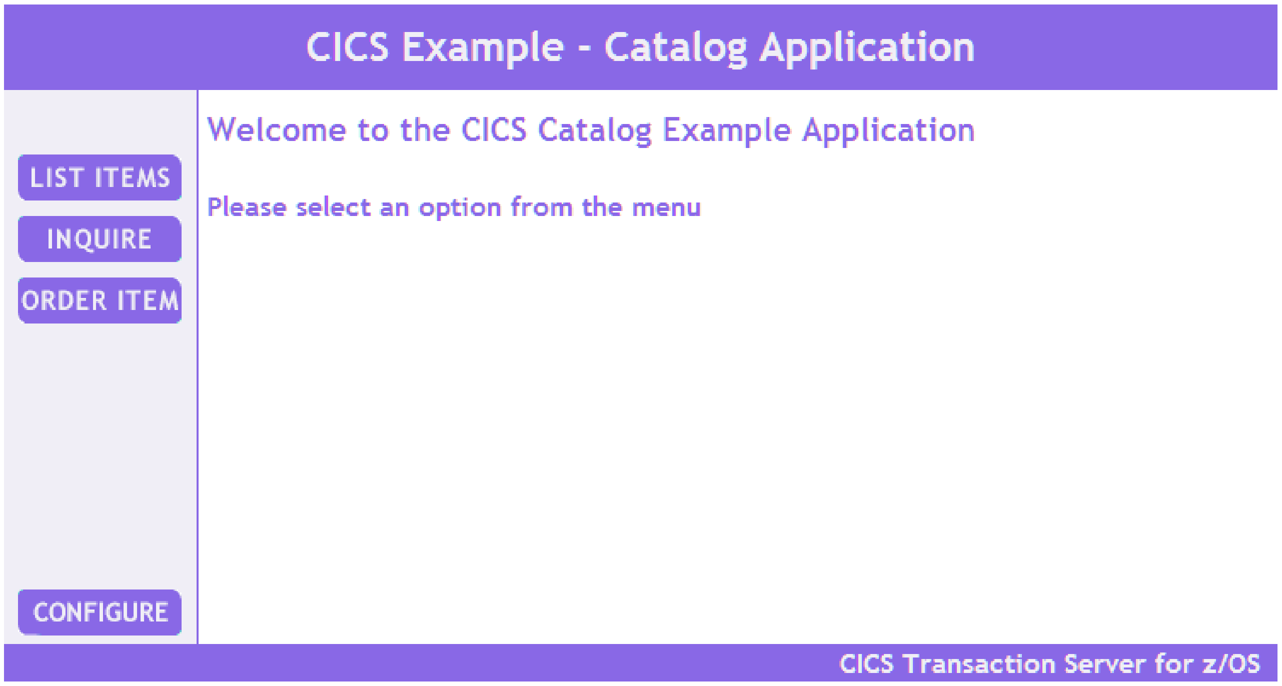
Server Address and Port

Outbound WebService URI
If you specified YES in the **Outbound WebService** field, specify the location of the Web service that implements the dispatch order function. If you are using the supplied CICS endpoint set this to: `http://`*`myserver`*`:`*`myport`*`/exampleApp/dispatchOrder` where *`myserver`* and *`myport`* are your CICS server address and port respectively.

Configuring the Web client

Before you can use the Web client, you must configure it to call the appropriate end points in your CICS system.

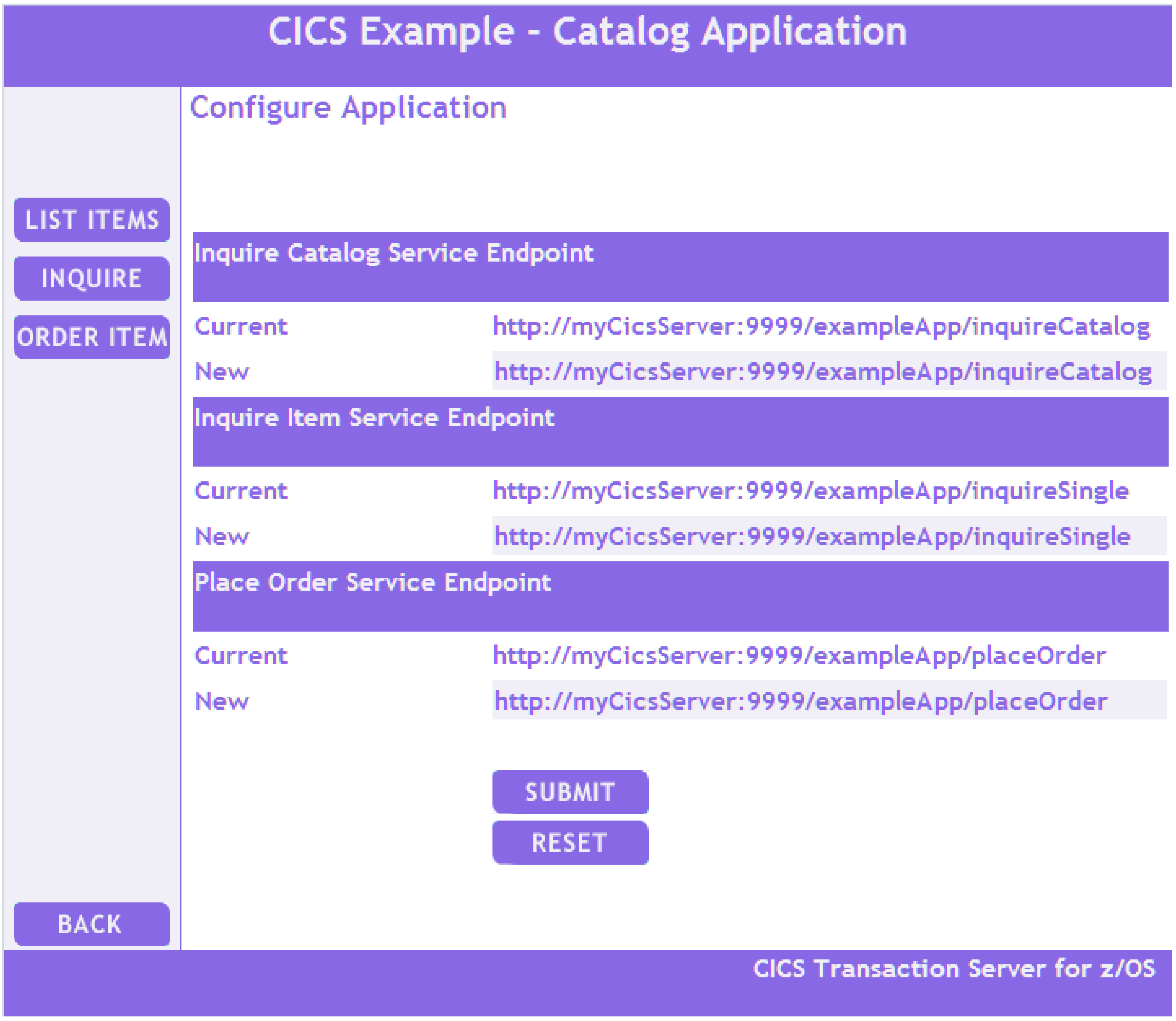
1. Enter the following in your Web browser: `http://`*`myserver`*`:9080/ExampleAppClientWeb/`, where *`myserver`* is the hostname of the server on which the Web service client is installed. The example application displays the following page:


CICS Example - Catalog Application
LIST ITEMS
INQUIRE
ORDER ITEM
CONFIGURE
CICS Transaction Server for z/OS

Welcome to the CICS Catalog Example Application

Please select an option from the menu

2. Click the **CONFIGURE** button to bring up the configuration page. The configuration page is displayed.


CICS Example - Catalog Application

Configure Application

Inquire Catalog Service Endpoint

Current	http://myCicsServer:9999/exampleApp/inquireCatalog
New	http://myCicsServer:9999/exampleApp/inquireCatalog

Inquire Item Service Endpoint

Current	http://myCicsServer:9999/exampleApp/inquireSingle
New	http://myCicsServer:9999/exampleApp/inquireSingle

Place Order Service Endpoint

Current	http://myCicsServer:9999/exampleApp/placeOrder
New	http://myCicsServer:9999/exampleApp/placeOrder

CICS Transaction Server for z/OS

3. Enter the new endpoints for the Web service. There are three endpoints to configure:

   Inquire catalog

   Inquire item

   Place order

   a. In the URLs replace the string 'myCicsServer' with the name of the system on which your CICS is running.
   b. Replace the port number '9999' with the port number configured in the TCPIPSERVICE, in the example this to 30000.
4. Click the **SUBMIT** button.

The Web application is now ready to run.

Note: The URL the Web services invoke is stored in an HTTP session. It is therefore necessary to repeat this configuration step each time a browser is first connected to the client.

Running the example application

You can run the example application in two ways: you can use the BMS interface, and you can use a Web client.

Running the example application with the BMS interface

The base application can be invoked using its BMS interface.

1. Enter transaction EGUI from a CICS terminal. The example application displays the following menu:

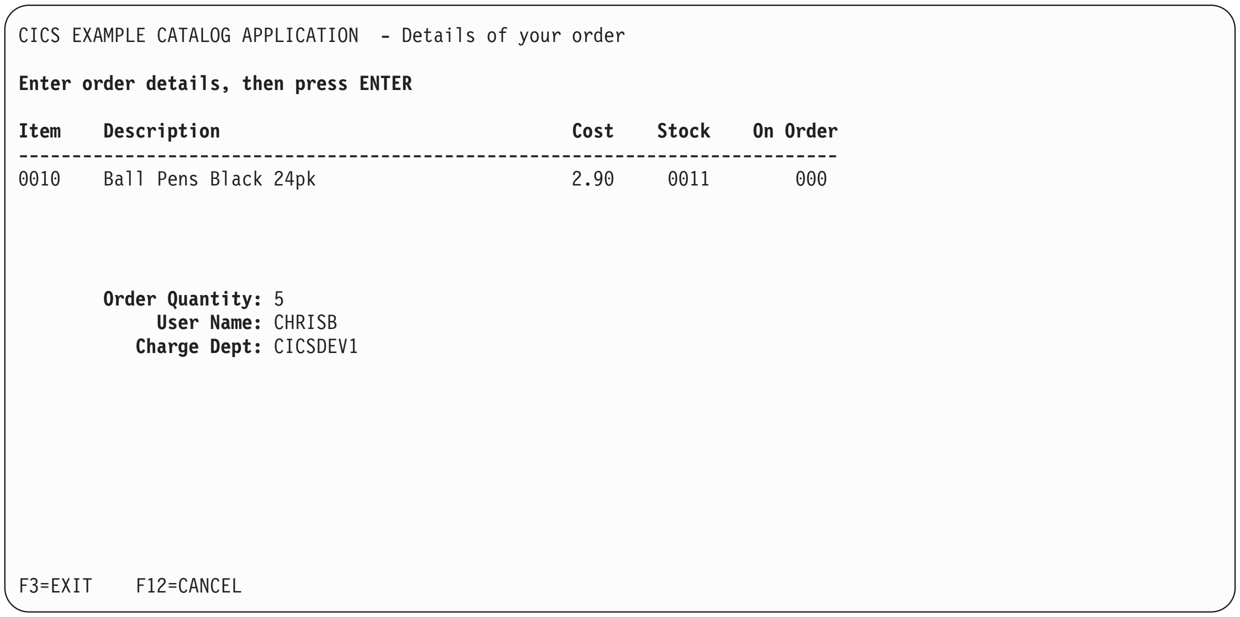
```
CICS EXAMPLE CATALOG APPLICATION  - Main Menu

Select an action, then press ENTER

Action . . . .     1. List Items
                   2. Order Item Number ____
                   3. Exit













F3=EXIT    F12=CANCEL
```

   The options on the menu enable you to list the items in the catalog, order an item, or exit the application.

2. Type 1 and press ENTER to select the LIST ITEMS option. The application displays a list of items in the catalog.

```
CICS EXAMPLE CATALOG APPLICATION  - Inquire Catalog

Select a single item to order with /, then press ENTER

Item    Description                                 Cost   Order
-----------------------------------------------------------------
0010    Ball Pens Black 24pk                        2.90     /
0020    Ball Pens Blue 24pk                         2.90     _
0030    Ball Pens Red 24pk                          2.90     _
0040    Ball Pens Green 24pk                        2.90     _
0050    Pencil with eraser 12pk                     1.78     _
0060    Highlighters Assorted 5pk                   3.89     _
0070    Laser Paper 28-lb 108 Bright 500/ream       7.44     _
0080    Laser Paper 28-lb 108 Bright 2500/case     33.54     _
0090    Blue Laser Paper 20lb 500/ream              5.35     _
0100    Green Laser Paper 20lb 500/ream             5.35     _
0110    IBM Network Printer 24 - Toner cart       169.56     _
0120    Standard Diary: Week to view 8 1/4x5 3/4   25.99     _
0130    Wall Planner: Eraseable 36x24              18.85     _
0140    70 Sheet Hard Back wire bound notepad       5.89     _
0150    Sticky Notes 3x3 Assorted Colors 5pk        5.35     _


F3=EXIT    F7=BACK   F8=FORWARD   F12=CANCEL
```

3. Type / in the **ORDER** column, and press ENTER to order an item. The application displays details of the item to be ordered.



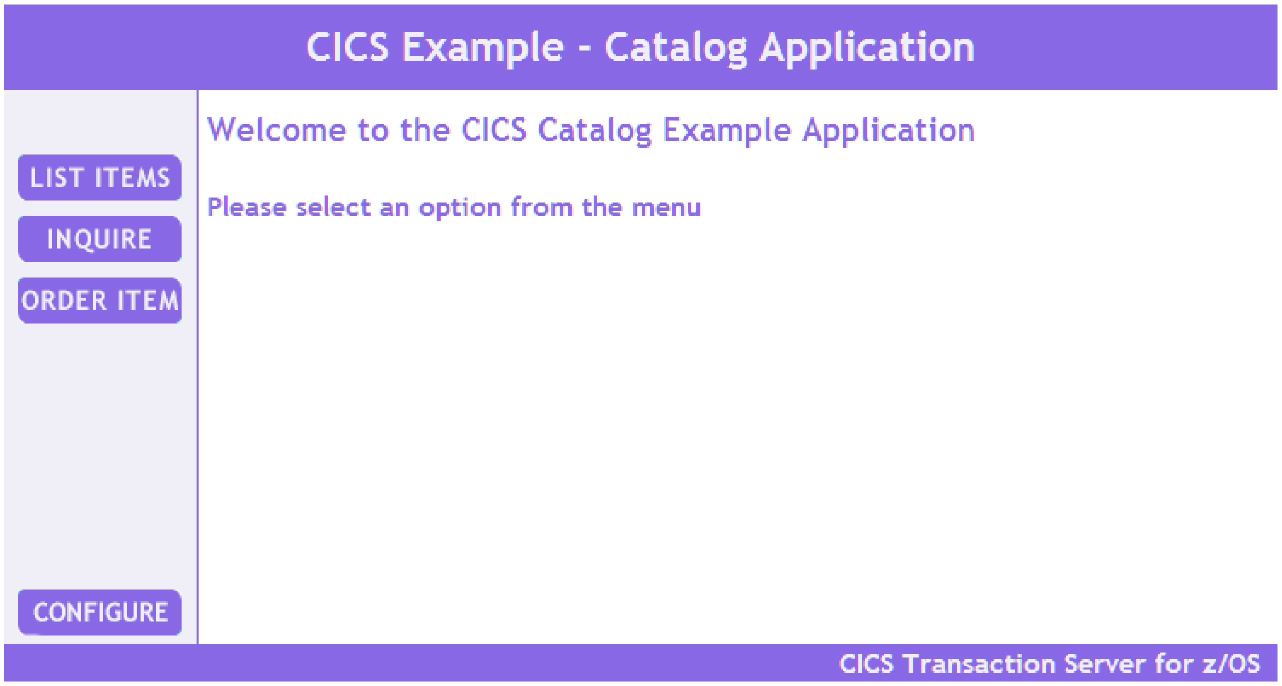
```
CICS EXAMPLE CATALOG APPLICATION  - Details of your order

Enter order details, then press ENTER

Item    Description                                 Cost    Stock    On Order
-----------------------------------------------------------------------------
0010    Ball Pens Black 24pk                        2.90     0011        000




        Order Quantity: 5
             User Name: CHRISB
           Charge Dept: CICSDEV1









F3=EXIT    F12=CANCEL
```

4. If there is sufficient stock to fulfil the order, enter the following information.
   a. Complete the `ORDER QUANTITY` field. Specify the number of items you want to order.
   b. Complete the `USERID` field. Enter a 1 to 8-character string. The base application does not check the value that is entered here.
   c. Complete the `CHARGE DEPT` field. Enter a 1 to 8-character string. The base application does not check the value that is entered here.
5. Press `ENTER` to submit the order and return to the main menu.
6. Select the EXIT option to end the application.

The Web service enabled application

You can invoke the example application from a Web browser.

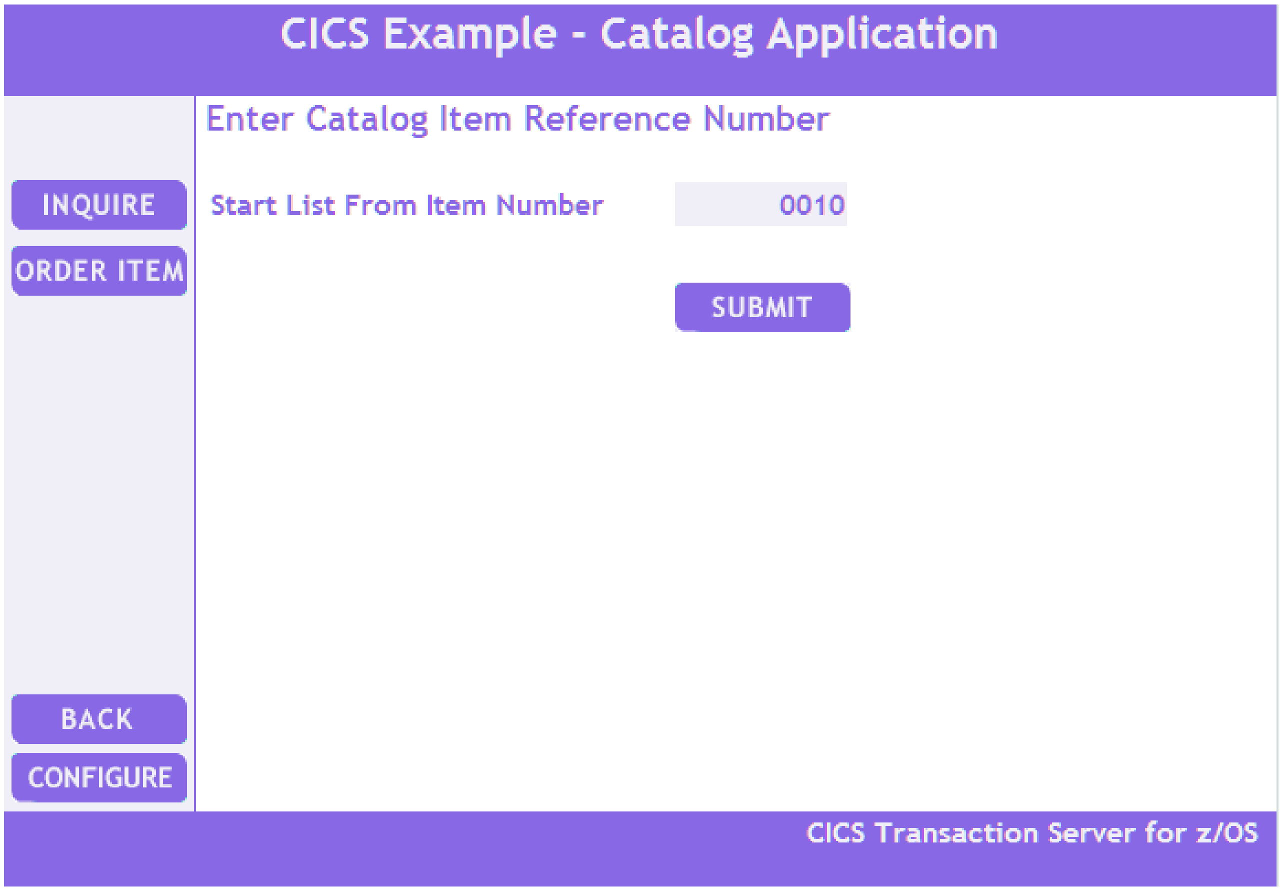
1. Enter the following in your Web browser: `http://`*`myserver`*`:9080/ExampleAppClientWeb/`, where *`myserver`* is the host name of the server on which the Web service client is installed. The example application displays the following page:


CICS Example - Catalog Application
LIST ITEMS
INQUIRE
ORDER ITEM
CONFIGURE
CICS Transaction Server for z/OS

Welcome to the CICS Catalog Example Application

Please select an option from the menu

2. Click the **INQUIRE** button. The example application displays the following page:

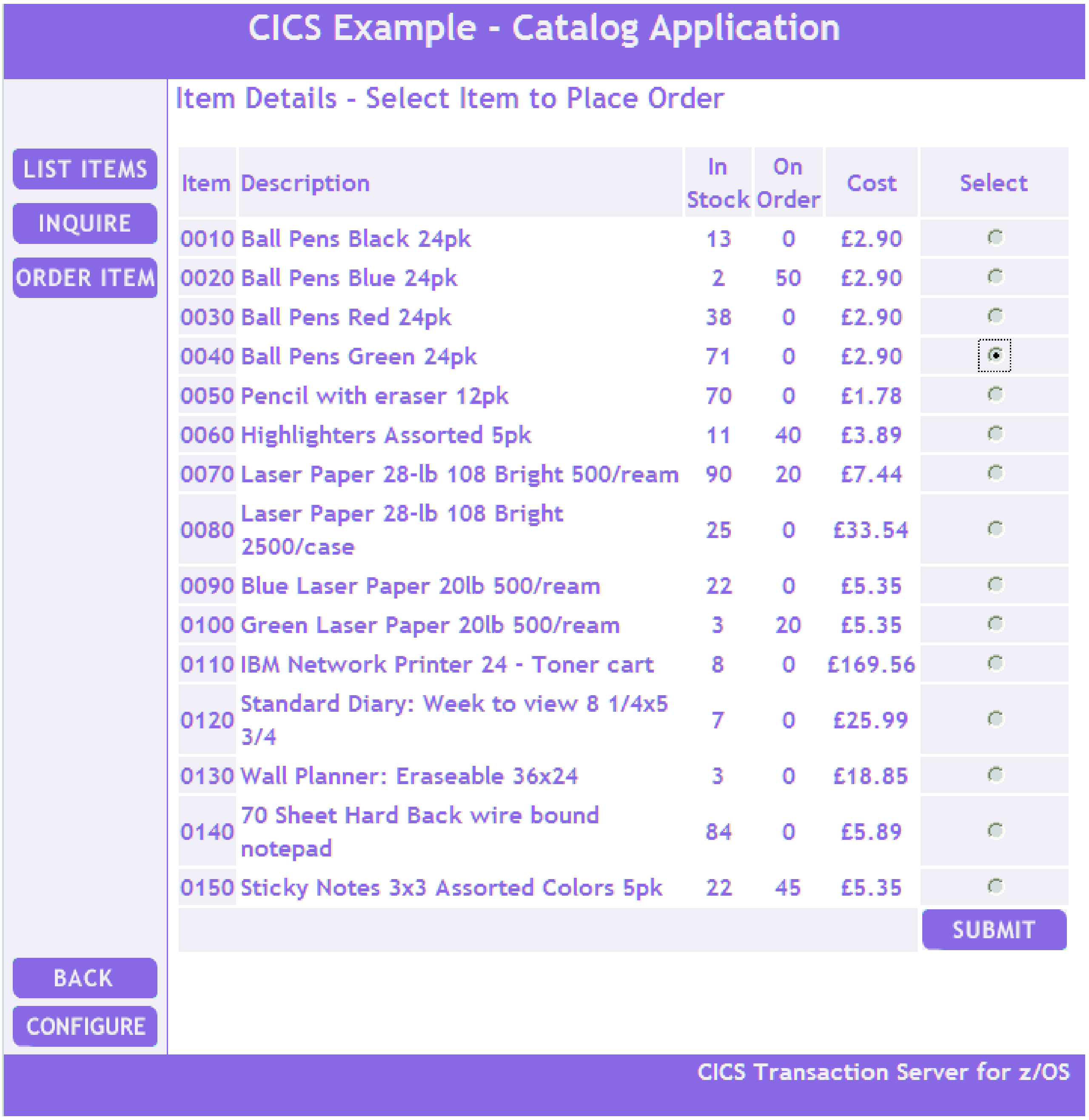

CICS Example - Catalog Application
INQUIRE
ORDER ITEM
SUBMIT
BACK
CONFIGURE
CICS Transaction Server for z/OS

Enter Catalog Item Reference Number

Start List From Item Number 0010

3. Enter an item number, and click the **SUBMIT** button.

Tip: The base application is primed with item numbers in the sequence 0010, 0020, ... through 0210.
The application displays the following page, which contains a list of items in the catalog, starting with the item number that you entered.

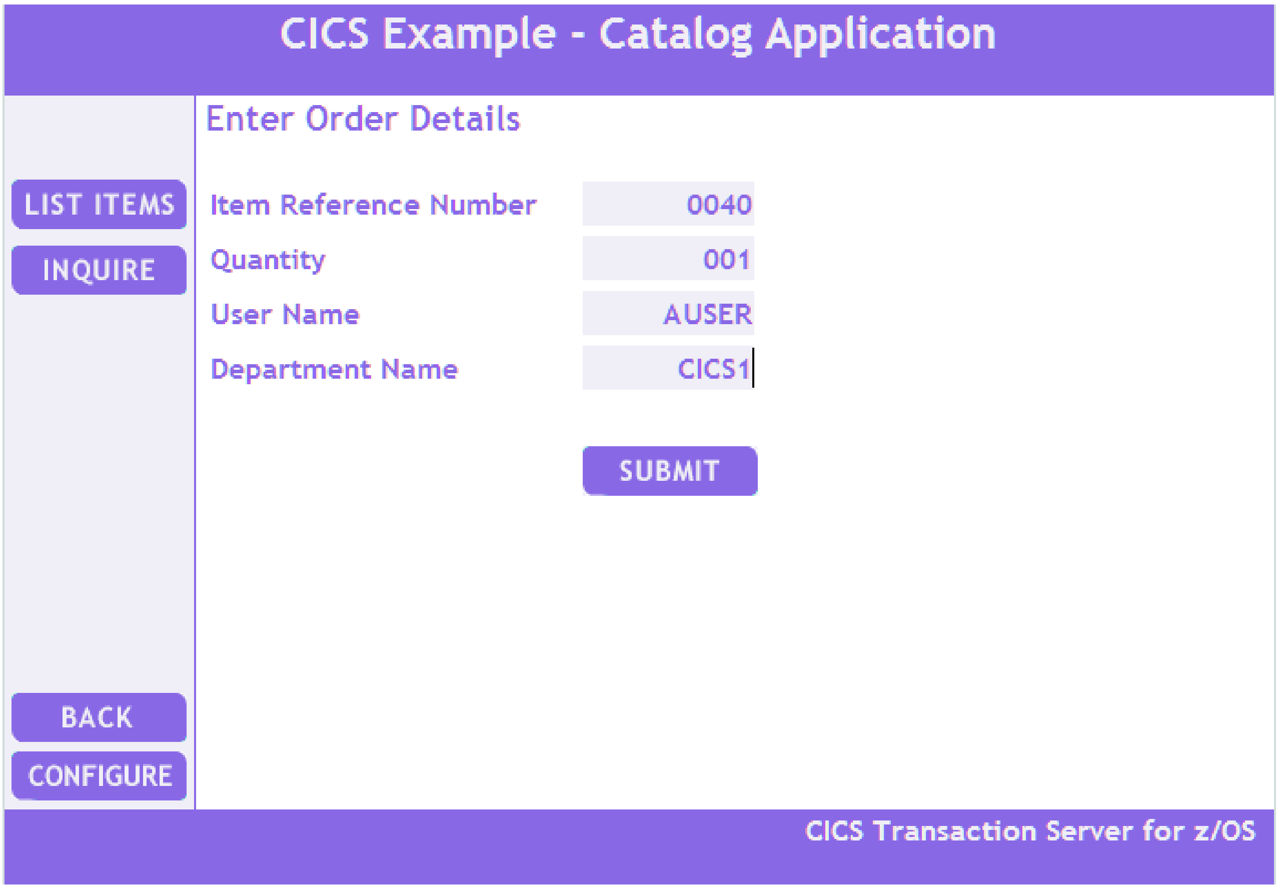

CICS Example - Catalog Application
LIST ITEMS
INQUIRE
ORDER ITEM
BACK
CONFIGURE
SUBMIT
CICS Transaction Server for z/OS

Item Details - Select Item to Place Order

Item	Description	In Stock	On Order	Cost	Select
0010	Ball Pens Black 24pk	13	0	£2.90	○
0020	Ball Pens Blue 24pk	2	50	£2.90	○
0030	Ball Pens Red 24pk	38	0	£2.90	○
0040	Ball Pens Green 24pk	71	0	£2.90	●
0050	Pencil with eraser 12pk	70	0	£1.78	○
0060	Highlighters Assorted 5pk	11	40	£3.89	○
0070	Laser Paper 28-lb 108 Bright 500/ream	90	20	£7.44	○
0080	Laser Paper 28-lb 108 Bright 2500/case	25	0	£33.54	○
0090	Blue Laser Paper 20lb 500/ream	22	0	£5.35	○
0100	Green Laser Paper 20lb 500/ream	3	20	£5.35	○
0110	IBM Network Printer 24 - Toner cart	8	0	£169.56	○
0120	Standard Diary: Week to view 8 1/4x5 3/4	7	0	£25.99	○
0130	Wall Planner: Eraseable 36x24	3	0	£18.85	○
0140	70 Sheet Hard Back wire bound notepad	84	0	£5.89	○
0150	Sticky Notes 3x3 Assorted Colors 5pk	22	45	£5.35	○

4. Select the item that you want to order.
   a. Click the radio button in the **Select** column for the item you want to order.
   b. Click the **SUBMIT** button.

   The application displays the following page:

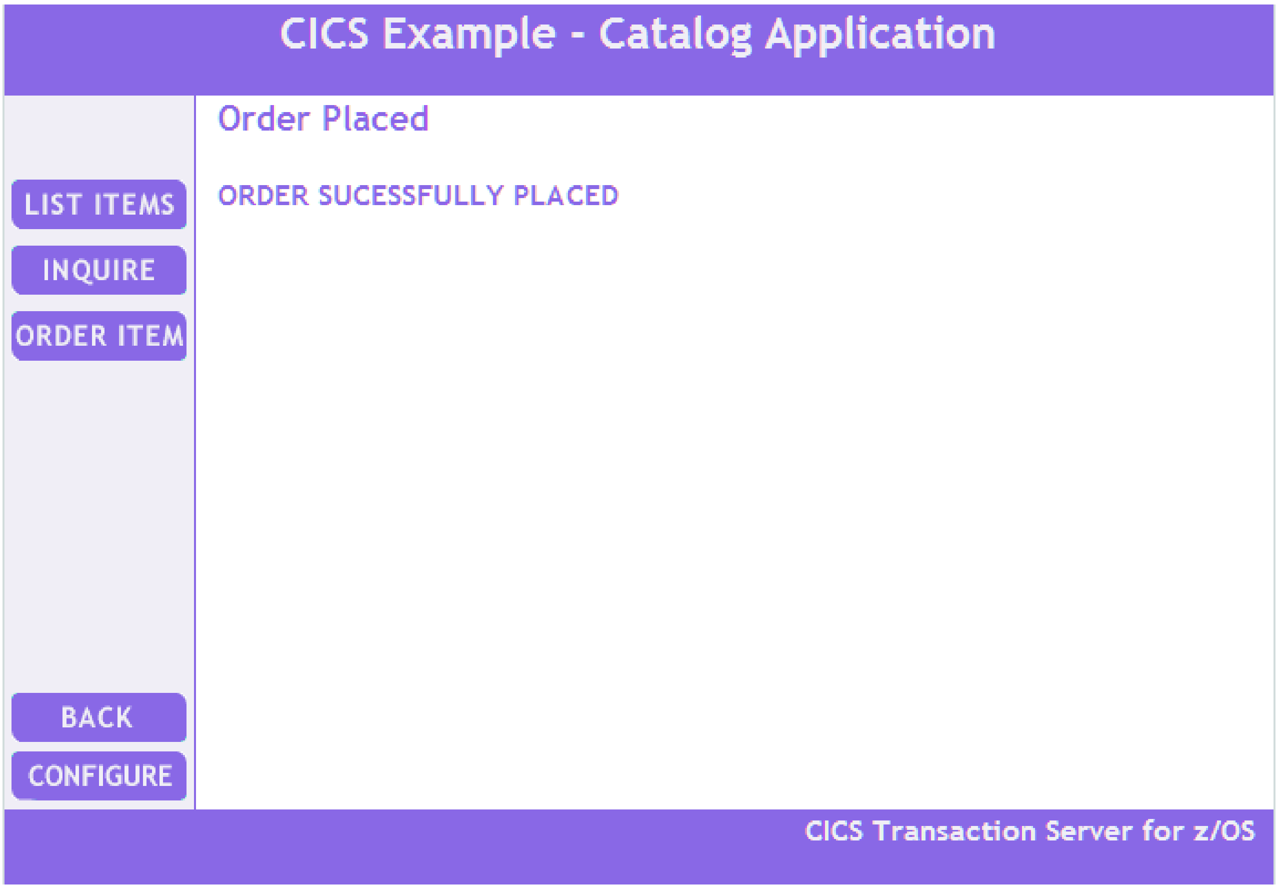

CICS Example - Catalog Application
LIST ITEMS
INQUIRE
BACK
CONFIGURE
SUBMIT
CICS Transaction Server for z/OS

Enter Order Details

Item Reference Number	0040
Quantity	001
User Name	AUSER
Department Name	CICS1

5. To place an order, enter the following information.
   a. Complete the `Quantity` field. Specify the number of items you want to order.
   b. Complete the `User Name` field. Enter a 1 to 8-character string. The base application does not check the value that is entered here.
   c. Complete the `Department Name` field. Enter a 1 to 8-character string. The base application does not check the value that is entered here.
   d. Click the **SUBMIT** button.

   The application displays the following page to confirm that the order has been placed:


CICS Example - Catalog Application
Order Placed
LIST ITEMS
INQUIRE
ORDER ITEM
BACK
CONFIGURE
CICS Transaction Server for z/OS

ORDER SUCESSFULLY PLACED

Using XML Services for the Enterprise to create and deploy CICS Web services artifacts

Web services in CICS provides an interpretive engine that converts XML data to and from data structures. The interpretive engine does not support all the data constructs and types in the COBOL language making it necessary for the CICS Web services developer to write additional code or a wrapper to process unsupported types. The behavior of the interpretive engine is not configurable, whereas a user may have very specific needs in processing SOAP messages.

A standard interface between CICS combined with a user supplied program that provides XML conversion to and from data structures is called the ″Vendor″ interface. The Vendor interface allows users to have pluggable XML conversion. XML converters generated by XML Services for the Enterprise tools have broader support for data constructs and types. We recommend using these XML converters with the Vendor Interface. For improved debugging, CICS Transaction Server Version 3.1 treats the compiled converters as user code which allows debugging should a failure occur. The interpretive engine cannot be debugged or changed by the user.

New in CICS Transaction Server Version 3.1, is a batch job called DFHLS2WS (Language Structure to WSDL) which is equivalent to the bottom-up approach of web services development (see Introduction to XML Services for the Enterprise). XML Services for the Enterprise tools, used in combination with the Vendor interface as a replacement for DFHLS2WS provide expanded functionality to the end user. This combination helps a user to enable web service interface with a COBOL data type that is not supported by the CICS interpretive conversion engine, generally without requiring the user to write any additional wrapper conversion program.

Related concepts

Artifacts necessary to enable a web service under CICS

Related tasks

Installing web services under CICS
Creating artifacts to enable a web service under CICS

Artifacts necessary to enable a web service under CICS

WSBind file

The WSBind file is a resource that describes to CICS the specifics of the web service. For example, it contains information about what the system should do to convert an input XML document to a COBOL data structure and what to do to convert the output COBOL data to the output XML document. XML Services for the Enterprise tools generate the WSBind file for the Vendor interface. For more information on the Vendor interface, see Using XML Services for the Enterprise to create and deploy CICS Web Services artifacts. The WSBind file is an EBCDIC binary file. The extension of the generated WSBind file is always set to **.wsbind**.

XML Services for the Enterprise Converters

In the CICS Vendor scenario, CICS delegates conversion of SOAP requests and response messages to Vendor conversion programs. XML Services for the Enterprise tools generate programs suitable for use with the CICS Vendor interface. These programs consist of a driver and two XML converters (inbound and outbound). The XML converters convert an input XML document to a COBOL data structure and convert the output COBOL data to the output XML document. The driver program manages the communication between CICS and the XML converters.

WSDL

The WSDL file describes the web service to the web service clients. The WSDL file can also be used by the CICS system to validate messages received and sent by the web service. The validation can be turned on and off when the CICS Web service resource is installed or configured. Validation is useful when you test or debug your web service. Validation will slow down the web service performance and you may want to turn it off in the production version of the web service.

Note: Make sure that you use WSDL, WSBind file, and the XML converter set all generated from the same source by XML Services for the Enterprise tools. Do not mix artifacts from other sources or tools. For example do not use the XML converters with the WSDL file or the WSBind file generated by the DFHLS2WS batch job (the CICS Web service assistant tool) that comes with CICS Transaction Server version 3.1.

Related concepts

Introduction to using WebSphere Developer XML Services for the Enterprise

Related tasks

Installing web services under CICS
Creating artifacts to enable a web service under CICS

Creating Web services artifacts for CICS

This section describes the development steps using XML Services for the Enterprise tools to create the needed artifacts to install a new web service in CICS. Specifically, it describes how to use Web services enablement wizard. (You can also achieve these tasks using the Batch Processor).

Locating the CICS application source and copy books

In order to generate the artifacts needed to enable an application as a web service, the Web Services Enablement wizard must have access to either a complete program or copy book containing the data structure that is the interface to application.

Since generated artifacts (the XML converters, the driver, the WSBind file and the WSDL file) must be transferred to a z/OS system, you can use the z/OS projects and system perspectives of WebSphere Developer to assist with this task. Also if your program source and copy books are located on z/OS you can access them using z/OS projects perspective. For more information on z/OS perspectives and projects see Systems, projects, and properties

Create a local project and import the program source files for the CICS program to the project. If the program source files exist on a remote system, use the remote systems explorer to copy them to your local project.

Generating Vendor conversion artifacts

In the navigator view, right click on the program source file containing the interface data structure and select **Enable Web Service** -> **Generate Enablement Components**. A Wizard appears with the first page prompting you to select the inbound and outbound data structures for your application. The first data structure in the file is automatically selected, so take care that this is the correct choice and select **next**.

The second page prompts for properties of the generated artifacts and the type of converter. On the **XML Converter Options** tab select **Web services in CICS** from the Converter type drop down menu. Ensure that the all the codepage entries are set to the codepage of your z/OS system. On the WSDL and XSD options page enter the Endpoint URI for this web service.

Note: The local portion of the URI (excludes server and port) is used as the default for the local URI in the Vendor WSBind (for example, /exampleApp/inquireSingle).

Select **next**.

Next, the vendor WSBind properties page is displayed. If your CICS program communicates via a container, make sure to change the value of the program interface drop down and enter the name of the container that the application expects. The **Advanced properties** tab allows you to specify the needed information for CICS to auto-install your web service. If you do not specify these properties you will have to define them at install time during the manual creation of the web service definitions in CICS. Select **next**.

The next page solicits the locations and names of the generated artifacts in the file system (except for the WSBind whose path is specified on the WSBind properties page). You may elect to generate all of the XML conversion code in one file, while the WSBind, WSDL and XSD must be separate. Deselecting the check box next to and artifact indicates that you do not want to generate that artifact. For example you may not want to generate XSD in this scenario, since you will be using the WSDL only if you run with validation turned on.

Building the XML converters

The XML converters consist of multiple programs that must be compiled and statically linked together with the converter driver program as the main entry point. Using the z/OS projects perspective create a remote project that refers to the target system for your web service. This system requires a version of Enterprise COBOL that supports XML parsing (version 3.1 or later). Copy the XML Converter files to the remote project. Nominate the converter driver as the main entry point, choose to generate JCL that will build the converter load module and store it in a PDS or PDSE. The target PDSE should be in the DFHRPL concatenation of the target CICS region so that CICS can find the load module. Submit the JCL.

Related concepts

Artifacts necessary to enable a web service under CICS
Using XML Services for the Enterprise to create and deploy
CICS
Web Services artifacts
Working with projects
Setting up the data perspective
Systems, projects, and properties

Related tasks

Installing web services under CICS
Creating a z/OS project
Submitting a job and obtaining the output

Related reference

z/OS perspectives
z/OS Projects perspective

Deploying the Web services artifacts to CICS

Setting up a CICS web services provider type PIPELINE

A TCPIPSERVICE resource using the HTTP protocol and listening on the desired port must be created and installed. First, create a PIPELINE resource that uses the previously created TCPIPRESOURCE. Within the PIPELINE resource definition, the WSDir or ″pickup″ directory must be defined which enables auto-install of web services directly from WSBind files. You can find detailed information on setting up a provider type PIPELINE in the CICS 3.1 documentation.

Moving generated artifacts to the host system

While building the XML converters, we deployed the XML converter load module to the host system. You now need to transmit the remaining artifacts, the WSBind and WSDL to the WSDir or ″pickup″ directory for the CICS PIPELINE under which the web service will be installed. The ″pickup″ directory exists in an HFS on the target system. Both the WSBind and WSDL files are sensitive to codepage translation. Since the WSBind is in EBCDIC and the WSDL declares a UTF-8 encoding declaration, you must transmit these files in binary mode to the host system.

Auto-installing the web service

After having transmitted the WSBind and WSDL to the PIPELINE pickup directory you may do an auto-install if all of the fields on the advanced tab of the WSBind properties page in WebSphere Developer are completed correctly. You may then issue a

```
CEMT PERFORM PIPELINE(pipelinename) SCAN
```

If this completes successfully you should see a new WEBSERVICE resource created by doing a

```
CEMT INQUIRE WEBSERVICE(*)
```

The name of the WEBSERVICE is derived from the first 31 characters of the WSBind file name. If you do a

```
CEMT INQUIRE URIMAP(*)
```

you will also see that a URIMAP resource is automatically created. The URIMAP resource maps a local URI to WEBSERVICE resource. By default full WSDL validation is turned off (for performance reasons). To turn it on you may do a

```
CEMT SET WEBSERVICE(webservicename)
```

and change ″novalidation″ to ″validation″. Doing this causes CICS to use the provided WSDL to do full validation of SOAP requests and responses related to this particular WEBSERVICE resource. The location of the WSDL that CICS uses for validation is visible when viewing a WEBSERVICE resource. If the WSDL specified in the WSBind file is not found at the expected location in the filesystem, the WSDL entry in the WEBSERVICE resource will be empty or blank.

Manually installing the web service

The manual install is recommended for cases where it is not possible to know all of the necessary details to populate the advanced tab of the WSBind properties page in WebSphere Developer. The WSBind and WSDL need to be moved to the pickup directory. The details of creating URIMAP and WEBSERVICE resources manually are explained in depth in the CICS documentation.

Related concepts

Artifacts necessary to enable a web service under CICS
Using XML Services for the Enterprise to create and deploy
CICS
Web Services artifacts

Related tasks

Creating artifacts to enable a web service under
CICS

Enabling the catalog example for web services using WebSphere Developer

You can use XML Services for the Enterprise tools in WebSphere Developer to generate XML conversion artifacts which with the CICS Vendor interface provide conversion services for request and response SOAP messages for the catalog manager COBOL application.

In this section we will focus on enabling one of the functions of the base application as a web service. This web service (named inquireSingle) will provide a web service client with the ability to query individual items in the catalog.

Related concepts

Batch Processor

Related tasks

Creating a web service interface with the Web Services Enablement wizard

Generating web services artifacts for the inquireSingle web service

The copy book containing the combined interface definition for the catalog manager is located in the CICS samples dataset with member name `DFH0XCP1`. The inquireSingle web service uses a subset of the interface definition which can be found in a separate copy book `DFH0XCP4`.

You can run the Enable Web Service Wizard against either of these copy books, keeping in mind that if you choose to go with DFH0XCP1 you should make sure on the first page of the wizard in the data structure selection view to select the redefinition `CA-INQUIRE-SINGLE REDEFINES CA-REQUEST-SPECIFIC` and all of its child elements.

Note: filler items will not be visible

One of the benefits of using XML services for the Enterprise tools is support for redefines. You do not have to factor out your interface definition into separate copy books for each redefinition.

XML services for the Enterprise tools expect that all COBOL data structures imported into the Enable Web Service wizard have a level 01 containing element declared. You must add the declaration 01 DFH0XCMN to the top of whichever copy book you decide to use. The name of the level 01 is important in this case so that the root element name used by the XML converters matches what is expected by the CICS Web Client for the catalog application.

Refer to Locating CICS application source and/or copybooks on how to access the copy books mentioned above in the WebSphere Developer workspace. Once the source for the copy books is in the workspace refer to Generating Vendor conversion artifacts for detail on how to generate the the XML converters, driver, and WSBind and WSDL files, using the following inputs to the Enable Web Service Wizard.

To access the Enable Web Services wizard, [Right Click on `DFH0XCP1.cbl` and select **Enable Web Service** -> **Generate enablement components**.

Enter the following values (If an input is not specified, accept the default):

Data Structures page
- **Inbound data structure** : DFH0XCMN
- **Redefinition** : INQUIRE-SINGLE
- **Outbound data structure** : DFH0XMCN
- **Redefinition** : INQUIRE-SINGLE

Generation Options page
- **Converter type** : ″Web Services for CICS″
- **Business program name** : ″DFH0XCMN″
- **Inbound Codpage** : Codepage of your CICS system
- **Host Codepage** : Codepage of your CICS system
- **Outbound Codepage** : Codepage of your CICS system
- **Endpoint URI**: ″http://yourserver:yourport/exampleApp/inquireSingle″

Web Services for CICS page
- **WSBind file name** : ″inquireSingle″
- **PIPELINE name** : ″EXPIPE01″
- **WSDL HFS file path** : ″/u/exampleapp/wsbind/inquireSingle.wsdl″

File, Dataset or Member Selection page
- **Converter driver file name** : ″DFHXCP4D″

- **Inbound Converter file name** : ″DFHXCP4I″
- **Outbound Converter file name** : ″DFHXCP4O″
- **WSDL file name** : ″inquireSingle″

Related tasks

Creating a web service interface with the Web Services Enablement wizard

Installing and deploying the InquireSingle web service

After generating the converters in Generating web services artifacts for the inquireSingle web service, refer to Building the XML converters to build the converters and deploy the load module to host system. Next refer to Moving generated artifacts to the host system to deploy the WSBind and WSDL files to the ″EXPIPE01″ PIPELINE WSDir (pickup directory). Finally refer to Auto-installing the web service to install the new web service and check if it is in service. You may then proceed to invoke the inquireSingle web service from the web client.

Related tasks

Configuring the Web client

Combined Interface Definition DFH0XCP1

```
>>     01 DFH0XCMN.

     *    Catalogue COMMAREA structure
         03 CA-REQUEST-ID           PIC X(6).
           03 CA-RETURN-CODE          PIC 9(2).
           03 CA-RESPONSE-MESSAGE     PIC X(79).
           03 CA-REQUEST-SPECIFIC     PIC X(911).
      *    Fields used in Inquire Catalog
           03 CA-INQUIRE-REQUEST REDEFINES CA-REQUEST-SPECIFIC.
               05 CA-LIST-START-REF       PIC 9(4).
               05 CA-LAST-ITEM-REF        PIC 9(4).
               05 CA-ITEM-COUNT           PIC 9(3).
               05 CA-INQUIRY-RESPONSE-DATA PIC X(900).
               05 CA-CAT-ITEM  REDEFINES CA-INQUIRY-RESPONSE-DATA
                               OCCURS 15 TIMES.
                   07 CA-ITEM-REF         PIC 9(4).
                   07 CA-DESCRIPTION      PIC X(40).
                   07 CA-DEPARTMENT       PIC 9(3).
                   07 CA-COST             PIC X(6).
                   07 IN-STOCK            PIC 9(4).
                   07 ON-ORDER            PIC 9(3).
      *    Fields used in Inquire Single
           03 CA-INQUIRE-SINGLE REDEFINES CA-REQUEST-SPECIFIC.
               05 CA-ITEM-REF-REQ         PIC 9(4).
               05 FILLER                  PIC 9(4).
               05 FILLER                  PIC 9(3).
               05 CA-SINGLE-ITEM.
                   07 CA-SNGL-ITEM-REF    PIC 9(4).
                   07 CA-SNGL-DESCRIPTION  PIC X(40).
                   07 CA-SNGL-DEPARTMENT  PIC 9(3).
                   07 CA-SNGL-COST        PIC X(6).
                   07 IN-SNGL-STOCK       PIC 9(4).
                   07 ON-SNGL-ORDER       PIC 9(3).
               05 FILLER                  PIC X(840).
      *    Fields used in Place Order
           03 CA-ORDER-REQUEST REDEFINES CA-REQUEST-SPECIFIC.
               05 CA-USERID               PIC X(8).
               05 CA-CHARGE-DEPT          PIC X(8).
```

```
    05 CA-ITEM-REF-NUMBER       PIC 9(4).
    05 CA-QUANTITY-REQ          PIC 9(3).
    05 FILLER                   PIC X(888).
```

Components of the base application

Table 6. Application components

Memebr name in SDFHSAMP	Type	Comment
DFH0XCMN	Cobol Source	Source code for the catalog manager application
DFH0XVDS	Cobol Source	Source code for the VSAM data-store module
DFH0XSDS	Cobol Source	Source code for the stubbed data-store module
DFH0XSOD	Cobol Source	Source code for the stubbed version of the order dispatch module
DFH0XWOD	Cobol Source	Source code for the order dispatch module that makes an outbound Web service request
DFH0XODE	Cobol Source	Source code for the stubbed version of the order dispatch endpoint
DFH0XSSM	Cobol Source	Source code for the stubbed version of the stock manager module
DFH0XGUI	Cobol Source	Source code for the BMS interface controller application
DFH0XCFG	Cobol Source	Source code for the application configuration module
DFH0XCP1	Copybook	Cobol copybook definition for catalog manager inquire and place order operations
DFH0XCP2	Copybook	Cobol copybook definition for dispatch order and stock manager operations
DFH0XCP3	Copybook	Cobol copybook definition for the inquire list operation
DFH0XCP4	Copybook	Cobol copybook definition for the inquire single operation
DFH0XCP5	Copybook	Cobol copybook definition for the place order operation
DFH0XCP6	Copybook	Cobol copybook definition for the dispatch order operation
DFH0XCP7	Copybook	Cobol copybook definition that is mapped to a SOAP request for the dispatch order operation
DFH0XCP8	Copybook	Cobol copybook definition that is mapped to a SOAP response for the dispatch order operation
DFH0XM1	Copybook	Cobol copybook for BMS interface
DFH0XM2U	Copybook	Customized Cobol copybook for BMS inquire interface

Table 6. Application components (continued)

Memebr name in SDFHSAMP	Type	Comment
DFH0XM3	Copybook	Cobol copybook for BMS interface to configuration module
DFH0XS1	BMS Mapset	BMS Mapset for application user interface
DFH0XS2	BMS Mapset	BMS Mapset for the inquire operation of the application user interface
DFH0XS3	BMS Mapset	BMS Mapset for the configuration module

Table 7. CICS Resource Definitions

Resource name	Resource type	Comment
EXAMPLE	CICS Resource definition group	CICS resource definitions required for the example application
EGUI	TRANSACTION	Transaction to invoke program DFH0XGUI to start the BMS interface to the application (Customizable)
ECFG	TRANSACTION	Transaction to invoke the program DFH0XCFG to start the example configuration BMS interface (Customizable)
EXMPCAT	FILE	File definition of the EXMPCAT VSAM file for the application catalog (Customizable)
EXMPCONF	FILE	File definition of the EXMPCONF application configuration file.

The catalog manager program

The catalog manager is the controlling program for the business logic of the example application, and all interactions with the example application pass through it.

To ensure that the program logic is simple, the catalog manager performs only limited type checking and error recovery.

The catalog manager supports a number of operations. Input and output parameters for each operation are defined in a single data structure, which is passed to and from the program in a COMMAREA.

COMMAREA structures

```
*    Catalogue COMMAREA structure
           03 CA-REQUEST-ID            PIC X(6).
           03 CA-RETURN-CODE           PIC 9(2).
           03 CA-RESPONSE-MESSAGE      PIC X(79).
           03 CA-REQUEST-SPECIFIC      PIC X(911).
      *    Fields used in Inquire Catalog
           03 CA-INQUIRE-REQUEST REDEFINES CA-REQUEST-SPECIFIC.
               05 CA-LIST-START-REF        PIC 9(4).
               05 CA-LAST-ITEM-REF         PIC 9(4).
               05 CA-ITEM-COUNT            PIC 9(3).
               05 CA-INQUIRY-RESPONSE-DATA PIC X(900).
               05 CA-CAT-ITEM  REDEFINES CA-INQUIRY-RESPONSE-DATA
                               OCCURS 15 TIMES.
                   07 CA-ITEM-REF          PIC 9(4).
                   07 CA-DESCRIPTION       PIC X(40).
               07 CA-DEPARTMENT       PIC 9(3).
               07 CA-COST             PIC X(6).
               07 IN-STOCK            PIC 9(4).
               07 ON-ORDER            PIC 9(3).
      *    Fields used in Inquire Single
           03 CA-INQUIRE-SINGLE REDEFINES CA-REQUEST-SPECIFIC.
               05 CA-ITEM-REF-REQ         PIC 9(4).
               05 FILLER                  PIC 9(4).
               05 FILLER                  PIC 9(3).
               05 CA-SINGLE-ITEM.
                   07 CA-SNGL-ITEM-REF     PIC 9(4).
                   07 CA-SNGL-DESCRIPTION  PIC X(40).
                   07 CA-SNGL-DEPARTMENT   PIC 9(3).
                   07 CA-SNGL-COST         PIC X(6).
                   07 IN-SNGL-STOCK        PIC 9(4).
                   07 ON-SNGL-ORDER        PIC 9(3).
               05 FILLER                  PIC X(840).
      *    Fields used in Place Order
           03 CA-ORDER-REQUEST REDEFINES CA-REQUEST-SPECIFIC.
               05 CA-USERID               PIC X(8).
               05 CA-CHARGE-DEPT          PIC X(8).
               05 CA-ITEM-REF-NUMBER      PIC 9(4).
               05 CA-QUANTITY-REQ         PIC 9(3).
               05 FILLER                  PIC X(888).



 *    Dispatcher/Stock Manager COMMAREA structure
           03 CA-ORD-REQUEST-ID               PIC X(6).
           03 CA-ORD-RETURN-CODE              PIC 9(2).
           03 CA-ORD-RESPONSE-MESSAGE         PIC X(79).
           03 CA-ORD-REQUEST-SPECIFIC         PIC X(23).
      *    Fields used in Dispatcher
           03 CA-DISPATCH-ORDER REDEFINES CA-ORD-REQUEST-SPECIFIC.
               05 CA-ORD-ITEM-REF-NUMBER      PIC 9(4).
               05 CA-ORD-QUANTITY-REQ         PIC 9(3).
               05 CA-ORD-USERID               PIC X(8).
               05 CA-ORD-CHARGE-DEPT          PIC X(8).
      *    Fields used in Stock Manager
           03 CA-STOCK-MANAGER-UPDATE REDEFINES CA-ORD-REQUEST-SPECIFIC.
               05 CA-STK-ITEM-REF-NUMBER      PIC 9(4).
               05 CA-STK-QUANTITY-REQ         PIC 9(3).
               05 FILLER                      PIC X(16).
```

Return codes

Each operation of the catalog manager can return a number of return codes.

Type	Code	Explanation
General	00	Function completed without error
Catalog file	20	Item reference not found
	21	Error opening, reading, or ending browse of catalog file
	22	Error updating file
Configuration file	50	Error opening configuration file
	51	Data store type was neither STUB nor VSAM
	52	Outbound Web service switch was neither Y nor N

Type	Code	Explanation
Remote Web service	30	The EXEC CICS INVOKE WEBSERVICE command returned an INVREQ condition
	31	The EXEC CICS INVOKE WEBSERVICE command returned an NOTFND condition
	32	The EXEC CICS INVOKE WEBSERVICE command returned a condition other than INVREQ or NOTFND
Application	97	Insufficient stock to complete order
	98	Order quantity was not a positive number
	99	DFH0XCMN received a COMMAREA in which the CA-REQUEST-ID field was not set to one of the following: 01INQC, 01INQS, or 01ORDR

INQUIRE CATALOG operation

This operation returns a list of up to 15 catalog items, starting with the item specified by the caller.

Input parameters

CA-REQUEST-ID
A string that identifies the operation. For the INQUIRE CATALOG command, the string contains “`01INQC`”

CA-LIST-START-REF
The reference number of the first item to be returned.

Output parameters

CA-RETURN-CODE

CA-RESPONSE-MESSAGE
A human readable string, containing “*`num`* `ITEMS RETURNED`” where *`num`* is the number of items returned.

CA-LAST-ITEM-REF
The reference number of the last item returned.

CA-ITEM-COUNT
The number of items returned.

CA-CAT-ITEM
An array containing the list of catalog items returned. The array has 15 elements; if fewer than 15 items are returned, the remaining array elements contain blanks.

INQUIRE SINGLE ITEM operation

This operation returns a single catalog item specified by the caller.

Input parameters

CA-REQUEST-ID
A string that identifies the operation. For the INQUIRE SINGLE ITEM command, the string contains “`01INQS`”

CA-ITEM-REF-REQ
The reference number of the item to be returned.

Output parameters

CA-RETURN-CODE

CA-RESPONSE-MESSAGE
A human readable string, containing `RETURNED ITEM: REF=`*`item-reference`*' where *`item-reference`* is the reference number of the returned item.

CA-SINGLE-ITEM
An array containing in its first element the returned catalog item.

PLACE ORDER operation

This operation places an order for a single item. If the required quantity is not available a message is returned to the user. If the order is successful, a call is made to the Stock Manager informing it what item has been ordered and the quantity ordered.

Input parameters

CA-REQUEST-ID
A string that identifies the operation. For the PLACE ORDER operation, the string contains `'01ORDR'`

CA-USERID
An 8-character user ID which the application uses for dispatch and billing.

CA-CHARGE-DEPT
An 8-character department ID which the application uses for dispatch and billing.

CA-ITEM-REF-NUMBER
The reference number of the item to be ordered.

CA-QUANTITY-REQ
The number of items required.

Output parameters

CA-RETURN-CODE

CA-RESPONSE-MESSAGE
A human readable string, containing `'ORDER SUCCESSFULLY PLACED'`.

DISPATCH STOCK operation

This operation places a call to the stock dispatcher program, which in turn dispatches the order to the customer.

Input parameters

CA-ORD-REQUEST-ID
A string that identifies the operation. For the DISPATCH ORDER operation, the string contains `'01DSPO'`

CA-ORD-USERID
An 8-character user ID which the application uses for dispatch and billing.

CA-ORD-CHARGE-DEPT
An 8-character department ID which the application uses for dispatch and billing.

CA-ORD-ITEM-REF-NUMBER
The reference number of the item to be ordered.

CA-ORD-QUANTITY-REQ
The number of items required.

Output parameters

CA-ORD-RETURN-CODE

NOTIFY STOCK MANAGER operation

This operation takes details of the order that has been placed to decide if stock replenishment is necessary.

Input parameters

CA-ORD-REQUEST-ID
A string that identifies the operation. For the NOTIFY STOCK MANAGER operation, the string contains `'01STKO'`

CA-STK-ITEM-REF-NUMBER
The reference number of the item to be ordered.

CA-STK-QUANTITY-REQ
The number of items required.

Output parameters

CA-ORD-RETURN-CODE

BMS presentation manager

The presentation manager is responsible for all interactions with the end user via 3270 BMS panels. No business decisions are made in this program.

Data handler

The data handler provides the interface between the catalog manager and the data store.

The example application provides two versions of the data handler:

- The first version uses a VSAM file as the data store.
- The second version is a dummy program that always returns the same data on an inquire and does not store the results of any update requests.

Dispatch manager

The dispatch manager is responsible for dispatching the order to the customer once the order has been confirmed.

The example application provides two versions of the dispatch manager program:

- The first version is a dummy program that returns a correct response to the caller, but takes no other action.
- The second version is a Web service requester program that makes a request to the endpoint address defined in the configuration file.

Order dispatch endpoint

The order dispatch program is a Web service provider program that is responsible for dispatching the item to the customer.

In the example application, the order dispatcher is a dummy program that returns a correct response to the caller, but takes no other action. It makes it possible for all configurations of the example Web services to be operable.

Stock manager

The stock manager is responsible for managing the replenishment of the stock.

In the example program, the stock manager is a dummy program that returns a correct response to the caller, but takes no other action.

Application configuration

The example application includes a program that lets you configure the base application.

File Structures and Definitions

The example application uses two VSAM files: the catalog file which contains the details of all items stocked and their stock levels, and the configuration file which holds user-selected options for the application.

Catalog file

The catalog file is a KSDS VSAM file which contains all information relating to the product inventory.

Records in the file have the following structure:

Name	COBOL data type	Description
WS-ITEM-REF-NUM	PIC 9(4)	Item reference number
WS-DESCRIPTION	PIC X(40)	Item description
WS-DEPARTMENT	PIC 9(3)	Department identification number
WS-COST	PIC ZZZ.99	Item price
WS-IN-STOCK	PIC 9(4)	Number of items in stock
WS-ON-ORDER	PIC 9(3)	Number of items on order

Configuration file

The configuration file is a KSDS VSAM file which contains information used to configure the example application.

The configuration file is a KSDS VSAM file with 3 distinct records.

Table 8. General information record

Name	COBOL data type	Description
PROGS-KEY	PIC X(9)	Key field for the general information record, containing ’EXMP-CONF’
filler	PIC X	
DATASTORE	PIC X(4)	A character string that specifies the type of data store program to be used. Values are: ’STUB’ ’VSAM’
filler	PIC X	

Table 8. General information record (continued)

Name	COBOL data type	Description
DO-OUTBOUND-WS	PIC X	A character that specifies whether the dispatch manager is make an outbound Web service request. Values are: `'Y'` `'N'`
filler	PIC X	
CATMAN-PROG	PIC X(8)	The name of the catalog manager program
filler	PIC X	
DSSTUB-PROG	PIC X(8)	The name of the dummy data handler program
filler	PIC X	
DSVSAM-PROG	PIC X(8)	The name of the VSAM data handler program
filler	PIC X	
ODSTUB-PROG	PIC X(8)	The name of the dummy order dispatcher module
filler	PIC X	
ODWEBS-PROG	PIC X(8)	The name of the outbound Web service order dispatcher program
filler	PIC X	
STKMAN-PROG	PIC X(8)	The name of the stock manager program
filler	PIC X(10)	

Table 9. Outbound URL record

Name	COBOL data type	Description
URL-KEY	PIC X(9)	Key field for the general information record, containing `'OUTBNDURL'`
filler	PIC X	
OUTBOUND-URL	PIC X(255)	Outbound URL for the order dispatcher Web service request

Table 10. Catalog file information

Name	COBOL data type	Description
URL-FILE-KEY	PIC X(9)	Key field for the general information record, containing `'VSAM-NAME'`
filler	PIC X	
CATALOG-FILE-NAME	PIC X(8)	Name of the CICS FILE resource used for the catalog file

Chapter 10. XML Sevices for the Enterprise tools and IMS SOAP Gateway

Overview of IMS SOAP Gateway

IMS SOAP Gateway enables IMS applications to be exposed as Web services.

The IMS SOAP Gateway is a lightweight, XML-based connectivity solution that enables IMS applications to inter-operate outside of the IMS environment through SOAP to provide and request services independently of platform, environment, application language, or programming model.

You can enable IMS COBOL applications for Web services by using the XML Services for the Enterprise tools to generate Web service artifacts for IMS COBOL applications. You then deploy these Web service artifacts to the IMS SOAP Gateway to make an IMS application available as a Web service. Different types of client applications, such as Microsoft® .NET, Java™, and third-party applications, can then submit SOAP requests into IMS to drive the business logic of the COBOL applications.

The XML Services for the Enterprise tools generates the Web service artifacts that allow SOAP client access to the existing IMS applications without requiring modification to the applications. From a COBOL copybook that describes the input and output message format, the XML Services for the Enterprise tool generates the following Web service artifacts:

- COBOL converters and driver file, which is a single file that contains the input message COBOL converter, the output message COBOL converter, and a driver.
- Correlator file, which contains information that enables IMS SOAP Gateway to set IMS properties and call the IMS application.
- Web Services Description Language (WSDL) file, which describes the Web service interface of the IMS application so that the client can communicate with the Web service.

The IMS SOAP Gateway can assist an organization in the following areas:

- Enterprise modernization
- Application development
- Business integration
- Web services implementation

For more information about using the IMS SOAP Gateway, including samples, see http://www.ibm.com/software/data/ims/soap/

Enabling IMS applications for Web services with IMS SOAP Gateway

This topic provides only an overview of the tasks required to enable IMS applications for Web services.

To use IMS SOAP Gateway to create Web services from existing IMS applications, you must have IMS and IMS Connect configured properly. Fore more information about completing the tasks described in this topic, see http://www.ibm.com/software/data/ims/soap/.

To enable IMS applications for Web services:

1. Install IMS SOAP Gateway.
2. Configure IMS Connect for IMS SOAP Gateway.
3. Generate the Web services artifacts for an IMS application by using the XML Services for the Enterprise tools. See “Generating Web services artifacts for IMS SOAP Gateway” on page 122.

4. Deploy the Web service to IMS SOAP Gateway.
5. Copy and compile artifacts in IMS Connect.
6. Create the client application.
7. Run the client application.

Generating Web services artifacts for IMS SOAP Gateway

Use the XML Services for the Enterprise tools to generate the artifacts that are needed to enable existing IMS COBOL applications for Web services.

To generate the artifacts that are needed to enable existing IMS COBOL applications for Web services, you must have a COBOL copybook that describes the format of the input and output messages for the application.

This task is part of the larger task of enabling IMS applications for Web services. Information about enabling IMS applications for Web services, is available from the IMS SOAP Gateway Web site at http://www.ibm.com/software/data/ims/soap/. To generate Web services artifacts for IMS SOAP Gateway:

1. Create a project.
2. Import the COBOL copybook that describes the format of the input and output messages.
3. Start the Enable Web Service wizard:
   a. Right-click on the COBOL copybook file.
   b. Select **Enable Web Services** → **Generate enablement code**.
4. Select the data structures for the inbound and outbound converters:
   a. Click **Change COBOL Options**. The COBOL Import Properties panel displays.
   b. In the **Platform** field, select z/OS.
   c. Click **Finish**. The Data structures panel displays.
   d. For the inbound data structure, select the COBOL data structure that corresponds to the input message of the IMS application.
   e. Select the Outbound data structure tab.
   f. Select the COBOL data structure that corresponds to the output message of the IMS application.
   g. Click **Next** to continue.
5. Specify generation options:
   a. In the **Converter type** field, select IMS SOAP Gateway.
   b. In the **Host code page** field, select the code page that the host uses. IMS SOAP Gateway supports only UTF-8 encoding for the inbound and outbound code pages. Therefore, you cannot change these settings.
   c. Specify any additional properties.
   d. Select the WSDL and XSD Options tab.
   e. In the **Endpoint URI** field, change the host and port name to the location of IMS SOAP Gateway. This URI specifies the address of the Web service.
   f. Specify any additional properties.
   g. Click **Next** to continue.
6. Specify the IMS SOAP Gateway correlator properties and click **Next**.
7. Specify location and names of the Web service artifacts.
   a. If necessary, change the default location and names of the COBOL converters and driver.
   b. Ensure that **Generate all to driver** is selected.
   c. Select the WSDL and XSD tab.

d. If necessary, change the default location and names of the WSDL file.
e. Ensure that **WSDL file name** is selected.
f. Optionally, select the inbound and outbound XSD files to be generated. These files are not required by IMS SOAP Gateway.
g. Click **Finish**.

The following files are generated:
- COBOL converters and driver file
- Correlator file
- WSDL file
- Inbound and outbound XSD files (optional)

After you create the Web services artifacts, you can deploy the Web service to IMS SOAP Gateway.