CICS Transaction Server for z/OS

CICS Web サービス・ガイド

バージョン 3 リリース 1

SD88-6383-00
(英文原典：SC34-6458-00)

CICS Transaction Server for z/OS

CICS Web サービス・ガイド

バージョン 3 リリース 1

SD88-6383-00
(英文原典：SC34-6458-00)

お願い

本書および本書で紹介する製品をご使用になる前に、 179 ページの『特記事項』に記載されている情報をお読みください。

本書は、CICS Transaction Server for z/OS (プログラム番号 5655-M15) バージョン 3、リリース 1、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原　典： SC34-6458-00
CICS Transaction Server for z/OS
CICS Web Services Guide
Version 3 Release 1

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2005.3

この文書では、平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財) 日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

前書き

本書の内容

本書では、CICS での Web サービスの使用方法について説明します。

本書の対象読者

本書の対象読者は次のとおりです。

- Web サービス環境での CICS アプリケーションの配置を検討している計画担当者および設計者。
- Web サービスをサポートするために CICS の構成を担当しているシステム・プログラマー。
- Web サービス環境で配置されるアプリケーションを担当するアプリケーション・プログラマー。

第 1 章 CICS および Web サービス

Web サービスは、ワールド・ワイド・ウェブ (WWW) がプログラムとエンド・ユーザー間の対話に果たしてきた役割を、プログラム相互間の対話に提供できます。Web サービスを利用することにより、アプリケーションの統合をかつてないほど迅速、容易、かつ低コストで実現できます。

CICS Transaction Server for z/OS は、以下に示すような Web サービスの包括的なサポートを提供します。

- CICS アプリケーションは、サービス・リクエスター、サービス・プロバイダー、またはその両方として異種の Web サービス環境に関与できます。
- HTTP および MQ のサポート
- CICS Transaction Server for z/OS には、CICS Web サービス・アシスタントが組み込まれています。これは、WSDL サービス記述を高水準プログラム言語のデータ構造にマップしたり、その*逆方向に* マップしたりするときに役立つ 1 組のユーティリティー・プログラムです。ユーティリティー・プログラムは、以下のプログラム言語をサポートしています。

  COBOL

  PL/I

  C

  C++
- Web サービスの CICS サポートは、以下のようなオープン・スタンダードに準拠しています。

  SOAP 1.1 および 1.2

  HTTP 1.1

  WSDL 1.1
- Web サービスの CICS サポートは、Web Services Interoperability Organization (WS-I) Basic Profile 1.0 に準拠することにより、その他の Web サービス実装環境との間で最大限のインターオペラビリティーを確保できます。プロファイルは、非専有の一連の Web サービス仕様であり、これらの仕様の説明および改訂も付記されています。これらを総合することにより、Web サービスのさまざまな実装環境間でインターオペラビリティーを実現することができます。

Web サービスとは

Web サービスとは、ネットワークを介して相互に運用可能なマシン間の対話をサポートする目的で設計されたソフトウェア・システムのことです。 Web サービスは、マシン処理可能な形式 (特に、Web サービス記述言語 (WSDL)) で記述されているインターフェースを持ちます。

Web サービスでは、1 つまたは一連の特定のタスクが実行されます。 Web サービスは、XML のサービス記述と呼ばれる、標準的で正式な XML の概念を使用して記述されます。このサービス記述は、メッセージ・フォーマット (操作の詳細を記述)、トランスポート・プロトコル、場所など、サービスとの対話に必要な詳細をすべて提供します。

インターフェースの性質上、Web サービスの実装詳細は表示されません。これにより、Web サービス実装先のハードウェアまたはソフトウェアのプラットフォームや、記述に使用したプログラム言語とは独立して使用できるようになります。

この結果、Web サービス・ベースのアプリケーションによる、疎結合状態でコンポーネント指向のテクノロジー相互実装が可能になり、促進されます。Web サービスは、複雑な集計またはビジネス・トランザクションを実行するために、単独または他の Web サービスと組み合わせて使用できます。

Web サービスがビジネスに役立つ仕組み

Web サービスは、Web を介してビジネス機能の配置、およびビジネス機能へのアクセスを提供するテクノロジーです。 Web サービスを利用することにより、アプリケーションの統合をかつてないほど迅速、容易、かつ低コストで実現できます。

Web サービスは、以下の点でビジネスに役立つことができます。

- ビジネス実施コストの削減
- ソリューション配置の高速化
- 新規機会の開拓

これらすべてを達成するための鍵は、HTTP、XML、SOAP、WSDL などの既存および先進の標準を基に作成される共通のプログラム間通信モデルです。

CICS が Web サービスをサポートすることにより、再プログラミングの労力を最小限に抑えながら、既存のアプリケーションを新しい方法で配置することが可能になります。

Web サービス用語

Extensible Markup Language (XML)
文書マークアップの標準の 1 つで、単純で人間が読めるタグでデータをマークアップするための汎用構文。この規格は、World Wide Web Consortium (W3C) に公認されている。

最初の SOAP 送信側
SOAP メッセージ・パスの開始点で SOAP メッセージを発信する SOAP 送信側

サービス・プロバイダー
Web サービス機能を提供するソフトウェアの集合。

サービス・プロバイダー・アプリケーション
サービス・プロバイダーで使用されるアプリケーションのこと。通常、サービス・プロバイダー・アプリケーションは、サービス・プロバイダーのビジネス・ロジック・コンポーネントを提供する。

サービス・リクエスター
サービス・プロバイダーから Web サービスを要求する役割を持つソフトウェアの集合。

サービス・リクエスター・アプリケーション
サービス・リクエスターで使用されるアプリケーション。通常、サービス・

リクエスター・アプリケーションは、サービス・リクエスターのビジネス・ロジック・コンポーネントを提供する。

Simple Object Access Protocol
SOAP を参照。

SOAP 以前は、Simple Object Access Protocol の頭字語。非集中の分散環境で情報を交換するための単純なプロトコル。これは、次の 3 つの部分から構成される XML ベースのプロトコルである。

- メッセージの内容およびその処理方法を記述するためのフレームワークを定義するエンベロープ。
- アプリケーション定義のデータ・タイプのインスタンスを表現するための一連のエンコード規則。
- リモート・プロシージャー・コールおよび応答を表現するための規則。

SOAP は、HTTP などの他のプロトコルと併用できる。

SOAP 中間ノード
SOAP 受信側と SOAP 送信側の両方の機能を備え、SOAP メッセージの内部から宛先を指定できる SOAP ノード。 SOAP 中間ノードは、自身を宛先とする SOAP ヘッダー・ブロックを処理し、最終の SOAP 受信側に向けて SOAP メッセージを転送する役割を果たす。

SOAP メッセージ・パス
1 つの SOAP メッセージが通過する一連の SOAP ノードのこと。これには、最初の SOAP 送信側、SOAP 中間ノード (存在しないか 1 つ以上)、最終の SOAP 受信側が含まれる。

SOAP ノード
SOAP メッセージ上で動作する処理ロジック。

SOAP 受信側
SOAP メッセージを受信する SOAP ノード。

SOAP 送信側
SOAP メッセージを送信する SOAP ノード。

最終の SOAP 受信側
SOAP メッセージの最終の宛先となる SOAP 受信側のこと。SOAP 本体およびこれを宛先とするすべての SOAP ヘッダー・ブロックの内容を処理する役割を果たす。

UDDI Universal Description, Discovery and Integration

Universal Description, Discovery and Integration
Universal Description, Discovery and Integration (UDDI) とは、Web サービスに関する Web ベースの分散情報レジストリーの仕様のこと。 UDDI は、企業が自社提供の Web サービスに関する情報を登録できる仕様の一連の実装形態でもあり、これによって他の企業はこの企業の情報を検索できる。

Web サービス
ネットワークを介した相互に運用可能なマシン間の対話をサポートする目的

で設計されたソフトウェア・システムのこと。マシン処理可能な形式 (特に、Web サービス記述言語 (WSDL)) で記述されているインターフェースがある。

Web サービス・バインディング・ファイル
WEBSERVICE リソースと関連付けられているファイルで、さらに入出力メッセージとアプリケーション・データ構造との間でデータをマップするために CICS が使用する情報が格納されているファイルのこと。

Web サービス記述
サービス・プロバイダーが Web サービスをサービス・リクエスターに呼び出すために仕様をやり取りする場合の手段となる XML 文書。Web サービス記述は、Web サービス記述言語 (WSDL) で記述される。

Web サービス記述言語
Web サービスを記述するための XML アプリケーション。サービスによって提供された抽象機能の記述と、この機能が提供される仕組みや条件など、サービスの具体的な詳細とを分離する目的で設計された。

WSDL Web サービス記述言語。

XML Extensible Markup Language。

XML ネーム・スペース
URI 参照によって識別される一まとまりの名前で、エレメント・タイプおよび属性名として XML 文書内で使用される。

XML スキーマ
構造を記述し、他の XML 文書の内容を制約する XML 文書。

XML スキーマ定義言語
XML スキーマを記述するための XML 構文。World Wide Web Consortium (W3C) によって推奨されている。

第 2 章 Web サービスのアーキテクチャー

Web サービスのアーキテクチャーは、サービス・プロバイダー、サービス・リクエスター、およびオプションのサービス・レジストリーの 3 つのコンポーネント間の対話が基本になっています。

サービス・プロバイダー

Web サービス機能を提供するソフトウェアの集合。内訳は次のとおりです。

- アプリケーション・プログラム
- ミドルウェア
- これらが稼働するプラットフォーム

サービス・リクエスター

サービス・プロバイダーから Web サービスを要求する役割を持つソフトウェアの集合。内訳は次のとおりです。

- アプリケーション・プログラム
- ミドルウェア
- これらが稼働するプラットフォーム

サービス・レジストリー

サービス・プロバイダーが提供するサービスの記述をサービス・プロバイダー自身が発行する場所で、さらにサービス・リクエスターがそのサービスを見つける場所。

このレジストリーは、Web サービス・アーキテクチャーのオプションのコンポーネントです。サービス・リクエスターおよびサービス・プロバイダーは、レジストリーなしで通信できる状況が多数存在するためです。例えば、サービスを提供する組織がサービスのユーザーにサービス記述を直接配布する場合、E メールの添付ファイル、FTP サイトからのダウンロード、場合によっては CD-ROM の配布などの方法が可能です。

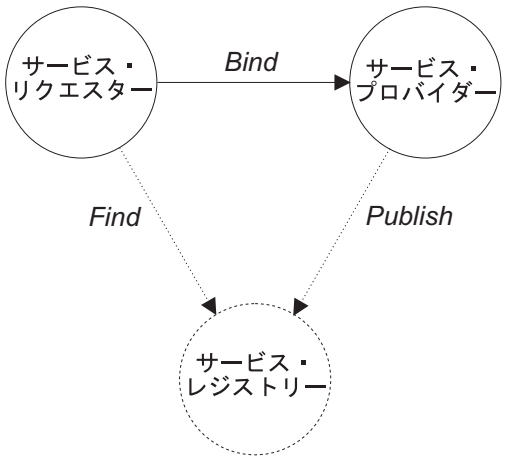


図 1. Web サービスのコンポーネントおよび対話

CICS は、リクエスター・コンポーネントとプロバイダー・コンポーネントを実装するために直接サポートします。サービス・レジストリーをCICS に配置するには、追加のソフトウェアが必要になります。ただし、Web サービスのアーキテクチャーはプラットフォームに依存するため、サービス・レジストリーが必要な場合は、別のプラットフォームに配置できます。

コンポーネント間の対話には、以下の操作が必要です。

Bind サービス・リクエスターはサービス記述を使用してサービス・プロバイダーをバインドし、Web サービスのインプリメンテーションと対話します。

Publish
サービス・レジストリーを使用した場合、サービス・プロバイダーは、リクエスターがサービス・プロバイダーの記述を見つけられるように、その記述をレジストリー内に発行します。

Find サービス・レジストリーを使用した場合、サービス・リクエスターはレジストリー内にあるサービス記述を見つけます。

Web サービス記述

Web サービス記述とは、サービス・プロバイダーがサービス・リクエスターに対して Web サービスを呼び出すための仕様を伝達する基準となる文書です。 Web サービス記述は、Web サービス記述言語 (WSDL) と呼ばれる XML アプリケーションで表現されます。

Web サービス記述では、サービス・プロバイダーとサービス・リクエスターとの通信を確保するために必要な共有知識やカスタマイズ・プログラミングの労力を最小限に抑えられるように Web サービスが記述されます。例えば、サービス・プロバイダーとサービス・リクエスターは、相手側で稼働しているプラットフォームを認識する必要はなく、相手側で作成されているプログラム言語を認識する必要もありません。

WSDL の構造により、サービス記述は次のように区分されます。

- 抽象的なサービス・インターフェース定義。サービスのインターフェースを記述し、サービスの実装と呼び出しを行うプログラムを作成可能にします。
- 具体的なサービス実装定義。プロバイダーの Web サービスのネットワーク (またはエンドポイント) の場所および実装に関するその他の具体的な詳細を記述し、サービス・リクエスターからサービス・プロバイダーへの接続を可能にします。

このことは、7 ページの図 2 に図示されています。

WSDL 文書には、ネットワーク・サービスの定義に以下の主要なエレメントが使用されます。

<types>
一定の型体系 (XML スキーマなど) を使用したデータ・タイプ定義のコンテナー。メッセージ内で使用されるデータ・タイプを定義します。すべてのメッセージが単純なデータ・タイプから成る場合は、<types> エレメントは必要ありません。

<message>
操作の入力パラメーターおよび出力パラメーターを定義するために使用する XML データ・タイプを指定します。

<portType>
1 つ以上のエンドポイントにサポートされている一連の操作を定義します。<portType> エレメントの内部では、各操作は <operation> エレメントで記述されます。

<operation>
入出力データ・フローで表示できる XML メッセージを指定します。操作は、プログラム言語のメソッド・シグニチャーに相当します。

<binding>
特定の <portType> エレメントに関するプロトコル、データ・フォーマット、セキュリティーおよびその他の属性を記述します。

<port> エンドポイントのネットワーク・アドレスを指定し、これを <binding> エレメントに関連付けます。

<service>
Web サービスを一まとまりの関連エンドポイントとして定義します。<service> エレメントには、1 つ以上の <port> エレメントが格納されています。

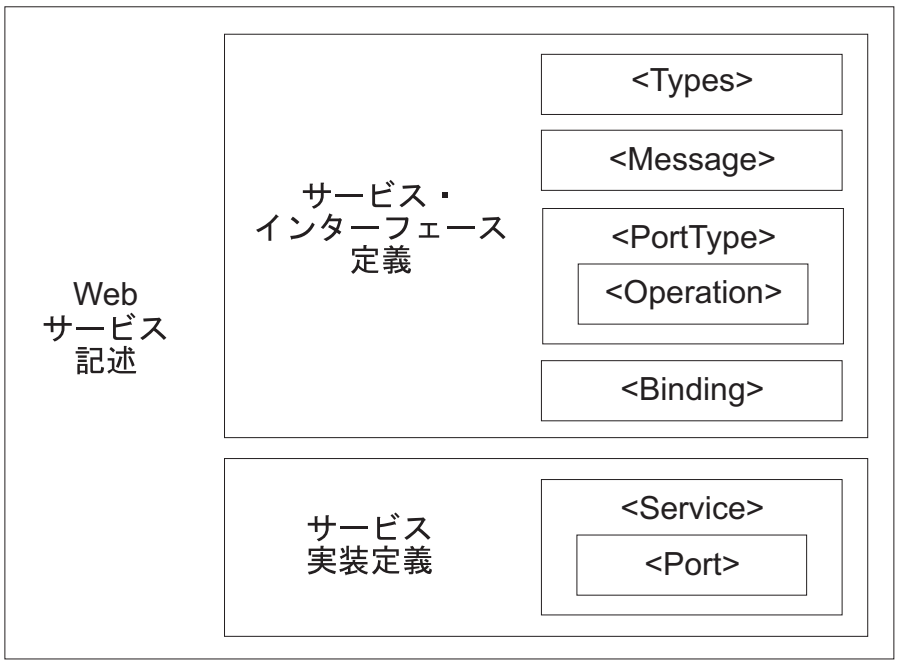


図 2. Web サービス記述の構造

Web サービス記述を区分する機能により、完全なサービス記述を作成する責務を分割できます。図のように、業界全体にわたって使用するため標準制定機関によって定義され、その業界内の個々の企業によって実装されるサービスについて考えます。

- 標準制定機関は、以下のエレメントを含むサービス・インターフェース定義を規定します。

  ```
  <types>
  <message>
  <portType>
  <binding
  ```

- サービスの実装を提案するサービス・プロバイダーは、以下のエレメントを含むサービス実装定義を規定します。

  ```
  <port>
  <service>
  ```

サービスの発行

サービスの記述は、多数の異なる機構を使用して発行できます。それぞれの機構には異なる機能があり、さまざまな状況での使用に適しています。 必要な場合、サービスの記述は複数の方法で発行できます。CICS はサービスの発行を直接サポートしていませんが、記述されている任意の機構を CICS と組み合わせて使用できます。

直接の発行

これは、サービス記述を発行するための最も単純な機構です。サービス・プロバイダーは、サービス記述をサービス・リクエスターに直接送信します。この実現方法には、E メールの添付ファイルの使用、FTP サイト、または CD ROM の配布などがあります。

Advertisement and Discovery of Services (ADS)
DISCO

これらの専有プロトコル群は、動的な発行機能を提供します。サービス・リクエスターは、サービス・プロバイダーによって指定され、URL で識別されるネットワークの場所から Web サービス記述を検索するために、単純な HTTP GET 機構を使用します。

Universal Description, Discovery and Integration (UDDI)

Web サービスに関する Web ベースの分散情報レジストリーの仕様。
UDDI は、企業が自社提供の Web サービスに関する情報を登録できる仕様の一連の実装形態でもあり、これによって他の企業はこの企業の情報を検索できます。

第 3 章 SOAP とは

SOAP とは、分散環境で情報を交換するためのプロトコルのことです。 SOAP メッセージは XML 文書としてエンコードされ、さまざまな下位プロトコルを使用して交換できます。

以前は Simple Object Access Protocol の頭字語であった SOAP は、非集中の分散環境で情報を交換するための単純なプロトコルです。 SOAP は World Wide Web Consortium (W3C) で開発され、W3C が発行した以下の文書で正式に定義されています。 SOAP に関して信頼できる詳細情報が必要な場合は、以下の資料を参照してください。

Simple Object Access Protocol (SOAP) 1.1 (W3C のメモ)

SOAP Version 1.2 Part 0: Primer (W3C 勧告)

SOAP Version 1.2 Part 1: Messaging Framework (W3C 勧告)

SOAP Version 1.2 Part 2: Adjuncts (W3C 勧告)

SOAP 仕様は、SOAP メッセージが SOAP ノード間に渡される分散処理モデルを記述しています。 SOAP メッセージは、SOAP 送信側から発信され、SOAP 受信側に送信されます。送信側と受信側の間では、メッセージは 1 つ以上の SOAP 中間ノードによって処理される場合があります。

SOAP メッセージは、SOAP ノード間、つまり SOAP 送信側から SOAP 受信側への片方向伝送ですが、メッセージを組み合わせると、要求と応答、対等の会話などのより複雑な対話を構成できます。

仕様には、以下の内容も記述されています。

- アプリケーション定義のデータ・タイプのインスタンスを表現するための一連のエンコード規則。
- リモート・プロシージャー・コールおよび応答を表現するための規則。

SOAP メッセージは、さまざまな下位プロトコルを使用して交換できます。

SOAP メッセージの構造

SOAP メッセージは、<Envelope> エレメントから構成される XML 文書としてエンコードされます。この <Envelope> エレメントには、オプションの <Header> エレメントと必須の <Body> エレメントが格納されています。 <Body> エレメントの内部に格納されている <Fault> エレメントは、エラー報告の用途で使用されます。

SOAP エンベロープ

SOAP <Envelope> は、各 SOAP メッセージのルート・エレメントであり、ここには、オプションの <Header> と必須の <Body> という 2 つの子エレメントが格納されています。

SOAP ヘッダー

SOAP <Header> は、SOAP エンベロープのオプションのサブエレメントであり、SOAP ノードがメッセージ・パスに沿って処理する対象のアプリケーション関連情報を渡すときに使用されます。

SOAP 本体

SOAP <Body> は、SOAP エンベロープの必須のサブエレメントで、ここにはメッセージの最終の受信側を目的とした情報が格納されています。

SOAP 障害

SOAP <Fault> は、SOAP 本体のサブエレメントで、エラー報告のために使用されます。

SOAP メッセージの <Body> に格納されている <Fault> エレメントを除くと、<Header> および <Body> 内部の XML エレメントは、これらのエレメントを使用するアプリケーションによって定義されます。ただし、SOAP 仕様のために、エレメントの構造には何らかの制約が課されます。

図 3 には、SOAP メッセージの主要なエレメントを示します。 11 ページの図 4 は、SOAP メッセージの例です。

図 3. SOAP メッセージの構造

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <n:name>Åke Jógvan Øyvind</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itinerary
    xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure>
     <p:departing>New York</p:departing>
     <p:arriving>Los Angeles</p:arriving>
     <p:departureDate>2001-12-14</p:departureDate>
     <p:departureTime>late afternoon</p:departureTime>
     <p:seatPreference>aisle</p:seatPreference>
   </p:departure>
   <p:return>
     <p:departing>Los Angeles</p:departing>
     <p:arriving>New York</p:arriving>
     <p:departureDate>2001-12-20</p:departureDate>
     <p:departureTime>mid-morning</p:departureTime>
     <p:seatPreference/>
   </p:return>
  </p:itinerary>
  <q:lodging
   xmlns:q="http://travelcompany.example.org/reservation/hotels">
   <q:preference>none</q:preference>
  </q:lodging>
 </env:Body>
</env:Envelope>
```

図 4. SOAP 1.2 メッセージの例

SOAP ヘッダー

SOAP <Header> は、SOAP メッセージ内部にあるオプションのエレメントです。このエレメントは、SOAP ノードがメッセージ・パスに沿って処理する対象のアプリケーション関連情報を渡すときに使用されます。

<Header> エレメントの直接の子エレメントは、ヘッダー・ブロックと呼ばれます。ヘッダー・ブロックは、アプリケーション定義の XML エレメントで、送信側から最後の受信側までのメッセージのパスで検出される可能性がある SOAP ノードを宛先にできるデータの論理グループを表します。

SOAP ヘッダー・ブロックは、SOAP 中間ノードと最終の SOAP 受信側ノードで処理できますが、実際のアプリケーションでは、すべてのヘッダー・ブロックが各ノードで処理されるわけではありません。むしろ、個々のノードは、通常、特定のヘッダー・ブロックを処理するよう設計されています。逆に言えば、個々のヘッダー・ブロックが特定のノードによって処理されるようになっています。

SOAP ヘッダーを使用すると、通信相手との間で事前に合意を得る必要なく、機能を非集中方式で SOAP メッセージに追加できます。 SOAP では、誰が機能に対応するか、およびその機能はオプションか必須かを示すために使用できる属性がいくつか定義されます。こうした「管理」情報には、メッセージの処理に関連した指示またはコンテキスト情報の受け渡しなどがあります。これにより、SOAP メッセージをアプリケーション固有の方式で拡張できます。

ヘッダー・ブロックはアプリケーション定義ですが、ヘッダー・ブロックに存在する SOAP 定義の属性は、SOAP ノードによるヘッダー・ブロックの処理方法を示しています。いくつかの重要な属性を以下に示します。

encodingStyle
SOAP メッセージの一部を直列化するときに使用するエンコード方式の規則を示します。

role (SOAP 1.2)
actor (SOAP 1.1)

SOAP 1.2 では、あるメッセージに対して特定のノードが稼働するかどうかが **role** 属性によって指定されます。特定のノードに指定された役割が、ヘッダー・ブロックの role 属性に一致した場合、このノードはヘッダーを処理します。役割が一致しない場合は、ヘッダー・ブロックを処理しません。 SOAP 1.1 では、**actor** 属性が同じ機能を実行します。

役割はアプリケーションによって定義され、URI で指定されます。例えば、`http://example.com/Log` は、ロギングを実行するノードの役割を指定しています。このノードに処理されるヘッダー・ブロックは、`env:role="http://example.com/Log"` を指定します (ここで、ネーム・スペースの接頭部 `env` は、`http://www.w3.org/2003/05/soap-envelope` の SOAP ネーム・スペース名に関連付けられます)。

SOAP 1.2 仕様では、アプリケーションによって定義されている役割のほかに、以下の 3 つの標準的な役割が定義されています。

`http://www.w3.org/2003/05/soap-envelope/none`
メッセージ・パス上の SOAP ノードがヘッダー・ブロックを直接処理することはありません。この役割を持つヘッダー・ブロックを使用すると、その他の SOAP ヘッダー・ブロックの処理に必要なデータを搬送できます。

`http://www.w3.org/2003/05/soap-envelope/next`
メッセージ・パス上にあるすべての SOAP ノードは、ヘッダー・ブロックを検査すると見込まれています (メッセージ・パスの前の方にあるノードによってヘッダーが削除されていないことが前提です)。

`http://www.w3.org/2003/05/soap-envelope/ultimateReceiver`
最終の受信側ノードのみがヘッダー・ブロックを検査すると見込まれています。

mustUnderstand
この属性は、SOAP ノードがアプリケーション全体の目的に重要なヘッダー・ブロックを無視しないようにするために使用します。 SOAP ノードが (**role** 属性または **actor** 属性を使用して)ヘッダー・ブロックを処理することを確認し、かつ **mustUnderstand**属性の値が `"true"` である場合、この SOAP ノードはその仕様に整合する方法でヘッダー・ブロックを処理するか、またはまったく処理

しない (で障害を通知する) 必要があります。ただし、属性の値が "false" である場合は、このノードがヘッダー・ブロックを処理する必要はありません。

mustUnderstand 属性は、実際にはヘッダー・ブロックの処理が必須かオプションかを示しています。

mustUnderstand 属性の値は次のとおりです。

true (SOAP 1.2)
1 (SOAP 1.1)
ノードは、仕様に整合する方法でヘッダー・ブロックを処理するか、またはまったく処理しない (で障害を通知する) 必要があります。

false (SOAP 1.2)
0 (SOAP 1.1)
ノードがヘッダー・ブロックを処理する必要はありません。

relay (SOAP 1.2 のみ)
SOAP 中間ノードは、ヘッダー・ブロックを処理すると、SOAP メッセージからヘッダー・ブロックを削除します。デフォルトでは、無視したすべてのヘッダー・ブロックを削除します (これは、**mustUnderstand** 属性の値が "false" であったためです)。ただし、**relay** 属性が "true" という値で指定されている場合、ノードはメッセージ内のヘッダー・ブロックを未処理のまま保存します。

SOAP 本体

<Body> は、SOAP メッセージで伝達される主な終端間情報の搬送媒体となる SOAP エンベロープ内部にある必須エレメントです。

<Body> エレメントとその関連の子エレメントは、最初の SOAP 送信側と最後の SOAP 受信側との間で情報を交換するために使用されます。SOAP では、<Body> に対して 1 つの子エレメント <Fault> を定義します。このエレメントは、エラーを報告するために使用されます。<Body> 内部のその他のエレメントは、それらを使用する Web サービスによって定義されます。

SOAP 本体の例

SOAP 障害

SOAP <Fault> エレメントは、SOAP メッセージ内部のエラー情報および状況情報を伝達するときに使用されます。

SOAP <Fault> エレメントは、存在する場合、本文の項目として存在する必要があり、Body エレメント内部に複数存在することはできません。SOAP <Fault> エレメントのサブエレメントは、SOAP 1.1 と SOAP 1.2 とで異なります。

SOAP 1.1

SOAP 1.1 では、SOAP <Fault> エレメントに以下のサブエレメントが格納されています。

<faultcode>
<faultcode> エレメントは、<Fault> エレメント内部の必須エレメントの 1 つです。このエレメントは、ソフトウェアが処理できる形式で障害に関する

情報を提供します。 SOAP は、基本的な SOAP 障害を網羅する SOAP 障害コードの小セットを定義します。このセットはアプリケーションによって拡張できます。

<faultstring>
<faultstring エレメントは、<Fault> エレメント内部の必須エレメントの 1 つです。このエレメントは、人間の読み手を対象とする形式で障害に関する情報を提供します。

<faultactor>
<faultactor> エレメントには、障害の発生原因となった SOAP ノードの URI が格納されています。 最後の SOAP 受信側以外の SOAP ノードは、障害コードの生成時に <faultactor> エレメントを包含する必要があります。 最後の SOAP 受信側はこのエレメントを包含することが必須ではありませんが、包含する場合もあります。

<detail>
<detail> エレメントは、<Body> エレメントに関連したアプリケーション固有のエラー情報を伝達します。このエレメントが存在する必要があるのは、<Body> エレメントの内容を正常に処理できなかった場合です。ヘッダー項目に属するエラー情報の情報を伝達するためにこのエレメントを使用することはできません。ヘッダー項目に属する詳細なエラー情報は、ヘッダー項目の内部に格納して搬送する必要があります。

SOAP 1.2

SOAP 1.2 では、SOAP <Fault> エレメントに以下のサブエレメントが格納されています。

<Code> <Code> エレメントは、<Fault> エレメント内部の必須エレメントの 1 つです。このエレメントは、ソフトウェアが処理できる形式で障害に関する情報を提供します。このエレメントには、<Value> エレメントとオプションの <Subcode> エレメントが格納されています。

<Reason>
<Reason エレメントは、<Fault> エレメント内部の必須エレメントの 1 つです。このエレメントは、人間の読み手を対象とする形式で障害に関する情報を提供します。 <Reason> エレメントには、1 つ以上の <Text> エレメントが格納されています。このエレメントのそれぞれには、障害に関する情報がさまざまな言語で記述されています。

<Node> <Node エレメントには、障害の発生原因となった SOAP ノードの URI が格納されています。 最後の SOAP 受信側以外の SOAP ノードは、障害コードの生成時に <Node エレメントを包含する必要があります。 最後の SOAP 受信側はこのエレメントを包含することが必須ではありませんが、包含する場合もあります。

<Role> <Role> エレメントには、障害が発生した箇所でノードが稼働していた役割を識別する URI が格納されています。

<Detail>
<Detail> エレメントは、オプションのエレメントで、障害を記述する SOAP 障害コードに関連したアプリケーション固有のエラー情報が格納され

ています。 <Detail> エレメントの存在は、障害のある SOAP メッセージのどの部分が処理されたかに関しては意味がありません。

SOAP ノード

SOAP ノードとは、SOAP メッセージ上で動作する処理ロジックのことです。

SOAP ノードが可能な処理は次のとおりです。

- SOAP メッセージの送信
- SOAP メッセージの受信
- SOAP メッセージの処理
- SOAP メッセージの中継

SOAP ノードが可能な役割は次のとおりです。

SOAP 送信側
SOAP メッセージを送信する SOAP ノード。

SOAP 受信側
SOAP メッセージを受信する SOAP ノード。

最初の SOAP 送信側
SOAP メッセージ・パスの開始点で SOAP メッセージを発信する SOAP 送信側

SOAP 中間ノード
SOAP 中間ノードとは、SOAP 受信側と SOAP 送信側の両方の機能を備え、SOAP メッセージの内部から宛先を指定できる SOAP ノードのことです。SOAP 中間ノードは、自身を宛先とする SOAP ヘッダー・ブロックを処理し、最終の SOAP 受信側に向けて SOAP メッセージを転送する役割を果たします。

最終の SOAP 受信側
SOAP メッセージの最終の宛先となる SOAP 受信側のことです。SOAP 本体およびこれを宛先とするすべての SOAP ヘッダー・ブロックの内容を処理する役割を果たします。一部の環境では、SOAP 中間ノードでの問題などの理由により、SOAP メッセージが最終の SOAP 受信側に到達できない場合があります。

SOAP メッセージ・パス

SOAP メッセージ・パスとは、1 つの SOAP メッセージが通過する一連の SOAP ノードのことです。これには、最初の SOAP 送信側、SOAP 中間ノード (存在しないか 1 つ以上)、最終の SOAP 受信側が含まれます。

最も簡単なケースでは、SOAP メッセージは 2 つのノード間で送信されます。つまり SOAP 送信側から SOAP 受信側までです。しかし、より複雑なケースでは、メッセージは SOAP 中間ノードによって処理されます。このノードでは、SOAP メッセージが受信され、さらに次のノードへ送信されます。 16 ページの図 5 に、SOAP メッセージ・パスの例を示します。ここでは、SOAP メッセージが最初の SOAP 送

信側ノードから最終の SOAP 受信側ノードに向けて送信され、その経路上で 2 つの SOAP 中間ノードを通過します。

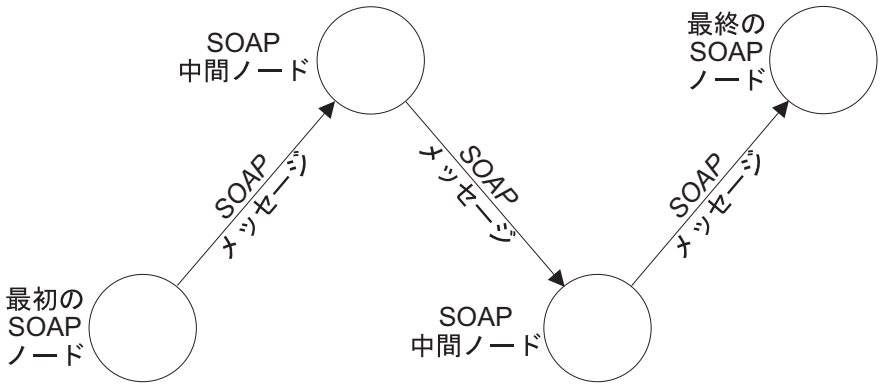


図 5. SOAP メッセージ・パスの例

SOAP 中間ノードは、SOAP 受信側と SOAP 送信側の両方の機能を備えています。SOAP メッセージのヘッダー・ブロックを処理でき、最終の受信側に向かって SOAP メッセージを転送します (ヘッダー・ブロックの処理は必須の場合もあります)。

最終の SOAP 受信側 とは、SOAP メッセージの最終的な宛先です。ヘッダー・ブロックの処理だけでなく、SOAP 本体を処理する役割も果たします。一部の環境では、SOAP 中間ノードでの問題などの理由により、SOAP メッセージが最終の SOAP 受信側に到達できない場合があります。

第 4 章 CICS が Web サービスをサポートする仕組み

CICS は、Web サービス環境で CICS アプリケーションを配置するための 2 つの異なる方法をサポートしています。一方の方法では、プログラミングの労力を最小限に抑えながら迅速な配置を実現できます。もう一方の方法では、各ユーザーの特定の必要性に合わせて記述したコードを使用することにより、Web サービス・アプリケーション全体にわたる柔軟性と制御を実現できます。 これら 2 つの方法は、1 つ以上のパイプラインと、Web サービスの要求および応答を対象として動作する 1 つ以上のメッセージ・ハンドラー・プログラムで構成されるインフラストラクチャーによって支えられています。

CICS アプリケーションを Web サービス環境で配置した場合に実行される処理は、以下のとおりです。

- CICS Web サービス・アシスタントを使用すると、アプリケーションを配置するために必要なプログラミングの労力を最小限に抑えることができます。

  例えば、既存のアプリケーションを Web サービスとして公開する場合は、高水準言語のデータ構造から着手して、Web サービス記述を生成することができます。 もう 1 つの方法として、既存の Web サービスと通信する場合は、Web サービス記述から着手し、作成するプログラムに使用可能な高水準言語の構造を生成することができます。

  CICS Web サービス・アシスタントは、アプリケーションを配置するために必要な CICS リソースも生成します。さらに、アプリケーションを実行すると、CICS は、出力ではアプリケーション・データを SOAP メッセージに変換し、入力では SOAP メッセージをアプリケーション・データに戻します。
- データの処理を完全に制御するには、独自のコードを記述して、サービス・リクエスターとサービス・プロバイダーとの間で、アプリケーション・データとメッセージ・フローとをマップします。

  例えば、Web サービス・インフラストラクチャーの範囲内で SOAP 以外のメッセージを使用する場合は、独自のコードを記述して、メッセージ・フォーマットとアプリケーションが使用するフォーマットとを変換します。

どちらの方法を採用する場合でも、独自のメッセージ・ハンドラーを使用して要求メッセージおよび応答メッセージに対する追加の処理を実行できます。または、SOAP メッセージの処理専用に設計された CICS 提供のメッセージ・ハンドラーを使用することもできます。

メッセージ・ハンドラーおよびパイプライン

メッセージ・ハンドラーとは、Web サービスの要求および応答について独自の処理を実行できるプログラムのことです。 パイプラインとは、順序どおり実行される一連のメッセージ・ハンドラーのことです。

パイプラインの運用には次に示す 2 つの異なる段階があります。

1. 要求段階。CICS がパイプライン内の各ハンドラーを次々と呼び出す段階です。各メッセージ・ハンドラーは、制御を CICS に戻す前に要求を処理できます。

2. この後に応答段階が続きます。この段階でも、CICS は各ハンドラーを次々と呼び出しますが、順序が逆になります。つまり、要求段階で最初に呼び出されるメッセージ・ハンドラーは、応答段階では最後に呼び出されます。各メッセージ・ハンドラーは、この段階のうちに応答を処理できます。

   要求の後に必ずしも応答があるわけではありません。つまり、一部のアプリケーションは、サービス・リクエスターからプロバイダーへの片方向のメッセージ・フローを使用します。この場合、処理すべきメッセージがなくても、応答段階で各ハンドラーが順に呼び出されます。

図 6 には、次の 3 つのメッセージ・ハンドラーのパイプラインを示します。

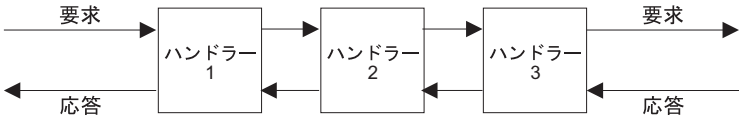


図 6. 一般的な CICS パイプライン

この例では、ハンドラーは次の順序で実行されます。

要求段階の場合

1. ハンドラー 1
2. ハンドラー 2
3. ハンドラー 3

応答段階の場合

1. ハンドラー 3
2. ハンドラー 2
3. ハンドラー 1

サービス・プロバイダーの場合、段階間の移行は、通常、要求を吸収するパイプラインの最後のハンドラー (端末ハンドラー) で実行され、応答が生成されます。サービス・リクエスターの場合、移行が実行されるのは、要求がサービス・プロバイダーで処理されるときです。ただし、要求段階のメッセージ・ハンドラーは、応答段階への即時の移行を強制できます。また、CICS によってエラーが検出された場合にも、即時の移行を実行することができます。

メッセージ・ハンドラーは、メッセージを変更することも、変更せずにそのままの状態にしておくこともできます。例を次に示します。

- 暗号化と復号を実行するメッセージ・ハンドラーは、暗号化されたメッセージを入力で受け取り、復号されたメッセージを次のハンドラーに渡します。出力では、逆の処理が行われます。つまり、非暗号化テキスト・メッセージを受け取り、暗号化されたメッセージを次のハンドラーに渡します。
- ロギングを実行するメッセージ・ハンドラーは、メッセージを調べ、関係のある情報をこのメッセージからログにコピーします。次のハンドラーに渡されるメッセージは変更されません。

重要: CICS TS の SOAP 機能に精通している場合は、このリリースの CICS でのパイプラインの構造が SOAP 機能に使用されているものと同じではないことに留意してください。

トランスポート関連ハンドラー

CICS は、Web サービス・リクエスターとプロバイダー間での 2 つのトランスポート機構の使用をサポートしています。使用中のトランスポート機構がどちらであるかによっては、異なるメッセージ・ハンドラーを呼び出すことが必要な場合があります。例えば、HTTP トランスポートを使用して外部ネットワークと通信する場合には、メッセージの一部を暗号化するメッセージ・ハンドラーが必要になります。しかし、機密保護機能のある内部ネットワークで MQ トランスポートを使用する場合、暗号化は必要ありません。

これをサポートするため、パイプラインを構成することにより、特定のトランスポート (HTTP または MQ) が使用中の場合にのみ呼び出されるハンドラーを指定できます。サービス・プロバイダーの場合は、さらに具体的に、特定の指定リソース (HTTP トランスポートの場合は TCPIPSERVICE、MQ トランスポートの場合は QUEUE) が使用中の場合にのみ呼び出されるハンドラーを指定できます。

このことは、図 7 に図示されています。

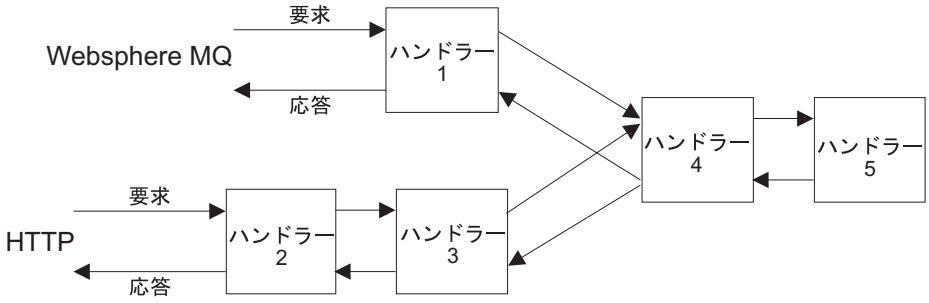


図 7. トランスポート関連ハンドラーを持つパイプライン

この例では、サービス・プロバイダーに適用されるものは次のとおりです。

- ハンドラー 1 は、MQ トランスポートを使用するメッセージの場合に呼び出されます。
- ハンドラー 2 および 3 は、HTTP トランスポートを使用するメッセージの場合に呼び出されます。
- ハンドラー 4 および 5 は、すべてのメッセージを対象として呼び出されます。
- ハンドラー 5 は、端末ノードです。

フローの中断

要求の処理時に、メッセージ・ハンドラーはメッセージを次のハンドラーに渡さない判断をする場合がありますが、応答を生成する場合もあります。メッセージの通常の処理は中断され、パイプライン内の一部のハンドラーは呼び出されません。例えば、 20 ページの図 8 のハンドラー 2 は、セキュリティー検査を実行する役割を持っています。

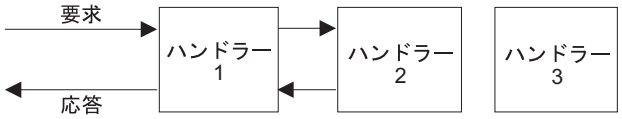


図 8. パイプライン・フローの中断

要求に正しいセキュリティー・クリデンシャルがない場合、ハンドラー 2 は、(要求をハンドラー 3 に渡さずに) 要求を抑止し、適切な応答を作成します。パイプラインは応答段階になっているため、ハンドラー 2 が制御を CICS に戻すと、呼び出される次のハンドラーはハンドラー 1 となり、ハンドラー 3 は完全にバイパスされます。

通常のメッセージ・フローをこのように中断するハンドラーは、メッセージの発信元が応答を期待する場合にのみ、中断する必要があります。例えば、アプリケーションがサービス・リクエスターからプロバイダーへの片方向のメッセージ・フローを使用する場合、ハンドラーは応答を生成してはいけません。

サービス・プロバイダー・パイプライン

サービス・プロバイダー・パイプラインでは、CICS が要求を受け取ります。この要求はパイプラインを介してターゲット・アプリケーション・プログラムに渡されます。 アプリケーションからの応答は、同じパイプラインを介してサービス・リクエスターに戻されます。

CICS がサービス・プロバイダーの役割を果たす場合、CICS は以下の操作を実行します。

1. サービス・リクエスターからの要求を受け取る。
2. 要求を調べ、ターゲット・アプリケーション・プログラムに関係のある内容を抽出する。
3. アプリケーション・プログラムを呼び出し、要求から抽出したデータを渡す。
4. アプリケーション・プログラムが制御を戻したら、アプリケーション・プログラムから戻されたデータを使用して応答を作成する。
5. サービス・リクエスターへ応答を送信する。

図 9 には、サービス・プロバイダー設定における次の 3 つのメッセージ・ハンドラーのパイプラインを示します。

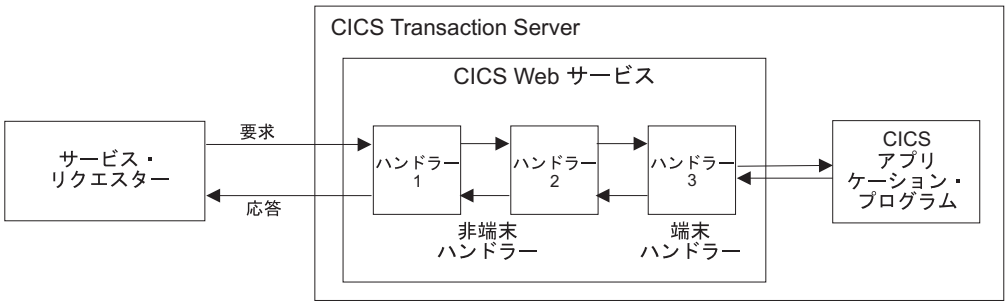


図 9. サービス・プロバイダー・パイプライン

1. CICS は、サービス・プロバイダーから要求を受け取ります。次に、この要求をメッセージ・ハンドラー 1 に渡します。
2. メッセージ・ハンドラー 1 は何らかの処理を実行して、要求をハンドラー 2 に渡します。(正確には、パイプラインを管理する CICS に制御を戻します。 次に CICS は次のメッセージ・ハンドラーに制御を渡します。)
3. メッセージ・ハンドラー 2 はハンドラー 1 から要求を受け取り、何らかの処理を実行して、要求をハンドラー 3 に渡します。
4. メッセージ・ハンドラー 3 は、このパイプラインの端末ハンドラーです。ハンドラー 3 は、要求に記載された情報を使用して、アプリケーション・プログラムを呼び出します。次に、アプリケーション・プログラムからの出力を使用して応答を生成し、この応答をハンドラー 2 に戻します。
5. メッセージ・ハンドラー 2 はハンドラー 3 から応答を受け取り、何らかの処理を実行して、応答をハンドラー 1 に渡します。
6. メッセージ・ハンドラー 1 はハンドラー 2 から応答を受け取り、何らかの処理を実行して、応答をサービス・リクエスターに戻します。

サービス・リクエスター・パイプライン

サービス・リクエスター・パイプラインでは、アプリケーション・プログラムが要求を作成します。この要求はパイプラインを介してサービス・プロバイダーに渡されます。 サービス・プロバイダーからの応答は、同じパイプラインを介してアプリケーション・プログラムに戻されます。

CICS がサービス・リクエスターの役割を果たす場合、CICS は以下の操作を実行します。

1. アプリケーション・プログラムから得られたデータを使用して、要求を作成する。
2. 要求をサービス・プロバイダーに送信する。
3. サービス・プロバイダーから応答を受信する。
4. 応答を調べ、オリジナル・アプリケーション・プログラムに関係のある内容を抽出する。
5. アプリケーション・プログラムに制御を戻す。

図 10 には、サービス・リクエスターの設定における次の 3 つのメッセージ・ハンドラーのパイプラインを示します。

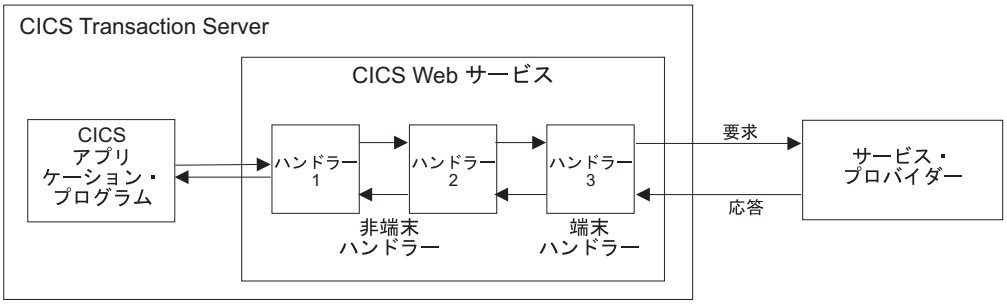


図 10. サービス・リクエスター・パイプライン

1. アプリケーション・プログラムが要求を作成します。
2. メッセージ・ハンドラー 1 は、アプリケーション・プログラムから要求を受け取り、何らかの処理を実行して、要求をハンドラー 2 に渡します。(正確には、パイプラインを管理する CICS に制御を戻します。 次に CICS は次のメッセージ・ハンドラーに制御を渡します。)
3. メッセージ・ハンドラー 2 はハンドラー 1 から要求を受け取り、何らかの処理を実行して、要求をハンドラー 3 に渡します。
4. メッセージ・ハンドラー 3 は、ハンドラー 2 から要求を受け取り、何らかの処理を実行して、要求をサービス・プロバイダーに渡します。
5. メッセージ・ハンドラー 3 はサービス・プロバイダーから応答を受け取り、何らかの処理を実行して、応答をハンドラー 2 に渡します。
6. メッセージ・ハンドラー 2 はハンドラー 3 から応答を受け取り、何らかの処理を実行して、応答をハンドラー 1 に渡します。
7. メッセージ・ハンドラー 1 はハンドラー 2 から応答を受け取り、何らかの処理を実行して、応答をアプリケーション・プログラムに戻します。

CICS パイプラインおよび SOAP

Web サービスの要求と応答を処理するために CICS が使用するパイプラインは、各メッセージ・ハンドラーで実行できる処理に関する制約が少ないという点で一般的です。ただし、多くの Web サービス・アプリケーションは SOAP メッセージを使用しており、これらのメッセージを処理する場合は、SOAP 仕様に準拠する必要があります。 したがって、CICS には、パイプラインを SOAP ノードとして構成するときに役立つ特殊な SOAP メッセージ・ハンドラー・プログラムが用意されています。

- パイプラインを構成すると、SOAP 1.1 または SOAP 1.2 をサポートできます。CICS システム内部には、多数のパイプラインを保持できます。このうちのいくつかは SOAP 1.1 または SOAP 1.2 をサポートします。
- パイプラインは、サービス・リクエスターまたはサービス・プロバイダーで使用するために、次のように構成できます。
  - サービス・リクエスター・パイプラインは、要求の最初の SOAP 送信側になり、かつ応答の最終の SOAP 受信側になります。
  - サービス・プロバイダー・パイプラインは、要求の最終の SOAP 受信側になり、かつ応答の最初の SOAP 送信側になります。

  CICS パイプラインを SOAP 中間ノードとして機能するように構成することはできません。
- CICS パイプラインを構成すると、複数の SOAP メッセージ・ハンドラーを保持できます。
- CICS 提供の SOAP メッセージ・ハンドラーを構成すると、1 つ以上のユーザー作成ヘッダー処理ルーチンを呼び出すことができます。
- CICS 提供の SOAP メッセージ・ハンドラーを構成すると、WS-I Basic Profile 1.0 に準拠するといういくつかの側面と、SOAP メッセージに特定のヘッダーを存在させることを実現できます。

SOAP メッセージ・ハンドラー、およびそのヘッダー処理ルーチンは、パイプライン構成ファイルで指定します。

SOAP メッセージおよびアプリケーション・データ構造

多くの場合、CICS Web サービス・アシスタントは、アプリケーション・プログラムで使用されている上位データ構造と、SOAP メッセージの <Body> エレメントの内容との間でデータを変換するコードを生成できます。 これらの場合には、アプリケーション・プログラムを作成するときに、SOAP本体の解析または構成を行う必要がありません。これらの作業は CICS によって実行されます。

CICS は実行時に、データを変換するためにアプリケーション・データ構造と SOAP メッセージの形式に関する情報を必要とします。この情報は、次の 2 つのファイルに保持されます。

- Web サービスのバインディング・ファイル

  このファイルは、ユーティリティー・プログラム DFHLS2WS を使用して、アプリケーション言語のデータ構造を基に CICS Web サービス・アシスタントによって生成されるか、またはユーティリティー・プログラム DFHWS2LS を使用して、Web サービス記述を基に生成されます。CICS はバインディング・ファイルを使用して、Web サービス・アプリケーションが使用するリソースを生成し、アプリケーションのデータ構造と SOAP メッセージ間のマッピングを行います。

- Web サービス記述

  これは、既存の Web サービス記述である場合と、ユーティリティー・プログラム DFHLS2WS を使用して、アプリケーション言語のデータ構造を基に生成する場合があります。CICS では、Web サービス記述を使用して、SOAPメッセージの完全な検証を行います。

図 11 に、これらのファイルがサービス・プロバイダーで使用される様子を示します。

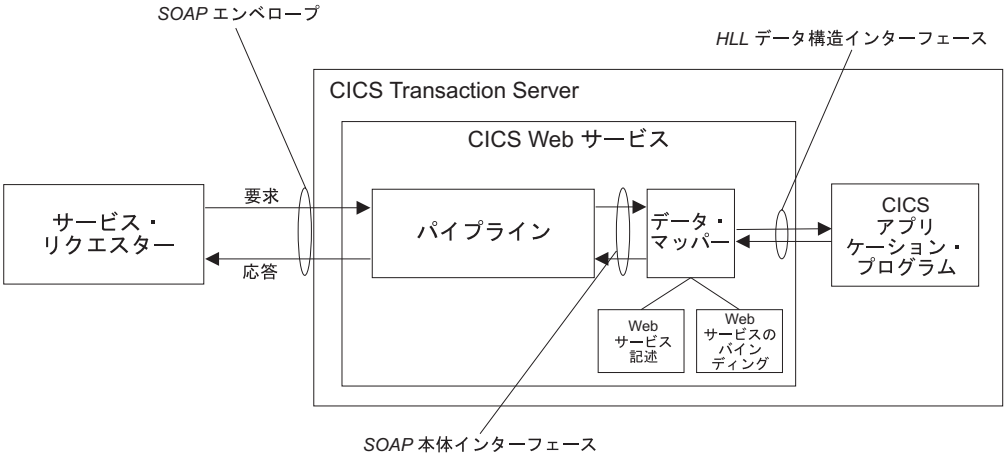


図 11. サービス・プロバイダーでの SOAP 本体からアプリケーション・データ構造へのマッピング

24 ページの図 12 には、これらのファイルがサービス・リクエスターで使用される様子を示します。

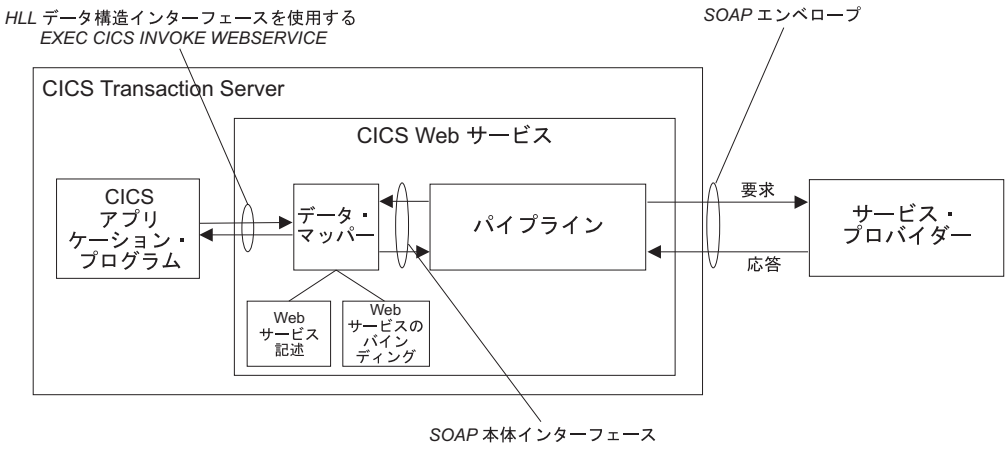


図 12. サービス・リクエスターでの SOAP 本体からアプリケーション・データ構造へのマッピング

どちらの場合も、特定の CICS アプリケーション・プログラムが Web サービスの設定で動作できる実行環境は、3 つのオブジェクトによって定義されます。これらは、パイプライン、Web サービス・バインディング・ファイル、および Web サービス記述です。これら 3 つのオブジェクトは、WEBSERVICE リソース定義の属性として CICS に定義されています。

SOAP メッセージを使用している場合でも、次に示すように、CICS Web サービス・アシスタントが生成する変換を使用できない状況がいくつか存在します。

- SOAP メッセージと高水準言語で同じデータが表現できない場合

  CICS がサポートしているすべての高水準言語、および XML スキーマでは、さまざまなデータ・タイプがサポートされています。しかし、高水準言語で使用されるデータ・タイプと XMLスキーマで使用されるデータ・タイプとの間に 1 対 1 対応は存在しないため、データを一方で表現できても他方では表現できないという場合が存在します。こうした状況では、次のいずれかの手段を検討する必要があります。

  – アプリケーション・データ構造を変更します。この方法は、必然的にアプリケーション・プログラム自体の変更が必要になるため、実現は困難です。
  – ラッパー・プログラムを作成します。このプログラムは、アプリケーション・データを CICS が処理可能な形式に変換し、さらに SOAP メッセージ本文に変換します。この方法を実行した場合は、アプリケーション・プログラムを変更せずに済みます。この場合は、CICS Web サービス・サポートがラッパー・プログラムと直接対話し、アプリケーション・プログラムとは間接的にのみ対話します。

- アプリケーション・プログラムが、CICS Web サービス・アシスタントでサポートされない言語で記述されている場合

  こうした状況では、次のいずれかの手段を検討する必要があります。

  – CICS Web サービス・アシスタントがサポートするいずれかの言語 (COBOL、PL/I、C または C++) で記述されたラッパー・プログラムを作成します。

– CICS Web サービス・アシスタントを使用する代わりに、SOAP メッセージとアプリケーション・プログラムのデータ構造間のマッピングを行うプログラムを独自に作成します。

WSDL およびアプリケーション・データ構造

Web サービス記述には、Web サービスが使用する入出力メッセージの抽象表現が含まれています。 CICS では、Web サービス記述を使用して、アプリケーション・プログラムが使用するデータ構造を構成します。 CICS は、実行時に、アプリケーション・データ構造とメッセージとのマッピングを実行します。

Web サービス記述の代表例は、以下のとおりです。

- 1 つ以上の操作
- 各操作ごとに、入力メッセージ、およびオプションの出力メッセージ。
- メッセージごとに、XML データ・タイプの観点で定義されたメッセージ構造。メッセージ内で使用される複合データ・タイプは、Web サービス記述内にある `<types>` エレメントに記述されている XML スキーマで定義されます。簡単なメッセージは、`<types>` エレメントを使用しないで記述できます。

WSDL には、操作の抽象定義と関連メッセージが記述されています。WSDL をアプリケーション・プログラム内に直接使用することはできません。操作を実装するには、サービス・プロバイダーが以下の処理を実行する必要があります。

- メッセージの構造を把握するために WSDL の構文解析を行う。
- 入力メッセージを解析して出力メッセージを作成する。
- 入出力メッセージの内容と、アプリケーション・プログラムで使用されているデータ構造とのマッピングを実行する。

サービス・リクエスターは、操作を呼び出すために同じことを行う必要があります。

CICS Web サービス・アシスタントを使用すると、前述の大半の処理がユーザーの代わりに実行されるため、ユーザーは入出力メッセージを構成する方法や WSDL を詳細に理解する必要なくアプリケーション・プログラムを作成できます。

CICS Web サービス・アシスタントは、以下の 2 つのユーティリティー・プログラムで構成されています。

DFHWS2LS
このユーティリティー・プログラムは、Web サービス記述を開始点にしています。このプログラムでは、アプリケーション・プログラムに使用できる高水準言語データ構造を構成するために、メッセージの記述や、メッセージに使用されているデータ・タイプを使用します。

DFHLS2WS
このユーティリティー・プログラムは、高水準言語データ構造を開始点にしています。このプログラムでは、メッセージの記述を格納する Web サービス記述を構成するための構造体と、言語構造から導出されたこれらのメッセージで使用されるデータ・タイプが使用されます。

いずれのユーティリティー・プログラムも、アプリケーション・プログラムのデータ構造と SOAP メッセージ間のマッピングを実行するために CICS が実行時に使用する Web サービス・バインディング・ファイルを生成します。

COBOL から WSDL へのマッピングの例

この例では、COBOL プログラムで使用されているデータ構造が、CICS Web サービス・アシスタントによって生成された Web サービス記述内でどのように表現されているかを示します。

図 13 は、単純な COBOL データ構造を示しています。

```
*    カタログ COMMAREA 構造
         03 CA-REQUEST-ID            PIC X(6).
         03 CA-RETURN-CODE           PIC 9(2).
         03 CA-RESPONSE-MESSAGE      PIC X(79).
     *    Place Order (発注) で使用されているフィールド
         03 CA-ORDER-REQUEST.
             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).
```

図 13. WSDL で定義されている入力メッセージの COBOL レコード定義

Web サービス記述の対応するフラグメントでの重要なエレメントを、27 ページの図 14 に示します。

```
<xsd:sequence>
    <xsd:element name="CA-REQUEST-ID" nillable="false">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:length value="6"/>
                <xsd:whiteSpace value="preserve"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="CA-RETURN-CODE" nillable="false">
        <xsd:simpleType>
            <xsd:restriction base="xsd:short">
                <xsd:maxInclusive value="99"/>
                <xsd:minInclusive value="0"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:element>
    <xsd:element name="CA-RESPONSE-MESSAGE" nillable="false">
        ...
    </xsd:element>
    <xsd:element name="CA-ORDER-REQUEST" nillable="false">
        <xsd:complexType mixed="false">
            <xsd:sequence>
                <xsd:element name="CA-USERID" nillable="false">
                    <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                    <xsd:length value="8"/>
                    <xsd:whiteSpace value="preserve"/>
                    </xsd:restriction>
                    </xsd:simpleType>
                </xsd:element>
                <xsd:element name="CA-CHARGE-DEPT" nillable="false">
                    ...
                </xsd:element>
                <xsd:element name="CA-ITEM-REF-NUMBER" nillable="false">
                    ...
                </xsd:element>
                <xsd:element name="CA-QUANTITY-REQ" nillable="false">
                    ...
                </xsd:element>
                <xsd:element name="FILLER" nillable="false">
                    ...
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:sequence>
```

図 14. COBOL データ構造から導出された WSDL フラグメント

Web サービスのバインディング・ファイル

Web サービスのバインディング・ファイルには、入出力メッセージとアプリケーション・データ構造との間でデータをマップするために CICS が使用する情報が格納されています。

Web サービス記述には、Web サービスが使用する入出力メッセージの抽象表現が含まれています。サービス・プロバイダーまたはサービス・リクエスターのアプリケーションを実行する場合、CICS に必要な情報は、メッセージの内容をアプリケーションが使用するデータ構造へマップする方法になります。この情報は、Web サービスのバインディング・ファイルに保持されます。

Web サービスのバインディング・ファイルの作成方法は、次のとおりです。

- 言語構造を WSDL を基に生成する場合は、ユーティリティー・プログラム DFHWS2LS
- WSDL を言語構造を基に生成する場合は、ユーティリティー・プログラム DFHLS2WS

実行時に、CICS は Web サービスのバインディング・ファイルを使用してアプリケーション・データ構造と SOAP メッセージとのマッピングを行います。 Web サービスのバインディング・ファイルは、WEBSERVICE リソースの WSBIND 属性で、CICS に対して定義されます。

第 5 章 CICS での Web サービス入門

CICS で Web サービスを始めるには、いくつかの方法があります。 どの方法が最適かは、対象の題材に関する習得済みの知識の量や、Web サービスを使用する計画の進捗度により異なります。

CICS での Web サービスに関するいくつかの開始点を以下に示します。

- アプリケーションの例をインストールします。 CICS には、Web サービス・プロバイダーとして使用できるカタログ管理アプリケーションの例が用意されています。この例には、CICS で動作するアプリケーションを最小限の作業量で取得するために必要なすべてのコードおよびリソース定義が格納されています。ここには、多くの共通 Web サービス・クライアント上で実行されるサービスと対話するためのコードも収録されています。

  CICS で Web サービスを配置できる迅速な『概念実証』デモンストレーションが必要な場合や、CICS での Web サービスを習得するための『実践』方式が必要な場合は、この実例アプリケーションを使用してください。
- サービス・プロバイダーまたはサービス・リクエスターとしてアプリケーションを配置する作業にすぐに取り掛かります。 CICS で Web サービスを使用することによってアプリケーションおよび関連インフラストラクチャーの開発を開始する方法については、すでに十分な知識があるという前提です。
- CICS の SOAP 機能からマイグレーションします。

Web サービスに応じた CICS システムの構成

Web サービスを使用するには、CICS システムを正しく構成する必要があります。

PL/I の言語環境サポートをインストール済みであることを確認してください。 このサポートが必要なのは、CICS 提供の SOAP 1.1 および SOAP 1.2 メッセージ・ハンドラー・プログラムを使用する場合です。

詳細は、「*CICS インストール・ガイド*」を参照してください。

Web サービスのための CICS リソース

CICS で Web サービスをサポートする CICS リソースを以下に示します。

PIPELINE

PIPELINE リソース定義は、あらゆる場合に必要です。これは、サービス要求および応答に対して作用するメッセージ・ハンドラー・プログラムに関する情報を提供します。一般に、単一の PIPELINE 定義で定義されたインフラストラクチャーを、多数のアプリケーションで使用できます。メッセージ・ハンドラーに関する情報は、間接的に指定されます。PIPELINE はノードとその構成の XML 記述を格納する HFS ファイルの名前を指定します。

サービス・リクエスター用に作成された PIPELINE リソースは、サービス・プロバイダーでは使用できず、サービス・プロバイダー用に作成された PIPELINE リソースもサービス・リクエスターでは使用できません。2 種類の PIPELINE は、CONFIGFILE 属性に指定されているパイプライン構成フ

ファイルの内容によって区別されます。サービス・プロバイダーでは、最上位のエレメントが <provider_pipeline> で、サービス・リクエスターでは <requester_pipeline> です。

WEBSERVICE

WEBSERVICE リソース定義は、アプリケーション・データ構造と SOAP メッセージの間のマッピングが CICS Web サービス・アシスタントを使用して生成されている場合にのみ必要です。このリソース定義では、配置される CICS アプリケーション・プログラムの実行時環境の性質を Web サービスの設定で定義します。

CICS は WEBSERVICE リソースの通常のリソース定義メカニズムを備えていますが、一般にこのリソースは PIPELINE のピックアップ・ディレクトリーがスキャンされると、Web サービス・バインディング・ファイルから自動的に作成されます。これは、PIPELINE リソースがインストールされたとき、あるいは PERFORM PIPELINE SCAN コマンドの結果として行われます。この場合に WEBSERVICE リソースに適用される属性は、 Web サービス・アシスタントによって作成された Web サービス・バインディング・ファイルから取得されます。バインディング・ファイル内の情報は、Web サービス記述から取得されるか、または Web サービス・アシスタントのパラメーターとして提供されます。

サービス・リクエスター用に作成された WEBSERVICE リソースは、サービス・プロバイダーでは使用できず、サービス・プロバイダー用に作成された WEBSERVICE リソースもサービス・リクエスターでは使用できません。2 種類の WEBSERVICE は、PROGRAM 属性によって区別されます。サービス・プロバイダーでは、属性の指定が必要です。サービス・リクエスターでは属性を省略する必要があります。

URIMAP

URIMAP 定義は、サービス・プロバイダーのみで必要です。この定義には、インバウンド Web サービス要求の URI を、その要求を処理する他のリソース (PIPELINE など) とマップする情報が含まれます。

CICS は通常のリソース定義メカニズムを備えていますが、CICS Web サービス・アシスタントを使用して配置されたサービス・プロバイダーでは、URIMAP リソースは通常の場合、PIPELINE のピックアップ・ディレクトリーがスキャンされると、Web サービス・バインディング・ファイルから自動的に作成されます。これは、PIPELINE リソースがインストールされたとき、あるいは PERFORM PIPELINE SCAN コマンドの結果として行われます。この場合に URIMAP リソースに適用される属性は、 Web サービス・アシスタントによって作成された Web サービス・バインディング・ファイルから取得されます。バインディング・ファイル内の情報は、Web サービス記述から取得されるか、または Web サービス・アシスタントのパラメーターとして提供されます。

TCPIPSERVICE

TCPIPSERVICE 定義は、HTTP トランスポートを使用するサービス・プロバイダーで必要です。この定義には、インバウンド要求を受信するポートに関する情報が含まれます。

特定のアプリケーション・プログラムをサポートするために必要なリソースは、以下の条件によって異なります。

- アプリケーション・プログラムがサービス・プロバイダーであるか、サービス・リクエスターであるか
- アプリケーションが CICS Web サービス・アシスタントを使用して配置されるかどうか

サービス・リクエスターまたはサービス・プロバイダー	CICS Web サービス・アシスタントの使用	PIPELINE が必要であるか	WEBSERVICE が必要であるか	URIMAP が必要であるか	TCPIPSERVICE が必要であるか
プロバイダー	はい	はい	はい (ただし、1 を参照)	はい (ただし、1 を参照)	2 を参照
	いいえ	はい	いいえ	はい	2 を参照
リクエスター	はい	はい	はい	いいえ	いいえ
	いいえ	はい	いいえ	いいえ	いいえ

注:

1. CICS Web サービス・アシスタントを使用してアプリケーション・プログラムを配置する場合、PIPELINE のピックアップ・ディレクトリーのスキャン時に WEBSERVICE リソースおよび URIMAP リソースを自動的に作成することができます。これは、PIPELINE リソースがインストールされたとき、あるいは PERFORM PIPELINE SCAN コマンドの結果として行われます。
2. TCPIPSERVICE リソースは、HTTP トランスポートを使用するときに必要です。WebSphere MQ トランスポートの使用時は、定義が必要です。

一般に、CICS システムに多数の Web サービス・アプリケーションを配置する場合、それぞれのタイプのリソースを複数作成することになります。この場合、アプリケーション間でいくつかのリソースを共用できます。

表 1.

1 つのファイルまたはリソースに対して	作成できるリソース
パイプライン構成ファイル	• このファイルを参照する複数の PIPELINE リソース
PIPELINE リソース	• PIPELINE を参照する複数の URIMAP リソース • PIPELINE を参照する複数の WEBSERVICE リソース • PIPELINE のピックアップ・ディレクトリー内の複数の Web サービス・バインディング・ファイル

表 1. (続き)

1 つのファイルまたはリソースに対して	作成できるリソース
Web サービス・バインディング・ファイル	• バインディング・ファイルから自動的に生成される 1 つの URIMAP リソースのみ。ただし、RDO を使用してさらに複数の URIMAP を定義することができます。 • バインディング・ファイルから自動的に生成される 1 つの WEBSERVICE リソースのみ。ただし、RDO を使用してさらに複数の WEBSERVICE を定義することができます。
WEBSERVICE	• 複数の URIMAP リソース。WEBSERVICE リソースがバインディング・ファイルから自動的に生成される場合は、これに対応する URIMAP リソースが 1 つだけあります。ただし、RDO を使用してさらに複数の URIMAP リソースを定義することができます。
URIMAP	• URIMAP リソースで明示的に指定される場合は、1 つの TCPIPSERVICE のみ
TCPIPSERVICE	• 多数の URIMAP リソース

MQ トランスポートを使用するための CICS の構成

CICS で MQ トランスポートを Web サービスと組み合わせて使用するには、それに応じて CICS 領域を構成する必要があります。

詳細は、「*MQ Series for OS/390 System Setup Guide*」に記載されています。

1. STEPLIB 連結に以下のライブラリーを指定します。

   thlqual.SCSQANL*x*

   thlqual.SCSQAUTH

   ここで

   thlqual は、MQ ライブラリーの高位修飾子です。

   x は、各国語の言語を表す文字です。

2. DFHRPL 連結に以下のライブラリーを指定します。

   thlqual.SCSQLOAD

   thlqual.SCSQANL*x*

   thlqual.SCSQCICS

   thlqual.SCSQAUTH

   thlqual は、MQ ライブラリーの高位修飾子です。

   x は、各国語の言語を表す文字です。

3. 以下の CICS システム初期設定パラメーターを指定します。

```
INITPARM=(CSQCPARM='SN=queuemanager,TN=traceptid,IQ=initiation_queue')
MQCONN=YES
```

ここで

queuemanager は、サブシステム名です。

traceptid は、CICSトレース・エントリーでアダプターを識別するトレース番号です。

initiation_queue は、デフォルトの開始キューの名前です。

サービス・プロバイダーでのローカル・キューの定義

サービス・プロバイダーで WebSphere MQ トランスポートを使用する場合は、入力メッセージを処理するまで入力メッセージを保管する 1 つ以上のローカル・キューと、入力メッセージを処理する CICS トランザクションを指定する 1 つのトリガー・プロセスを定義する必要があります。

1. ローカル入力キューごとに、QLOCAL オブジェクトを定義します。 以下のコマンドを使用します。

```
DEFINE
QLOCAL('queuename')
DESCR('description')
PROCESS(processname)
INITQ('initqueue')
TRIGGER
TRIGTYPE(FIRST)
TRIGDATA('default_target_service')
BOTHRESH(nnn)
BOQNAME('requeuename')
```

ここで

queuename は、ローカル・キュー名です。

processname は、トリガー・イベント発生時にキュー・マネージャーによって開始されるアプリケーションを示すプロセス・インスタンスの名前です。各 QLOCAL オブジェクトにも、同じ名前を指定します。

initqueue は、使用される開始キューの名前です (例: SIT の INITPARM 内の CSQCPARM に IQ= を指定した場合 (下記参照))。

default_target_service は、要求にサービスが指定されていない場合、使用されるデフォルトのターゲット・サービスです。ターゲット・サービスは、形式が「/string」で、URIMAP 定義のパスと突き合わせるときに使用します。「/SOAP/test/test1」などがその例です。先頭文字は必ず「/」にする必要があります。

nnn は、行われる再試行の回数です。

requeuename は、障害発生メッセージの送信先キューの名前です。

2. トリガー・プロセスを指定する PROCESS オブジェクトを定義します。 以下のコマンドを使用します。

```
DEFINE
PROCESS(processname)
APPLTYPE(CICS)
APPLICID(CPIL)
```

ここで

processname は、プロセスの名前で、入力キューの定義時に使用した名前と同じにする必要があります。

サービス・リクエスターでのローカル・キューの定義

サービス・リクエスターで、アウトバウンド要求に対して MQ トランスポートを使用する場合は、事前定義された応答キューに応答が戻ることをターゲット Web サービスの URI で指定できます。 こうする場合は、QLOCAL オブジェクトで各応答キューを定義する必要があります。

要求に関連した URI が応答キューを指定していない場合、CICS は応答に動的キューを使用します。

任意: 事前定義の応答キューを指定する各 QLOCAL オブジェクトを定義するには、以下のコマンドを使用します。

```
DEFINE
QLOCAL('reply_queue')
DESCR('description')
BOTHRESH(nnn)
```

ここで

reply_queue は、ローカル・キュー名です。

nnn は、行われる再試行の回数です。

第 6 章 CICS での Web サービス使用の計画立案

CICS で Web サービスを使用する計画を立てるには、その前に以下の問題をアプリケーションごとに検討する必要があります。

サービス・プロバイダーまたはサービス・リクエスターの役割で CICS アプリケーションを配置する計画ですか?

Web サービスの CICS サポートを使用して接続することを求められている 1 組のアプリケーションが存在する場合があります。この場合、一方のアプリケーションはサービス・プロバイダー、もう一方はサービス・リクエスターになります。

既存のアプリケーション・プログラムを使用する計画ですか、それとも新規のアプリケーションを作成する計画ですか?

既存のアプリケーションが、適切に定義されたビジネス・ロジックのインターフェースを使用して設計されている場合は、このアプリケーションを、サービス・プロバイダーまたはサービス・リクエスターとして Web サービス設定に使用できる確率が高くなります。ただし、ほとんどの場合は、ビジネス・ロジックを Web サービス・ロジックに接続するラッパー・プログラムを作成する必要があります。

新規アプリケーションの作成を計画している場合は、ビジネス・ロジックを Web サービス・ロジックから分離した状態を維持するようにします。さらにこの場合も、この分離状態を実現するためにラッパー・プログラムを作成する必要があります。ただし、アプリケーションが Web サービスを考慮して設計されている場合、ラッパーは簡単に作成できる可能性が高くなります。

SOAP メッセージを使用する予定ですか?

SOAP は、Web サービス・アーキテクチャーの基本であり、CICS で提供されているサポートの多くでは、SOAP の使用が前提となっています。ただし、他のメッセージ・フォーマットを使用したい状況も考えられます。例えば、CICS Web サービス・インフラストラクチャーを使用して配置する独自のメッセージ・フォーマットを作成してある場合などです。CICS では、こうした処理が可能ですが、Web サービス・アシスタント、SOAP メッセージ・ハンドラーなど、CICS が提供する機能の一部は使用できなくなります。

データ構造と SOAP メッセージ間のマッピングを生成するために CICS Web サービス・アシスタントを使用する予定ですか?

Web サービス・アシスタントは、アプリケーションを Web サービス設定に迅速に配置する機能を備えています。その際、追加のプログラミングはほとんど必要ありません。さらに、追加のプログラミングが必要な場合でも、通常は簡単で、既存のビジネス・ロジックを変更せずに済みます。

ただし、Web サービス・アシスタントを使用せずに処理した方がうまく処理できる場合があります。例えば、データ構造を SOAP メッセージにマップする既存のコードがある場合は、Web サービス・アシスタントを使用してアプリケーションを再構築しても、メリットはありません。

既存のサービス記述を使用する予定ですか、それとも新規のサービス記述を作成する予定ですか?

状況によっては、既存のサービス記述を開始点として使用する必要があります。例を次に示します。

- アプリケーションはサービス・リクエスターであり、既存の Web サービスを呼び出すよう設計されている。
- アプリケーションはサービス・プロバイダーであり、既存の業界標準サービス記述にこのアプリケーションを適合させることを目的としている。

その他の状況では、アプリケーションに応じて新規のサービス記述を作成する必要があります。

次のステップ:

- サービス・プロバイダーの計画
- サービス・リクエスターの計画

サービス・プロバイダー・アプリケーションの計画

一般に、CICS アプリケーションは、ビジネス・ロジックとコミュニケーション・ロジックを確実に分離できるよう構造化する必要があります。 この手法に従うと、新規および既存のアプリケーションを Web サービス・プロバイダーで直接的に配置するのに役立ちます。 状況によっては、アプリケーション・プログラムと CICS Web サービス・サポートとの間に単純なラッパー・プログラムを介在させる必要があります。

図 15 には、コミュニケーション・ロジックとビジネス・ロジックとを確実に分離するために分割された標準的なアプリケーションを示します。

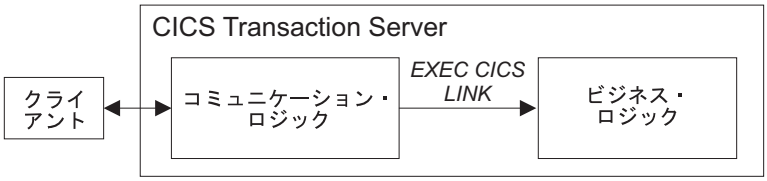


図 15. コミュニケーション・ロジックとビジネス・ロジックに分割されたアプリケーション

多くの場合、ビジネス・ロジックは、サービス・プロバイダー・アプリケーションの場合と同様に直接配置できます。 このことは、図 16 に図示されています。

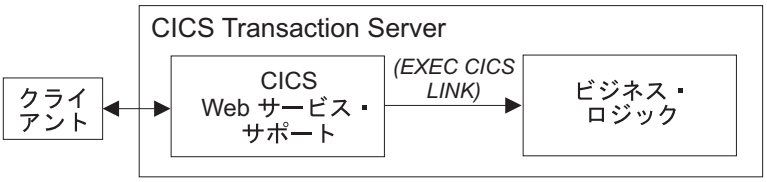


図 16. Web サービス・プロバイダーとしての CICS アプリケーションの単純な配置

この単純なモデルを使用する場合は、以下の条件が適用されます。

CICS Web サービス・アシスタントを使用して SOAP メッセージとアプリケーション・データ構造間のマッピングを生成する場合:

ビジネス・ロジックのインターフェースで使用されるデータ・タイプは、CICS Web サービス・アシスタントによってサポートされている必要があります。これに該当しない場合は、CICS Web サービス・サポートとビジネス・ロジックとの間にラッパー・プログラムを介在させる必要があります。

既存のプログラムを配置して既存の Web サービス記述に適合するサービスを提供する場合は、ラッパー・プログラムも必要になります。Web サービス・アシスタントを使用して Web サービス記述を処理すると、結果として得られるデータ構造がビジネス・ロジックのインターフェースと一致する可能性は非常に低くなります。

CICS Web サービス・アシスタントを使用していない場合:

サービス・プロバイダー・パイプラインに存在するメッセージ・ハンドラーは、ビジネス・ロジックと直接対話する必要があります。

ラッパー・プログラムの使用

CICS Web サービス・アシスタントではビジネス・ロジックと直接対話するためのコードを生成できない場合は、ラッパー・プログラムを使用します。例えば、ビジネス・ロジックのインターフェースは、CICS Web サービス・アシスタントがSOAPメッセージに直接マップできないデータ構造を使用する可能性があります。この状況では、ラッパー・プログラムを使用すると、必要なデータ操作を追加できます。

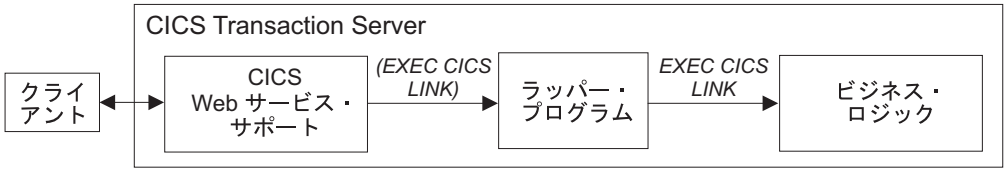


図 17. ラッパー・プログラム使用による、Web サービス・プロバイダーとしての CICS アプリケーションの配置

アシスタントがサポートできる 2 番目のデータ構造を設計して、これをラッパー・プログラムのインターフェースとして使用する必要があります。この結果、ラッパー・プログラムが実行する機能は、以下に示す 2 つの単純な機能となります。

- 2 つのデータ構造間でのデータの移動
- 既存のインターフェースによるビジネス・ロジックの呼び出し

サービス・リクエスター・アプリケーションの計画

一般に、CICS アプリケーションは、ビジネス・ロジックとコミュニケーション・ロジックを確実に分離できるよう構造化する必要があります。 この手法に従うと、新規および既存のアプリケーションを Web サービス・リクエスターで直接的に配置するのに役立ちます。 ほとんどすべての状況では、アプリケーション・プログラムと CICS Web サービス・サポートとの間に単純なラッパー・プログラムを介在させる必要があります。

図 18 には、コミュニケーション・ロジックとビジネス・ロジックとを確実に分離するために分割された標準的なアプリケーションを示します。このアプリケーションは、Web サービス・リクエスターでビジネス・ロジックを再使用するために最適な構造になっています。

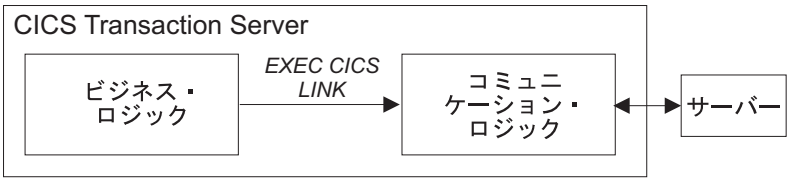


図 18. コミュニケーション・ロジックとビジネス・ロジックに分割されたアプリケーション

この状況では、既存の EXEC CICS LINK コマンドを使用して CICS Web サービス・サポートを呼び出すことはできません。

- CICS Web サービス・アシスタントを使用して SOAP メッセージとアプリケーション・データ構造間のマッピングを生成する場合は、EXEC CICS INVOKE WEBSERVICE コマンドを使用して、アプリケーションのデータ構造を CICS Web サービス・サポートに渡す必要があります。また、ビジネス・ロジックのインターフェースで使用されるデータ・タイプは、CICS Web サービス・アシスタントによってサポートされている必要があります。
- CICS Web サービス・アシスタントを使用していない場合は、独自のメッセージを作成して、プログラム DFHPIRT にリンクする必要があります。

このため、いずれの場合でも、プログラムを変更する準備が整っていないかぎり、ビジネス・ロジックは Web サービスを直接呼び出すことができないことになります。 Web サービス・アシスタントの場合、このオプションは 図 19 に示されていますが、いずれの場合もお勧めできません。

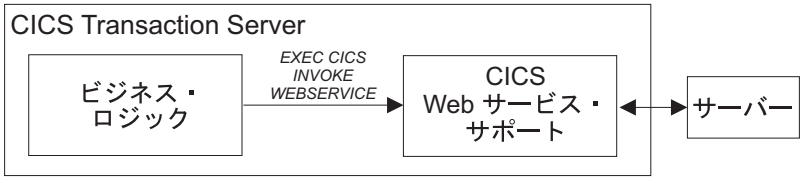


図 19. Web サービス・リクエスターとしての CICS アプリケーションの単純な配置

ラッパー・プログラムの使用

ビジネス・ロジックを未変更のまま維持する、より優れた解決策は、ラッパー・プログラムを使用することです。 この場合、ラッパー・プログラムには次の 2 つの目的があります。

- ラッパー・プログラムは、ビジネス・ロジックの代わりに、EXEC CICS INVOKE WEBSERVICE コマンド、つまり EXEC CICS LINK PROGRAM(DFHPIRT) を発行します。ビジネス・ロジックは未変更のままです。
- CICS Web サービス・アシスタントが SOAP メッセージに直接マップできないデータ構造をアプリケーションが使用する場合、ラッパー・プログラムは、これに必要なデータ操作を必要に応じて提供できます。

Web サービス・アシスタントが使用された場合、この構造は 図 20 に示されます。

図 20. ラッパー・プログラム使用による、Web サービス・リクエスターとしての CICS アプリケーションの配置

第 7 章 CICS Web サービス・アシスタント

CICS Web サービス・アシスタントとは、1 組のバッチ・ユーティリティーで、既存の CICS® アプリケーションを Web サービスに変換するのに役立ちます。また、これを使用すると、CICS アプリケーションが、外部のプロバイダーによって提供された Web サービスを使用できるようになります。このアシスタントは、サービス・プロバイダーやサービス・リクエスターが使用するための CICS アプリケーションの迅速な配置をサポートしており、プログラミングの労力が最小限で済みます。

CICS の Web サービス・アシスタントを使用する場合は、インバウンド・メッセージを解析し、アウトバウンド・メッセージを作成するための独自のコードを記述する必要はありません。CICS は、SOAP メッセージ本文とアプリケーション・プログラムのデータ構造間でデータをマップするからです。

リソース定義は、その大半が自動的に生成およびインストールされます。PIPELINE リソースを定義する必要はありますが、多くの場合は、CICS 提供のパイプライン構成ファイルを使用できます。この内容は次のとおりです。

basicsoap11provider.xml
: SOAP 1.1 メッセージ・ハンドラーを使用するサービス・プロバイダー用のパイプライン構成ファイル

basicsoap11requester.xml
: SOAP 1.1 メッセージ・ハンドラーを使用するサービス・リクエスター用のパイプライン構成ファイル

このアシスタントでは、単純な言語構造から WSDL 文書を作成したり、既存の WSDL 文書から言語構造を作成したりすることができ、COBOL、C/C++、および PL/I をサポートします。また、SOAP メッセージからコンテナーおよび COMMAREA への自動ランタイム変換、さらに、*この逆の変換* を可能にするために使用される情報も生成します。

ただし、アシスタントは起こりうることすべてに対処できるわけではなく、別の方法で対応することが必要な場合もあります。例を次に示します。

SOAP メッセージを使用しない場合
: SOAP 以外のプロトコルをメッセージに使用することが望ましい場合は、そうすることができます。ただし、アプリケーション・プログラムには、インバウンド・メッセージの解析とアウトバウンド・メッセージの作成を行う役割が与えられます。

SOAP メッセージは使用するが、CICS による解析はしない場合
: インバウンド・メッセージの場合、アシスタントは SOAP 本体をアプリケーション・データ構造にマップします。アプリケーションによっては、ユーザー自身による SOAP 本体の解析が必要な場合があります。

CICS Web サービス・アシスタントが、アプリケーションのデータ構造をサポートしない場合
: CICS Web サービス・アシスタントは、最も一般的なデータ・タイプおよびデータ構造をサポートしますが、サポートされていないデータ・タイプやデ

ータ構造もいくつかあります。この状況では、アプリケーションのデータを、アシスタントがサポートできる形式にマップするプログラム層を準備することを最初に考慮する必要があります。これが不可能な場合は、自分でメッセージを解析する必要があります。

CICS Web サービス・アシスタントを使用しないことにした場合は、以下の作業が必要です。

- インバウンド・メッセージを解析し、アウトバウンド・メッセージを作成するための独自のコードを準備する
- 独自のパイプライン構成ファイルを準備する
- 独自の URIMAP リソースおよび PIPELINE リソースを定義してインストールする

CICS Web サービス・アシスタントは、以下の 2 つのユーティリティー・プログラムで構成されています。

DFHLS2WS
言語構造を基にして Web サービス・バインディング・ファイルを生成します。このユーティリティーは、Web サービス記述も生成します。

DFHWS2LS
Web サービス記述を基にして Web サービス・バインディング・ファイルを生成します。このユーティリティーは、アプリケーション・プログラムで使用できる言語構造も生成します。

hlq.XDFHINST ライブラリー内に両方のプログラムを実行する JCL プロシージャーがあります。

CICS Web サービス・アシスタントの使用によるサービス・プロバイダー・アプリケーションの配置

CICS Web サービス・アシスタントは、サービス・プロバイダーの設定における CICS アプリケーションの配置タスクを単純化します。

このアシスタントを使用し、CICS アプリケーションをサービス・プロバイダーとして配置する場合は、以下の 2 つのオプションがあります。

- Web サービス記述から開始し、Web サービス・アシスタントを使用して言語データ構造を生成する。

  既存の Web サービス記述に適合するサービス・プロバイダーを実装する場合は、このオプションを使用します。
- 言語データ構造から開始し、Web サービス・アシスタントを使用して Web サービス記述を生成する。

  既存のプログラムを Web サービスとして公開しており、このプログラムのインターフェースの外観を Web サービス記述と SOAP メッセージで公開しようとする場合は、このオプションを使用します。

Web サービス記述を基にしたサービス・プロバイダー・アプリケーションの作成

CICS Web サービス・アシスタントを使用すると、Web サービス記述を基にしてサービス・プロバイダー・アプリケーションを作成できます。

Web サービス記述は、HFS のファイルに置く必要があります。

1. Web サービス・バインディング・ファイルを生成します。 Web サービス・バインディング・ファイルを生成する場合は、バッチ・プログラム DFHWS2LS を使用します。Web サービス・バインディング・ファイルの場合と同様に、このプログラムは言語データ構造を生成します。
2. Web サービス・バインディング・ファイルを、Web サービス・アプリケーションに使用する PIPELINE リソースのピックアップ・ディレクトリーにコピーします。
3. ラッパー・プログラムを作成する場合は、1 で生成した言語データ構造を使用します。 ラッパー・プログラムは、ビジネス・ロジックと対話するために、データを正しい形式に処理します。
4. システムに適切な PIPELINE リソース定義が存在しない場合は、作成してインストールしてください。 PIPELINE リソースによって指定される XML ファイルでは、インバウンド要求および応答を処理するために使用されるメッセージ・ハンドラーが定義されます。通常は、多くのアプリケーションが同じ PIPELINE 定義を使用できるため、システムに適切な PIPELINE がすでに存在する場合は、このステップを実行する必要はありません。
5. Web サービスを呼び出すときに使用する URI と一致する URIMAP を作成してインストールします。 URIMAP は、Web サービス要求の処理方法の詳細を示す WEBSERVICE リソースと PIPELINE リソースの名前を指定します。

   URIMAP は、スキャン機構によって自動的に作成できます。この場合、CICS は、URIMAP の作成に必要な情報を Web サービス・バインディング・ファイルから取得します。
6. WSDL および WSBIND ファイルの場所を指定する WEBSERVICE を作成してインストールします。 WEBSERVICE の作成には RDO を使用できますが、お勧めの方法は、WSBIND ファイルをスキャンすることです。こうすることにより、WSDL と整合した WEBSERVICE 定義を作成できます。

データ構造を基にしたサービス・プロバイダー・アプリケーションの作成

CICS Web サービス・アシスタントを使用すると、高水準言語のデータ構造を基にしてサービス・プロバイダー・アプリケーションを作成できます。

高水準言語のデータ構造を処理するには、その前に以下のことを確認する必要があります。

- データ構造は、ソース・プログラムとは別個に定義されている (例えば、COBOL コピーブック内で定義)。
- アプリケーション・プログラムが使用するデータ構造が入力と出力で異なる場合、このデータ構造は、区分データ・セットに存在する 2 つの異なるメンバーに定義されます。入力と出力で同じデータ構造を使用している場合は、1 つのメンバーにデータ構造を定義してください。

どのデータ構造を処理するかは、ラッパー・プログラムを使用しているかどうかによって異なります。

- ラッパー・プログラムを使用している場合、コピーブックはラッパー・プログラムのインターフェースになります。
- ラッパー・プログラムを使用していない場合、コピーブックはビジネス・ロジックのインターフェースになります。

1. Web サービス・バインディング・ファイルを生成します。 Web サービス・バインディング・ファイルを生成する場合は、バッチ・プログラム DFHLS2WS を使用します。Web サービス・バインディング・ファイルの場合と同様に、このプログラムは Web サービス記述を生成します。Web サービス記述は、ユーザーのサービスと対話するサービス・リクエスターを作成するときに使用できます。Web サービスに対して SOAP メッセージの実行時検証を実行する場合も Web サービス記述が必要になります。
2. Web サービス・バインディング・ファイルを、Web サービス・アプリケーションに使用する PIPELINE リソースのピックアップ・ディレクトリーにコピーします。
3. システムに適切な PIPELINE リソース定義が存在しない場合は、作成してインストールしてください。 PIPELINE リソースによって指定される XML ファイルでは、インバウンド要求および応答を処理するために使用されるメッセージ・ハンドラーが定義されます。通常は、多くのアプリケーションが同じ PIPELINE 定義を使用できるため、システムに適切な PIPELINE がすでに存在する場合は、このステップを実行する必要はありません。
4. Web サービスを呼び出すときに使用する URI と一致する URIMAP を作成してインストールします。 URIMAP は、Web サービス要求の処理方法の詳細を示す WEBSERVICE リソースと PIPELINE リソースの名前を指定します。

   URIMAP は、スキャン機構によって自動的に作成できます。この場合、CICS は、URIMAP の作成に必要な情報を Web サービス・バインディング・ファイルから取得します。
5. WSDL および WSBIND ファイルの場所を指定する WEBSERVICE を作成してインストールします。 WEBSERVICE の作成には RDO を使用できますが、お勧めの方法は、WSBIND ファイルをスキャンすることです。こうすることにより、WSDL と整合した WEBSERVICE 定義を作成できます。

サービスにアクセスするサービス・リクエスターを作成する必要があるすべてのユーザーが、Web サービス記述を使用できるようにしてください。

CICS Web サービス・アシスタントの使用によるサービス・リクエスター・アプリケーションの配置

CICS Web サービス・アシスタントは、サービス・リクエスターの設定における CICS アプリケーションの配置タスクを単純化します。

CICS Web サービス・アシスタントを使用して、CICS アプリケーションをサービス・リクエスターとして配置する場合は、Web サービス記述から開始し、これを基に言語データ構造を生成することをお勧めします。

サービス・リクエスターの場合、言語データ構造から開始する代わりに、Web サービス記述を生成することはお勧めできません。サービスの記述を提供することは、通常、サービス・プロバイダー側の役割です。

Web サービス記述を基にしたサービス・リクエスター・アプリケーションの作成

CICS Web サービス・アシスタントを使用すると、Web サービス記述を基にしてサービス・リクエスター・アプリケーションを作成できます。

Web サービス記述は、HFS のファイルに置く必要があります。

1. Web サービス・バインディング・ファイルを生成します。 Web サービス・バインディング・ファイルを生成する場合は、バッチ・プログラム DFHWS2LS を使用します。

   重要: DFHWS2LS を使用する場合は、PROGRAM パラメーターを指定しないでください。このパラメーターは、サービス・プロバイダーにのみ適用されます。

   Web サービス・バインディング・ファイルの場合と同様に、このプログラムは言語データ構造を生成します。
2. Web サービス・バインディング・ファイルを、Web サービス・アプリケーションに使用する PIPELINE リソースのピックアップ・ディレクトリーにコピーします。 PIPELINE がサービス・リクエスターに合わせて構成されていること、つまり、CONFIGFILE 属性に指定されている構成ファイルの最上位エレメントが `<requester_pipeline>` であることを確認します。
3. ラッパー・プログラムを作成する場合は、1 で生成した言語データ構造を使用します。 Web サービスと通信する場合は、ラッパー・プログラムで EXEC CICS INVOKE WEBSERVICE コマンドを使用します。このコマンドのオプションは、次のとおりです。
   - WEBSERVICE リソースの名前
   - Web サービスの呼び出し対象となる操作
4. システムに適切なパイプライン構成ファイルが存在しない場合は、作成してください。

   通常は、多くのサービス・リクエスター・アプリケーションが同じパイプライン構成を使用できるため、システムに適切な構成ファイルがすでに存在する場合は、このステップを実行する必要はありません。
5. システムに適切な PIPELINE リソース定義が存在しない場合は、作成してインストールしてください。 PIPELINE リソースは、パイプライン構成ファイルの名前を指定します。

   通常は、多くのサービス・リクエスター・アプリケーションが同じ PIPELINE 定義を使用できるため、システムに適切な PIPELINE がすでに存在する場合は、このステップを実行する必要はありません。
6. WEBSERVICE リソース定義を作成してインストールします。 WEBSERVICE の作成には RDO を使用できますが、お勧めの方法は、WSBIND ファイルをスキャンすることです。 PERFORM PIPELINE SCAN コマンドを使用します。こうすることにより、WSDL と整合した WEBSERVICE 定義を作成できます。

7. コミュニケーション・ロジックを置換できるラッパー・プログラムを作成します。

サービス・プロバイダーに応じた CICS インフラストラクチャーの作成

サービス・プロバイダーに応じた CICS インフラストラクチャーを作成するには、多数の CICS リソースを定義してインストールする必要があります。 多くの場合、CICS はこれらのリソースの一部を自動的に生成できます。

CICS Web サービス・アシスタントのヘルプを使用して配置したサービス・プロバイダー・アプリケーションの場合は、以下を定義する必要があります。

トランスポート・インフラストラクチャー

MQ トランスポートを使用している場合は、入力メッセージを処理するまで入力メッセージを保管する 1 つ以上のローカル・キューと、入力メッセージを処理する CICS トランザクションを指定する 1 つのトリガー・プロセスを定義する必要があります。

HTTP トランスポートを使用している場合は、インバウンド要求を受信するポートの定義に関する情報が記述されている TCPIPSERVICE を定義する必要があります。

PIPELINE リソース定義

この関連パイプライン構成ファイルを使用することにより、PIPELINE リソースは、インバウンド Web サービス要求および応答を処理するときに使用するパイプラインの属性を定義します。通常は 1 つのパイプラインが多数の異なる Web サービスの要求を処理できるため、CICS システムに新規の Web サービスを配置した場合は、既存のパイプラインを使用できます。

PIPELINE リソースは、構成ファイルだけでなく、ピックアップ・ディレクトリーも指定します。このディレクトリーには、Web サービス・バインディング・ファイルが格納されています。

PIPELINE リソースをインストールするか、または (CEMT または CICS システム・プログラミング・インターフェースを使用して) PERFORM PIPELINE SCAN コマンドを発行すると、CICS は、ピックアップ・ディレクトリーに格納されているファイルを読み取り、URIMAP リソースおよび WEBSERVICE リソースを動的に作成します。

URIMAP リソース定義

URIMAP は、Web サービス要求を処理するパイプラインを検索するときに使用します。 URIMAP リソースの定義とインストールは RDO を使用すれば実行可能ですが、このリソースは動的に作成することをお勧めします。

WEBSERVICE リソース定義

WEBSERVICE リソースは、アプリケーションの実行環境を定義します。

WEBSERVICE リソースの定義とインストールは RDO を使用すれば実行可能ですが、このリソースは、CICS Web サービス・アシスタントによって作成された Web サービス・バインディング・ファイルを使用して、動的に作成することをお勧めします。

ご使用のサービス・プロバイダーに応じた CICS インフラストラクチャーを作成するには、以下のステップを実行します。

1. トランスポート・インフラストラクチャーを定義します。 必要な各種のトランスポート構成ごとにこのステップを繰り返します。
2. パイプラインを定義します。 必要な各種のパイプライン構成ごとにこのステップを繰り返します。
3. アプリケーション・プログラムごとに Web サービス・バインディング・ファイルを作成します。 サービス・プロバイダーに使用するパイプラインのピックアップ・ディレクトリーにファイルを置きます。
4. アプリケーション・プログラムごとに URIMAP および WEBSERVICE リソースを作成します。 これを実行するには、PERFORM PIPELINE SCAN コマンドを使用します。 PIPELINE のピックアップ・ディレクトリーに Web サービス・バインディング・ファイルを追加した場合は、このステップを必ず繰り返してください。

これで、CICS システムには、各サービス・プロバイダーに必要な以下のインフラストラクチャーが格納されるようになりました。

- 1 つ以上のトランスポート・インフラストラクチャー
- 1 つ以上のパイプライン
- サービス・プロバイダーごとに以下のリソース
  - URIMAP
  - WEBSERVICE

以下の処理を実行する必要がある場合は、構成を拡張できます。

- 追加のトランスポート・インフラストラクチャーを定義するには、ステップ 1 を繰り返します。
- 追加のパイプラインを作成するには、ステップ 2 を繰り返します。
- Web アプリケーション・プログラムをパイプラインにさらに関連付けるには、ステップ 3 から 4 までを繰り返します。

サービス・リクエスターに応じた CICS インフラストラクチャーの作成

サービス・リクエスターに応じた CICS インフラストラクチャーを作成するには、多数の CICS リソースを定義してインストールする必要があります。 多くの場合、CICS はこれらのリソースの一部を自動的に生成できます。

CICS Web サービス・アシスタントのヘルプを使用して配置したサービス・リクエスター・アプリケーションの場合は、以下を定義する必要があります。

PIPELINE リソース定義

この関連パイプライン構成ファイルを使用することにより、PIPELINE リソースは、アウトバウンドの Web サービス要求および応答を処理するときに使用するパイプラインの属性を定義します。通常は 1 つのパイプラインが多数の異なる Web サービスの要求を処理できるため、CICS システムに新規のサービス・リクエスターを配置した場合は、既存のパイプラインを使用できます。

PIPELINE リソースは、構成ファイルだけでなく、ピックアップ・ディレクトリーも指定します。このディレクトリーには、Web サービス・バインディング・ファイルが格納されています。

PIPELINE リソースをインストールするか、または (CEMT または CICS システム・プログラミング・インターフェースを使用して) PERFORM PIPELINE SCAN コマンドを発行すると、CICS は、ピックアップ・ディレクトリーに格納されているファイルを読み取り、URIMAP リソースおよび WEBSERVICE リソースを動的に作成します。

WEBSERVICE リソース定義

WEBSERVICE リソースによって定義される実行環境では、CICS アプリケーションを Web サービス・リクエスターとして実行できます。
WEBSERVICE リソースの定義とインストールは RDO を使用すれば実行可能ですが、このリソースは動的に作成することをお勧めします。
WEBSERVICE リソース定義は、ターゲットの Web サービスごとに 1 つ存在します。

ご使用のサービス・リクエスターに応じた CICS インフラストラクチャーを作成するには、以下のステップを実行します。

1. パイプラインを定義します。 必要な各種のパイプライン構成ごとにこのステップを繰り返します。
2. サービス・リクエスター・アプリケーションごとに Web サービス・バインディング・ファイルを作成します。 サービス・リクエスターに使用するパイプラインのピックアップ・ディレクトリーにファイルを置きます。
3. アプリケーション・プログラムごとに WEBSERVICE リソースを作成します。これを実行するには、PERFORM PIPELINE SCAN コマンドを使用します。PIPELINE のピックアップ・ディレクトリーに Web サービス・バインディング・ファイルを追加した場合は、このステップを必ず繰り返してください。

これで、CICS システムには、各サービス・リクエスターに必要な以下のインフラストラクチャーが格納されるようになりました。

- 1 つ以上のパイプライン
- サービス・リクエスターごとに以下のリソース
  - WEBSERVICE

以下の処理を実行する必要がある場合は、構成を拡張できます。

- 追加のパイプラインを作成するには、ステップ 1 を繰り返します。
- Web サービス・リクエスター・アプリケーションをパイプラインにさらに関連付けるには、2 から 3 までを繰り返します。

SOAP メッセージの検証

CICS Web サービス・アシスタントを使用してアプリケーションを配置する場合は、Web サービス記述に格納されているスキーマに SOAPメッセージが適合していることを確認するために、SOAP メッセージを実行時に検証することを指定できます。

CICS では、Java プログラムを使用して SOAP メッセージを検証します。したがって、検証を実行するために、CICS 領域で Java サポートを使用可能にしておく必要があります。

SOAP メッセージの検証は、SOAP メッセージがアプリケーション・データ構造に変換されるポイントで行われます。 検証をオフにすると、CICS が SOAP メッセージを検証する範囲は、メッセージが適切な形式の XML を含むことを確認し、メッセージを変換するために必要な範囲に限定されます。

重要: Web サービス配置の開発およびテスト時には、完全な検証を実行すると、サービス・リクエスターとサービス・プロバイダー間のメッセージ交換での問題検出に役立ちます。ただし、SOAP メッセージの完全な検証を実行すると、それに関連した相当なオーバーヘッドが生じるため、テストが完全に行われる実動アプリケーションでメッセージを検証するのはお勧めできません。

SOAP メッセージを検証するには、以下のステップを実行します。

1. Web サービス記述を WEBSERVICE リソースに関連付けてあることを確認します。 これは、PIPELINE のピックアップ・ディレクトリーがスキャンされたときに、このディレクトリーに Web サービス記述が存在した場合、自動的に作成された WEBSERVICE リソース定義の場合になります。

   RDO で作成された WEBSERVICE 定義の場合、Web サービス記述は WSDLFILE 属性で指定されます。
2. WEBSERVICE の検証をオンにします。次の CEMT コマンドまたは SPI コマンド: `SET WEBSERVICE(name) VALIDATION` を使用します。 RDO で定義されている WEBSERVICE では、検証が必要かどうかを VALIDATION 属性で指定できますが、この設定は、WEBSERVICE のインストール後に、`SET WEBSERVICE` コマンドを使用すると変更できます。

Web サービスの検証が必要なくなったら、次のコマンド: `SET WEBSERVICE(name) NOVALIDATION` を使用して検証をオフにします。

DFHLS2WS: 高水準言語から WSDL への変換

カタログ式プロシージャー DFHLS2WS は、高水準言語データ構造を基にして、Web サービス記述および Web サービス・バインディング・ファイルを生成します。 CICS アプリケーション・プログラムをサービス・プロバイダーとして公開する場合に、DFHLS2WS を使用することができます。

DFHLS2WS のジョブ制御ステートメント

JOB ジョブを開始します。

EXEC プロシージャー名 (DFHLS2WS) を指定します。

DFHLS2WS には、Java 仮想マシン (JVM) を実行するのに十分な記憶域が必要です。EXEC ステートメントで `REGION=0M` と指定することをお勧めします。

INPUT.SYSUT1 DD
入力を指定します。入力パラメーターは、通常、入力ストリーム内に指定します。ただし、データ・セットや区分データ・セットのメンバーに定義することもできます。

シンボリック・パラメーター

以下のシンボリック・パラメーターは、カタログ式プロシージャー DFHLS2WS で定義されます。

JAVADIR=*path*
DFHLS2WS によって使用される Java ディレクトリーの名前を指定します。このパラメーターの値は /usr/lpp/ に追加され、これによって /usr/lpp/*path* という完全なパス名が得られます。

通常、このパラメーターを指定する必要はありません。デフォルト値は、**JAVADIR** パラメーターの CICS インストール・ジョブ (DFHISTAR) に指定された値です。

USSDIR=*path*
UNIX システム・サービス HFS の CICS TS ディレクトリーの名前を指定します。このパラメーターの値は /usr/lpp/cicsts/ に追加され、これによって /usr/lpp/cicsts/*path* という完全なパス名が得られます。

通常、このパラメーターを指定する必要はありません。デフォルト値は、**USSDIR** パラメーターの CICS インストール・ジョブ (DFHISTAR) に指定された値です。

TMPDIR=*tmpdir*
DFHLS2WS が一時ワークスペースとして使用する HFS のディレクトリーの場所を指定します。このジョブを実行するユーザー ID には、このディレクトリーに対する読み取り権限および書き込み権限が必要です。

デフォルト値は /tmp です。

TMPFILE=*tmpprefix*
一時ワークスペース・ファイルの名前を作成するために DFHLS2WS が使用する接頭部を指定します。

デフォルト値は DFHLS2WS です。

一時ワークスペース

DFHLS2WS は、実行時に、次の 3 つの一時ファイルを作成します。

tmpdir/*tmpprefix*.in

tmpdir/*tmpprefix*.out

tmpdir/*tmpprefix*.err

ここで

tmpdir は、**TMPDIR** パラメーターに指定されている値です。

tmpprefix は、**TMPFILE** パラメーターに指定されている値です。

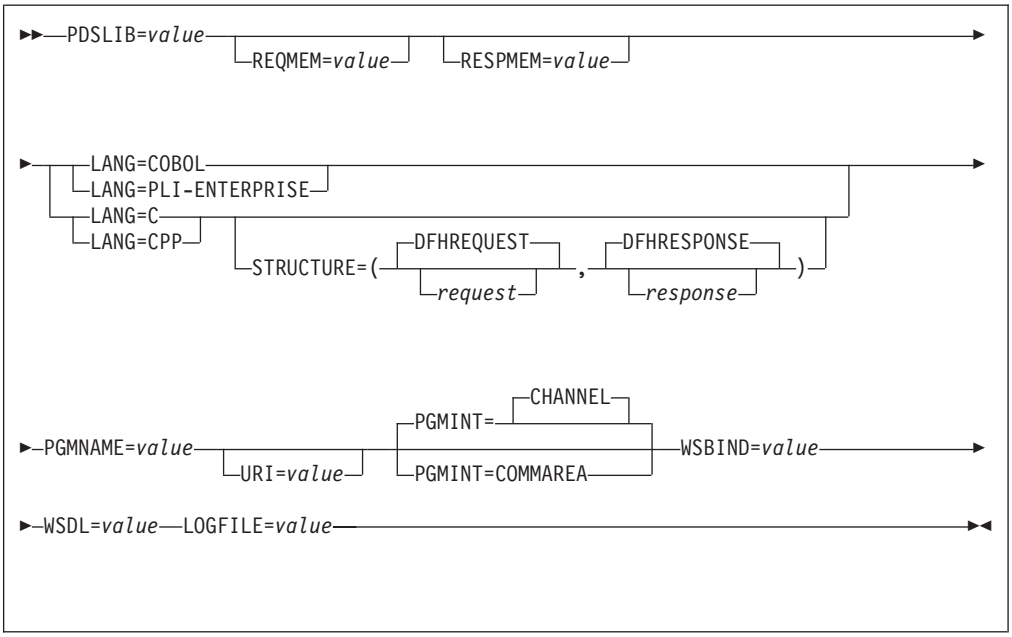
ファイルのデフォルトの名前は (**TMPDIR** および **TMPFILE** が指定されていない場合)、次のとおりです。

```
/tmp/DFHLS2WS.in
/tmp/DFHLS2WS.out
/tmp/DFHLS2WS.err
```

重要: DFHLS2WS は、生成された HFS ファイル名へのアクセスをロックしません。したがって、DFHLS2WS のインスタンスが同時に 2 つ以上動作し、同じ一時ワークスペース・ファイルを使用した場合、あるジョブが一時ワークスペース・ファイルを使用中に別のジョブがそのファイルを上書きするのを防ぐ方法はありません。この状況は、予測不能な障害の発生につながります。

したがって、この状況を回避できる命名規則および操作手順を考案することをお勧めします。例えば、システム・シンボリック・パラメーター SYSUID を使用すると、個々のユーザーに対して一意のワークスペース・ファイルを生成できます。

DFHLS2WS のパラメーターの入力



パラメーターの使用法

- 入力パラメーターの指定順序は自由です。
- 各パラメーターは改行後に記述を始める必要があります。
- パラメーターが長すぎて 1 行に収まらない場合は、行の末尾にアスタリスク文字 (*) を使用して、そのパラメーターが次の行に続くことを示します。アスタリスクより前の文字はすべて (スペースを含む) パラメーターの一部とみなされます。例を次に示します。

```
WSBIND=wsbinddir*
/app1
```

このコードは、次のコードと同じ意味になります。

```
WSBIND=wsbinddir/app1
```

- 行の先頭の文字の位置に # という文字がある場合、この文字はコメント文字を表します。この行は無視されます。

パラメーターの記述

LANG=COBOL
高水準言語構造のプログラム言語が COBOL であることを指定します。

LANG=PLI-ENTERPRISE
高水準言語構造のプログラム言語が Enterprise PL/I であることを指定します。

LANG=C
高水準言語構造のプログラム言語が C であることを指定します。

LANG=CPP
高水準言語構造のプログラム言語が C++ であることを指定します。

PDSLIB=*value*
処理の対象となる高水準言語データ構造が格納されている区分データ・セットの名前を指定します。要求および応答に使用されるデータ・セット・メンバーは、それぞれ、**REQMEM** パラメーターおよび **RESPMEM** パラメーターで指定されます。

制約事項: 区分データ・セット内のレコードは、80 バイトの固定長にする必要があります。

PGMINT=CHANNEL|COMMAREA
サービス・プロバイダーの場合は、CICS がターゲット・アプリケーション・プログラムにデータを渡す方法を次のように指定します。

CHANNEL
CICS は、チャネル・インターフェースを使用して、データをターゲット・アプリケーション・プログラムに渡します。

COMMAREA
CICS は、通信域を使用して、データをターゲット・アプリケーション・プログラムに渡します。

DFHLS2WS からの出力がサービス・リクエスターで使用される場合、このパラメーターは無視されます。

PGMNAME=*value*
Web サービスとして公開されるターゲット CICS アプリケーション・プログラムの名前を指定します。これは、CICS Web サービス・サポートのリンク先となるプログラムです。

REQMEM=*value*
Web サービス要求の高水準言語データ構造が格納されている区分データ・セット・メンバーの名前を指定します。
- サービス・プロバイダーの場合、Web サービス要求は、アプリケーション・プログラムの入力になります。
- サービス・リクエスターの場合、Web サービス要求は、アプリケーション・プログラムの出力になります。

RESPMEM=*value*

Web サービス応答の高水準言語データ構造が格納されている区分データ・セット・メンバーの名前を指定します。

- サービス・プロバイダーの場合、Web サービス応答は、アプリケーション・プログラムの出力になります。
- サービス・リクエスターの場合、Web サービス応答は、アプリケーション・プログラムの入力になります。

応答がない場合 (つまり、片方向のメッセージの場合) は、このパラメーターを省略します。

STRUCTURE=(*request*,*response*)

C と C++ の場合にのみ、REQMEM および RESPMEM パラメーターに指定された区分データ・セットのメンバーに格納されている高水準言語データ構造の名前を指定します。

request

REQMEM パラメーターを指定する場合に、要求を格納する高水準言語データ構造の名前を指定します。デフォルト値は DFHREQUEST です。

区分データ・セット・メンバーは、指定した名前を持つ高水準言語データ構造 (名前を指定しない場合は DFHREQUEST という名前の構造) を格納している必要があります。

response

RESPMEM パラメーターを指定する場合に、応答を格納する高水準言語データ構造の名前を指定します。デフォルト値は DFHRESPONSE です。

値を指定する場合、区分データ・セット・メンバーが、指定した名前を持つ高水準言語データ構造 (名前を指定しない場合は DFHRESPONSE という名前の構造) を格納している必要があります。

URI=*value*

サービス・プロバイダーでは、このパラメーターはクライアントが Web サービスのアクセスに使用することになる相対 URI を指定します。CICS は、DFHLS2WS によって作成された Web サービス・バインディング・ファイルから URIMAP リソースを生成するときに指定された値を使用します。このパラメーターは URIMAP 定義が適用される URI のパスの構成エレメントを指定します。

WSBIND=*value*

Web サービス・バインディング・ファイルの完全修飾 HFS 名です。DFHLS2WS は、このファイルが存在しない場合、ファイルを作成します (ディレクトリー構造は作成しません)。

WSDL=*value*

Web サービス記述を書き込むファイルの完全修飾 HFS 名です。DFHLS2WS は、このファイルが存在しない場合、ファイルを作成します (ディレクトリー構造は作成しません)。

LOGFILE=*value*

DFHLS2WS がアクティビティー・ログとトレース情報を書き込むファイルの完全修飾 HFS 名です。 DFHLS2WS は、このファイルが存在しない場合、ファイルを作成します (ディレクトリー構造は作成しません)。

通常はこのファイルを使用する必要はありませんが、DFHLS2WS に問題が発生した場合、このファイルの提出を IBM のサービス組織から依頼される場合があります。

その他の情報

- DFHLS2WS の実行に使用するユーザー ID をOMVS に定義する必要があります。このユーザー ID には、CICS HFS ファイル構造に対する読み取り権限と、LOGFILE パラメーターに指定されたディレクトリーに対する書き込み権限が必要です。
- Java を実行するため、このユーザー ID には十分な大きさのストレージを割り振る必要があります。

例

```
//LS2WS JOB  'accounting information',name,MSGCLASS=A
//  SET QT=''''
//JAVAPROG EXEC DFHLS2WS,
// TMPFILE=&QT.&SYSUID.&QT
//INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
REQMEM=DFH0XCP4
RESPMEM=DFH0XCP4
LANG=COBOL
PGMNAME=DFH0XCMN
URI=exampleApp/inquireSingle
PGMINT=COMMAREA
WSBIND=/u/exampleapp/wsbind/inquireSingle.wsbind
WSDL=/u/exampleapp/wsdl/inquireSingle.wsdl
/*
```

DFHWS2LS: WSDL から高水準言語への変換

カタログ式プロシージャー DFHWS2LS は、Web サービス記述を基にして、高水準言語データ構造および Web サービス・バインディング・ファイルを生成します。CICS アプリケーション・プログラムをサービス・プロバイダーとして公開する場合、またはサービス・リクエスターを作成する場合に、DFHWS2LS を使用することができます。

DFHWS2LS のジョブ制御ステートメント

JOB ジョブを開始します。

EXEC プロシージャー名 (DFHWS2LS) を指定します。

DFHWS2LS には、Java 仮想マシン (JVM) を実行するのに十分なストレージが必要です。EXEC ステートメントで `REGION=0M` と指定することをお勧めします。

INPUT.SYSUT1 DD

入力を指定します。入力パラメーターは、通常、入力ストリーム内に指定します。ただし、データ・セットや区分データ・セットのメンバーに定義することもできます。

シンボリック・パラメーター

以下のシンボリック・パラメーターは、カタログ式プロシージャー DFHWS2LS で定義されます。

JAVADIR=*path*

DFHWS2LS によって使用される Java ディレクトリーの名前を指定します。このパラメーターの値は /usr/lpp/ に追加され、これによって /usr/lpp/*path* という完全なパス名が得られます。

通常、このパラメーターを指定する必要はありません。デフォルト値は、**JAVADIR** パラメーターの CICS インストール・ジョブ (DFHISTAR) に指定された値です。

USSDIR=*path*

UNIX システム・サービス HFS の CICS TS ディレクトリーの名前を指定します。このパラメーターの値は /usr/lpp/cicsts/ に追加され、これによって /usr/lpp/cicsts/*path* という完全なパス名が得られます。

通常、このパラメーターを指定する必要はありません。デフォルト値は、**USSDIR** パラメーターの CICS インストール・ジョブ (DFHISTAR) に指定された値です。

TMPDIR=*tmpdir*

DFHWS2LS が一時ワークスペースとして使用する HFS のディレクトリーの場所を指定します。このジョブを実行するユーザー ID には、このディレクトリーに対する読み取り権限および書き込み権限が必要です。

デフォルト値は /tmp です。

TMPFILE=*tmpprefix*

一時ワークスペース・ファイルの名前を作成するために DFHWS2LS が使用する接頭部を指定します。

デフォルト値は DFHWS2LS です。

一時ワークスペース

DFHWS2LS は、実行時に、次の 3 つの一時ファイルを作成します。

tmpdir/*tmpprefix*.in

tmpdir/*tmpprefix*.out

tmpdir/*tmpprefix*.err

ここで

tmpdir は、**TMPDIR** パラメーターに指定されている値です。

tmpprefix は、**TMPFILE** パラメーターに指定されている値です。

ファイルのデフォルトの名前は (**TMPDIR** および **TMPFILE** が指定されていない場合)、次のとおりです。

/tmp/DFHWS2LS.in

/tmp/DFHWS2LS.out

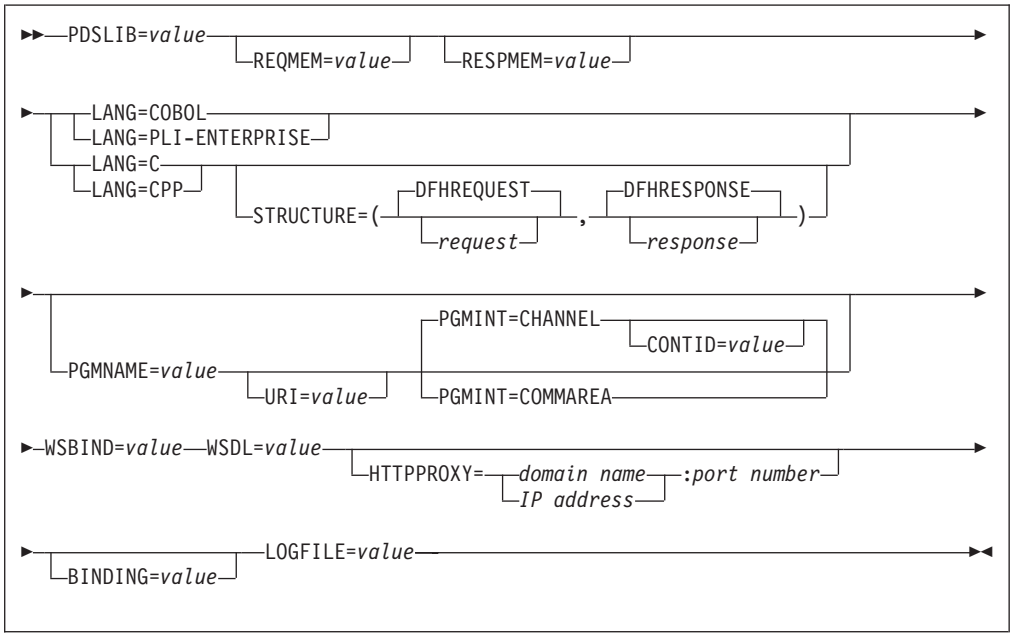
/tmp/DFHWS2LS.err

重要: DFHWS2LS は、生成された HFS ファイル名へのアクセスをロックしません。したがって、DFHWS2LS のインスタンスが同時に 2 つ以上動作し、同じ一時ワークスペース・ファイルを使用した場合、あるジョブが一時ワーク

スペース・ファイルを使用中に別のジョブがそのファイルを上書きするのを防ぐ方法はありません。この状況は、予測不能な障害の発生につながります。

したがって、この状況を回避できる命名規則および操作手順を考案することをお勧めします。例えば、システム・シンボリック・パラメーター SYSUID を使用すると、個々のユーザーに対して一意のワークスペース・ファイルを生成できます。

DFHWS2LS のパラメーターの入力



パラメーターの使用法

- 入力パラメーターの指定順序は自由です。
- 各パラメーターは改行後に記述を始める必要があります。
- パラメーターが長すぎて 1 行に収まらない場合は、行の末尾にアスタリスク文字 (*) を使用して、そのパラメーターが次の行に続くことを示します。アスタリスクより前の文字はすべて (スペースを含む) パラメーターの一部とみなされます。例を次に示します。

  ```
  WSBIND=wsbinddir*
  /app1
  ```

  このコードは、次のコードと同じ意味になります。

  ```
  WSBIND=wsbinddir/app1
  ```

- 行の先頭の文字の位置に # という文字がある場合、この文字はコメント文字を表します。この行は無視されます。

パラメーターの記述

HTTPPROXY={*domain namelIP address*}:*port number*

WSDL に、インターネット上にある他の WSDL ファイルへの参照が含まれており、DFHWS2LS を実行しているシステムがプロキシー・サーバーを使用して

インターネットにアクセスする場合は、そのプロキシー・サーバーのドメイン名、IP アドレス、およびポート番号を指定します。例を次に示します。

```
HTTPPROXY=proxy.example.com:8080
```

その他の場合、このパラメーターは必要ありません。

LANG=COBOL
高水準言語構造のプログラム言語が COBOL であることを指定します。

LANG=PLI-ENTERPRISE
高水準言語構造のプログラム言語が Enterprise PL/I であることを指定します。

LANG=C
高水準言語構造のプログラム言語が C であることを指定します。

LANG=CPP
高水準言語構造のプログラム言語が C++ であることを指定します。

PDSLIB=*value*
生成された高水準言語を含む区分データ・セットの名前を指定します。要求および応答に使用されるデータ・セット・メンバーは、それぞれ、**REQMEM** パラメーターおよび **RESPMEM** パラメーターで指定されます。

PGMINT=CHANNEL|COMMAREA
サービス・プロバイダーの場合は、CICS がターゲット・アプリケーション・プログラムにデータを渡す方法を次のように指定します。

CHANNEL
CICS は、チャネル・インターフェースを使用して、データをターゲット・アプリケーション・プログラムに渡します。

COMMAREA
CICS は、通信域を使用して、データをターゲット・アプリケーション・プログラムに渡します。

DFHWS2LS からの出力がサービス・リクエスターで使用される場合、このパラメーターは無視されます。

PGMNAME=*value*
このパラメーターは、CICS プログラムの名前を指定します。

サービス・プロバイダーで使用される Web サービス・バインディング・ファイルを生成するために DFHWS2LS を使用する場合、このパラメーターを必ず指定する必要があります。このパラメーターは、Web サービスとして公開するアプリケーション・プログラムの名前を指定します。

サービス・リクエスターで使用される Web サービス・バインディング・ファイルを生成するために DFHWS2LS を使用する場合、このパラメーターを省略する必要があります。

REQMEM=*value*
以下に示す Web サービス要求の高水準言語構造体が格納されている区分データ・セット・メンバーの名前を生成するときに DFHWS2LS が使用する 1 から 6 文字の接頭部を指定します。
- サービス・プロバイダーの場合、Web サービス要求は、アプリケーション・プログラムの入力になります。

- サービス・リクエスターの場合、Web サービス要求は、アプリケーション・プログラムの出力になります。

DFHWS2LS は、操作ごとに、区分データ・セットのメンバーを生成します。このプログラムは、接頭部に 2 桁の数値を付加することによってメンバー名を生成します。

このパラメーターはオプションですが、Web サービス記述に要求の定義が記述されている場合は、指定する必要があります。

RESPMEM=*value*

以下に示す Web サービス応答の高水準言語構造体が格納されている区分データ・セット・メンバーの名前を生成するときに DFHWS2LS が使用する 1 から 6 文字の接頭部を指定します。

- サービス・プロバイダーの場合、Web サービス応答は、アプリケーション・プログラムの出力になります。
- サービス・リクエスターの場合、Web サービス応答は、アプリケーション・プログラムの入力になります。

DFHWS2LS は、操作ごとに、区分データ・セットのメンバーを生成します。このプログラムは、接頭部に 2 桁の数値を付加することによってメンバー名を生成します。

応答がない場合 (つまり、片方向のメッセージの場合) は、このパラメーターを省略します。

STRUCTURE=(*request*,*response*)

C と C++ の場合にのみ、要求構造体と応答構造体の名前を生成する方法を指定します。

生成された要求構造体と応答構造体には、*requestnn* および *responsenn* という名前が付けられます。ここで、*nn* は、操作ごとに構造体を区別するために生成される数値接尾部を表します。

いずれかの名前または両方の名前を省略した場合、構造体の名前は指定した REQMEM パラメーターおよび RESPMEM パラメーターを基に生成された区分データ・セット・メンバー名と同じ名前になります。

URI=*value*

サービス・プロバイダーでは、このパラメーターはクライアントが Web サービスのアクセスに使用することになる相対 URI を指定します。CICS は、DFHWS2LS によって作成された Web サービス・バインディング・ファイルを基に URIMAP リソースを生成するときに指定された値を使用します。このパラメーターは URIMAP 定義が適用される URI のパスの構成エレメントを指定します。

サービス・リクエスターでは、このパラメーターではターゲット Web サービスの URI は**指定されません**。Web サービス記述に指定された URI が使用されます。ただし、 EXEC CICS INVOKE WEBSERVICE コマンドの URI オプションを使用して指定変更することができます。

WSBIND=*value*

Web サービス・バインディング・ファイルの完全修飾 HFS 名です。DFHWS2LS は、このファイルが存在しない場合、ファイルを作成します (ディレクトリー構造は作成しません)。

WSDL=*value*

Web サービス記述が格納されるファイルの完全修飾 HFS 名です。

LOGFILE=*value*

DFHWS2LS がアクティビティー・ログとトレース情報を書き込むファイルの完全修飾 HFS 名です。 DFHWS2LS は、このファイルが存在しない場合、ファイルを作成します (ディレクトリー構造は作成しません)。

通常はこのファイルを使用する必要はありませんが、DFHWS2LS に問題が発生した場合は、このファイルの提出を IBM のサービス組織から依頼される場合があります。

BINDING=*value*

Web サービス記述に複数の <binding> エレメントが格納されている場合は、このパラメーターを使用して、言語構造および Web サービス・バインディング・ファイルを生成するためにどのエレメントを使用するかを指定します。 Web サービス記述の <binding> エレメントに使用される name 属性の値を指定します。

その他の情報

- DFHWS2LS の実行に使用するユーザー ID を OMVS に定義する必要があります。このユーザー ID には、CICS HFS ファイル構造に対する読み取り権限と、LOGFILE パラメーターに指定されたディレクトリーに対する書き込み権限が必要です。
- Java を実行するため、このユーザー ID には十分な大きさのストレージを割り振る必要があります。

例

```
//LS2WS JOB  'accounting information',name,MSGCLASS=A
//  SET QT=''''
//JAVAPROG EXEC DFHLS2WS,
// TMPFILE=&QT.&SYSUID.&QT
//INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
REQMEM=DFH0XCP4
RESPMEM=DFH0XCP4
LANG=COBOL
PGMNAME=DFH0XCMN
URI=exampleApp/inquireSingle
PGMINT=COMMAREA
WSBIND=/u/exampleapp/wsbind/inquireSingle.wsbind
WSDL=/u/exampleapp/wsdl/inquireSingle.wsdl
/*

//WS2LS JOB  'accounting information',name,MSGCLASS=A
//  SET QT=''''
//JAVAPROG EXEC DFHWS2LS,
// TMPFILE=&QT.&SYSUID.&QT
//INPUT.SYSUT1 DD *
PDSLIB=//CICSHLQ.SDFHSAMP
REQMEM=DFH0XCP4
RESPMEM=DFH0XCP4
LANG=COBOL
PGMNAME=DFH0XCMN
URI=exampleApp/inquireSingle
PGMINT=COMMAREA
WSBIND=/u/exampleapp/wsbind/inquireSingle.wsbind
WSDL=/u/exampleapp/wsdl/inquireSingle.wsdl
/*
```

高水準言語と XML のスキーマ・マッピング

Web サービス記述は、XML スキーマを使用して SOAP メッセージ内部での単純データ・タイプまたは複合データ・タイプの使用法を記述します。ユーティリティー・プログラム DFHLS2WS および DFHWS2LS が高水準言語データ構造から Web サービス記述を生成し、Web サービス記述から高水準言語データ構造を生成する場合、これらのユーティリティー・プログラムはこれら 2 つの場合に使用されるデータ・タイプ間のマッピングを生成します。

- プログラム DFHLS2WS は、高水準言語データ・タイプを XML スキーマの <simpleType> エレメントにマップします。
- プログラム DFHWS2LS は、<simpleType> エレメントを高水準言語データ・タイプにマップします。

2 つのマッピングは対称的ではありません。これは、以下のことを意味します。

- DFHLS2WS を使用して言語データ構造を処理し、その結果として生成される Web サービス記述を DFHWS2LS を使用して処理する場合、最終のデータ構造が最初のデータ構造と同じになると期待することはできません。
- DFHWS2LS を使用して Web サービス記述を処理し、その結果として生成される言語データ構造を DFHLS2WS を使用して処理する場合、最終の Web サービス記述の XML スキーマが最初の Web サービス記述と同じになると期待することはできません。
- 場合によっては、DFHWS2LS が DFHLS2WS ではサポートされない言語データ・タイプを生成し、DFHLS2WS が DFHWS2LS ではサポートされないスキーマ・エレメントを生成することがあります。

DFHLS2WS が処理する言語構造は、CICS がサポートする言語コンパイラーで実装されるその言語の規則に従って正しくコーディングする必要があります。

DFHWS2LS は、 WSDL バージョン 1.1 に適合する Web サービス記述をサポートします。ただし、次の制限があります。

- リテラル・エンコードを使用する SOAP バインディングのみがサポートされます。
- DFHWS2LS でサポートされるトランスポート・プロトコルは、http、https、および WebSphere MQ Series のみです。
- データ・タイプ定義を、XML スキーマ定義言語 (XSD) を使用してエンコードする必要があります。スキーマ内部では、以下の制限があります。
  - SOAP メッセージで使用されるデータ・タイプを明示的に宣言する必要があります。DFHWS2LS は、スキーマ内の他のデータ・タイプから導出され、かつ宣言されていない SOAP メッセージのデータ・タイプをサポートしません。
  - DFHWS2LS は、以下のものをサポートしません。

    <any>、<list>、および <union> エレメント

    <sequence>、<all> および <choice> エレメントの maxOccurs および minOccurs 属性

    抽象型 (継承階層内の非端末型を除く)

    anyType および anySimpleType 型

循環参照 (例えば、型 A に型 B が含まれ、更には型 B に型 A が含まれる場合)

DFHWS2LS は、<attribute> エレメントを含む Web サービス記述を処理しますが、このエレメントは無視します。

- Web サービス記述内の一部のキーワードの長さには制限があります。例えば、操作名、バインディング名、およびパート名の長さは、 255 文字に制限されています (場合によっては、操作名の最大長がこれより多少短い場合があります)。
- バインディング・エレメントごとにサポートされるサービス・エレメントは 1 つだけです。

COBOL と XML スキーマのマッピング

ユーティリティー・プログラム DFHLS2WS および DFHWS2LS は、COBOL のデータ構造と、Web サービス記述の一部となる SOAP メッセージの XML スキーマ定義間のマッピングをサポートします。

COBOL から XML スキーマへのマッピング

DFHLS2WS は、次の表に従って COBOL データ記述エレメントをスキーマ・エレメントにマップします。この表にない COBOL のデータ記述エレメントは、DFHLS2WS ではサポートされません。次の制約事項も適用されます。

- レベル番号が 66、77 および 88 のデータ記述項目はサポートされていません。
- データ記述記入項目の次の文節はサポートされていません。

  OCCURS DEPENDING ON

  OCCURS INDEXED BY

  REDEFINES

  RENAMES (これはレベル 66 です)

  DATE FORMAT
- データ記述項目の次の文節は無視されます。

  BLANK WHEN ZERO

  JUSTIFIED

  VALUE
- 次の SIGN 文節はサポートされていません。

  SIGN LEADING

  SIGN TRAILING
- USAGE 文節の次の句はサポートされていません。

  COMPUTATIONAL-1

  COMPUTATIONAL-2

  OBJECT REFERENCE

  POINTER

  FUNCTION-POINTER

  PROCEDURE-POINTER

- DISPLAY および COMPUTATIONAL-5 データ記述項目でサポートされる唯一の PICTURE 文字は 9 と S です。
- DECIMAL データ記述項目でサポートされる PICTURE 文字は、 9、S、および V です。
- コンパイラー・オプションの影響を受ける可能性のある値を取るデータ記述項目はサポートされていません。

COBOL のデータ記述	**スキーマの simpleType**
PIC X(*n*) PIC A(*n*)	<xsd:simpleType> <restriction base="xsd:string"> <maxlength value="*n*"/> <xsd:whiteSpace value="preserve"/> </restriction> </simpleType>
PIC G(*n*) PIC N(*n*)	<xsd:simpleType> <restriction base="xsd:string"> <maxlength value="*m*"/> <xsd:whiteSpace value="preserve"/> </restriction> </simpleType> ここで *m* = 2*n
PIC S9 DISPLAY PIC S99 DISPLAY PIC S999 DISPLAY PIC S9999 DISPLAY	<xsd:simpleType> <xsd:restriction base="xsd:short"> <xsd:minInclusive value="-*n*"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> ここで、*n* は、文字「9」のパターンによって表現できる最大値です。
PIC S9(*z*) DISPLAY ここで 5 ≤ *z* ≤ 9	<xsd:simpleType> <xsd:restriction base="xsd:int"> <xsd:minInclusive value="-*n*"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> ここで、*n* は、文字「9」のパターンによって表現できる最大値です。
PIC S9(*z*) DISPLAY (9 < *z*)	<xsd:simpleType> <xsd:restriction base="xsd:long"> <xsd:minInclusive value="-*n*"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> ここで、*n* は、文字「9」のパターンによって表現できる最大値です。

COBOL のデータ記述	スキーマの simpleType
PIC 9 DISPLAY PIC 99 DISPLAY PIC 999 DISPLAY PIC 9999 DISPLAY	<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> ここで、*n* は、文字「9」のパターンによって表現できる最大値です。
PIC 9(*z*) DISPLAY ここで 5 ≤ *z* ≤ 9	<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> ここで、*n* は、文字「9」のパターンによって表現できる最大値です。
PIC 9(*z*) DISPLAY (9 < *z*)	<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> <xsd:minInclusive value="0"/> <xsd:maxInclusive value="*n*"/> </xsd:restriction> </xsd:simpleType> ここで、*n* は、文字「9」のパターンによって表現できる最大値です。
PIC S9(*n*) COMP-5 PIC S9(*n*) COMP-4 ここで *n* ≤ 4	<xsd:simpleType> <xsd:restriction base="xsd:short"/> </xsd:simpleType>
PIC S9(*n*) COMP-5 PIC S9(*n*) COMP-4 ここで 5 ≤ *n* ≤ 9	<xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType>
PIC S9(*n*) COMP-5 PIC S9(*n*) COMP-4 ここで 9 <*n*	<xsd:simpleType> <xsd:restriction base="xsd:long"/> </xsd:simpleType>
PIC 9(*n*) COMP-5 PIC 9(*n*) COMP-4 ここで *n* ≤ 4	<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"/> </xsd:simpleType>
PIC 9(*n*) COMP-5 PIC 9(*n*) COMP-4 ここで 5 ≤ *n* ≤ 9	<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"/> </xsd:simpleType>
PIC 9(*n*) COMP-5 PIC 9(*n*) COMP-4 ここで 9 <*n*	<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"/> </xsd:simpleType>
PIC S9(*m*)V9(*n*) COMP-3	<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*p*"/> <xsd:fractionDigits value="*n*"/> </xsd:restriction> </xsd:simpleType> ここで *p* = *m* + *n*

COBOL のデータ記述	スキーマの simpleType
PIC 9(*m*)V9(*n*) COMP-3	<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*p*"/> <xsd:fractionDigits value="*n*"/> <xsd:minInclusive value="0"/> </xsd:restriction> </xsd:simpleType> ここで *p = m + n*

XML スキーマから COBOL へのマッピング

DFHWS2LS は、次の表に従って、スキーマ・エレメントを COBOL のデータ記述エレメントにマップします。

CICS Web サービス・アシスタントは、次の規則を使用してスキーマ・エレメント名から COBOL 変数の固有の名前を生成します。

1. スキーマ・エレメント名が 23 文字を超えた場合は、この長さに切り捨てられます。
2. 必要に応じて、名前を固有にするため、2 文字が追加されます。
3. スキーマで変数が変化する基数を持つように指定する場合 (<xsd:minOccurs> および <xsd:maxOccurs> を指定する場合) に使用されるストリング -cont または -num 用に、5 文字が予約されます。

結果として生成される名前の全長は、30 文字以下になります。

スキーマの単純型	COBOL

<xsd:simpleType>
  <xsd:restriction base="xsd:*type*">
  </xsd:restriction>
</xsd:simpleType>
 	
ここで、*type* は以下のいずれかです。
  anyType
  anySimpleType
	サポートされていない

スキーマの単純型	COBOL
<xsd:simpleType> <xsd:restriction base="xsd:*type*"> <length value="*z*"/> </xsd:restriction> </xsd:simpleType> ここで、*type* は以下のいずれかです。 string normalizedString token Name NMTOKEN language NCName ID IDREF ENTITY hexBinary	PIC X(*z*)
<xsd:simpleType> <xsd:restriction base="xsd:*type*"> </xsd:restriction> </xsd:simpleType> ここで、*type* は以下のいずれかです。 duration date datetime time gDay gMonth gYear gMonthDay gYearMonth <xsd:simpleType> <xsd:restriction base="xsd:*type*"> </xsd:restriction> </xsd:simpleType> ここで、*type* は以下のいずれかです。 duration date datetime time gDay gMonth gYear gMonthDay gYearMonth	PIC X(32)

スキーマの単純型	COBOL
<xsd:simpleType> <xsd:restriction base="xsd:*type*"> </xsd:restriction> </xsd:simpleType> ここで、*type* は以下のいずれかです。 byte unsignedByte	PIC X DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType>	PIC S9999 COMP-5 SYNC または PIC S9999 DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"/> </xsd:simpleType>	PIC 9999 COMP-5 SYNC または PIC 9999 DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType>	PIC S9(9) COMP-5 SYNC または PIC S9(9) DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"/> </xsd:simpleType>	PIC 9(9) COMP-5 SYNC または PIC 9(9) DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:long"/> </xsd:simpleType>	PIC S9(18) COMP-5 SYNC または PIC S9(18) DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </xsd:restriction> </xsd:simpleType>	PIC 9(18) COMP-5 SYNC または PIC 9(18) DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*m*"/> <xsd:fractionDigits value="*n*"/> </xsd:restriction> </xsd:simpleType>	PIC 9(*p*)V9(*n*) COMP-3 ここで*p* = *m* − *n*
<xsd:simpleType> <xsd:restriction base="xsd:boolean"/> </xsd:simpleType>	PIC X DISPLAY
<xsd:simpleType> <xsd:restriction base="xsd:*type*float"> </xsd:restriction> </xsd:simpleType> ここで、*type* は以下のいずれかです。 float double	PIC X(32)

表に示したスキーマの型の一部は、<minInclusive> および <maxInclusive> ファセットに指定された値 (値がある場合) に応じて、COMP-5 SYNC または DISPLAY の COBOL 形式にマップします。

- 符号付き型 (short、int、およびlong) の場合、次のように指定するとき DISPLAY が使用されます。

```
<xsd:minInclusive value="-a"/>
<xsd:maxInclusive value="a"/>
```

ここで、*a* は 9 から成るストリングです。

- 符号なし型 (unsignedShort、unsignedInt、および unsignedLong) の場合は、次のように指定するとき DISPLAY が使用されます。

```
<xsd:minInclusive value="0"/>
<xsd:maxInclusive value="a"/>
```

ここで、*a* は 9 から成るストリングです。

この他の値を指定した場合、あるいは値を指定しなかった場合、COMP-5 SYNC が使用されます。

C および C++ と XML スキーマのマッピング

ユーティリティー・プログラム DFHLS2WS および DFHWS2LS は、C および C++ のデータ・タイプと、Web サービス記述の一部となる SOAP メッセージの XML スキーマ定義間のマッピングをサポートします。

C および C++ から XML スキーマへのマッピング

DFHLS2WS は、次の表に従って、C および C++ のデータ・タイプをスキーマ・エレメントにマップします。この表にない C および C++ の型は DFHLS2WS ではサポートされません。次の制約事項も適用されます。

- ヘッダー・ファイルが C または C++ の有効な構文であることが必要です。
- 次の C および C++ のデータ・タイプはサポートされません。

```
decimal
float
double
long double
wchar_t (C++ のみ)
```

- 次のデータ・タイプは、ヘッダー・ファイル内に存在しても無視されます。

ストレージ・クラス指定子:

```
auto
register
static
extern
mutable
```

修飾子

```
const
volatile
_Export (C++ のみ)
_Packed (C のみ)
```

関数指定子

```
inline (C++ のみ)
virtual (C++ のみ)
```

初期値

- ヘッダー・ファイルには、以下のものを指定できません。

共用体

クラス宣言

列挙型データ型

ポインター型変数

定義済みマクロ (名前の先頭と末尾が下線文字 (__) のマクロ)

行連結シーケンス (改行文字の直後にある \ 記号)

プロトタイプ関数宣言子

プリプロセッサー・ディレクティブ

ビット・フィールド

__cdecl (または _cdecl) キーワード (C++ のみ)

- アプリケーション・プログラマーは、int が 4 バイトにマップするように、32 ビットのコンパイラーを使用する必要があります。

C および C++ のデータ・タイプ	スキーマの simpleType
char[*z*]	<xsd:simpleType> <restriction base="xsd:string"> <length value="*z*"/> </restriction> </simpleType>
char	<xsd:simpleType> <restriction base="xsd:byte"> </restriction> </simpleType>
unsigned char	<xsd:simpleType> <restriction base="xsd:unsignedByte"> </restriction> </simpleType>
short	<xsd:simpleType> <restriction base="xsd:short"> </restriction> </simpleType>
unsigned short	<xsd:simpleType> <restriction base="xsd:unsignedShort"> </restriction> </simpleType>
int long	<xsd:simpleType> <restriction base="xsd:int"> </restriction> </simpleType>
unsigned int unsigned long	unsignedInt <xsd:simpleType> <restriction base="xsd:unsignedInt"> </restriction> </simpleType>
long long	<xsd:simpleType> <restriction base="xsd:long"> </restriction> </simpleType>

C および C++ のデータ・タイプ	スキーマの simpleType
unsigned long long	<xsd:simpleType> <restriction base="xsd:unsignedLong"> </restriction> </simpleType>
bool	<xsd:simpleType> <restriction base="xsd:boolean"> </restriction> </simpleType>

XML スキーマから C および C++ へのマッピング

DFHWS2LS は、次の表に従って、スキーマ・エレメントを C および C++ のデータ・タイプにマップします。

CICS Web サービス・アシスタントは、次の規則を使用してスキーマ・エレメント名から C および C++ 変数の固有の名前を生成します。

1. スキーマ・エレメント名が 50 文字を超えた場合は、この長さに切り捨てられます。
2. 必要に応じて、名前を固有にするため、2 文字が追加されます。
3. スキーマで変数が変化する基数を持つように指定する場合 (<xsd:minOccurs> および <xsd:maxOccurs> を指定する場合) に使用されるストリング -cont または -num 用に、5 文字が予約されます。

結果として生成される名前の全長は、57 文字以下になります。

スキーマの simpleType	C および C++
<xsd:simpleType> <xsd:restriction base="xsd:*type*"> </xsd:restriction> </xsd:simpleType> ここで、*type* は以下のいずれかです。 anyType anySimpleType	サポートされていない
string hexBinary <xsd:simpleType> <xsd:restriction base="xsd:*type*"> <length value="*z*"/> </xsd:restriction> </xsd:simpleType> ここで、*type* は以下のいずれかです。 string hexBinary	char[*z*]
<xsd:simpleType> <restriction base="xsd:byte"> </restriction> </simpleType>	char

スキーマの simpleType	C および C++
<xsd:simpleType> <restriction base="xsd:unsignedByte"> </restriction> </simpleType>	unsigned char
<xsd:simpleType> <restriction base="xsd:short"> </restriction> </simpleType>	short
<xsd:simpleType> <restriction base="xsd:unsignedShort"> </restriction> </simpleType>	unsigned short
<xsd:simpleType> <restriction base="xsd:int"> </restriction> </simpleType>	int
<xsd:simpleType> <restriction base="xsd:unsignedInt"> </restriction> </simpleType>	unsigned int
<xsd:simpleType> <restriction base="xsd:long"> </restriction> </simpleType>	int
<xsd:simpleType> <restriction base="xsd:unsignedLong"> </restriction> </simpleType>	unsigned int
<xsd:simpleType> <restriction base="xsd:boolean"> </restriction> </simpleType>	bool (C++ のみ) short (C のみ)
<xsd:simpleType> <restriction base="xsd:float"> </restriction> </simpleType>	float
<xsd:simpleType> <restriction base="xsd:double"> </restriction> </simpleType>	double

例

C++ の例とその後に結果として得られる WSDL

PL/I と XML スキーマのマッピング

ユーティリティー・プログラム DFHLS2WS および DFHWS2LS は、PL/I のデータ構造と、Web サービス記述の一部となる SOAP メッセージの XML スキーマ定義間のマッピングをサポートします。

Enterprise PL/I から XML スキーマへのマッピング

DFHLS2WS は、次の表に従って、Enterprise PL/I のデータ・タイプをスキーマ・エレメントにマップします。この表にない Enterprise PL/I の型は DFHLS2WS ではサポートされません。次の制約事項も適用されます。

- COMPLEX、FLOAT、VARYING、および VARYINGZ 属性を持つデータ項目はサポートされません。
- DECIMAL(*p*,*q*) として指定されたデータ項目は、$p \geq q$ の場合のみサポートされます。
- BINARY(*p*,*q*) として指定されたデータ項目は、$q = 0$ の場合のみサポートされます。
- データ項目に PRECISION 属性を指定しても、この属性は無視されます。
- サポートされる唯一のピクチャー文字は「9」です。
- 文字「9」が 20 個を超えるピクチャー・ストリングはサポートされません。

Enterprise PL/I	スキーマ
FIXED BINARY (*n*) ここで $n \leq 7$	<xsd:simpleType> <xsd:restriction base="xsd:byte"/> </xsd:simpleType>
FIXED BINARY (*n*) ここで $8 \leq n \leq 15$	<xsd:simpleType> <xsd:restriction base="xsd:short"/> </xsd:simpleType>
FIXED BINARY (*n*) ここで $16 \leq n \leq 31$	<xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType>
FIXED BINARY (*n*) ここで $32 \leq n \leq 63$	<xsd:simpleType> <xsd:restriction base="xsd:long"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(*n*) ここで $n \leq 8$	<xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(*n*) ここで $9 \leq n \leq 16$	<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"/> </xsd:simpleType>
UNSIGNED FIXED BINARY(*n*) ここで $17 \leq n \leq 32$	<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"/> </xsd:simpleType>

Enterprise PL/I	スキーマ
UNSIGNED FIXED BINARY(*n*) ここで 33 ≤ *n* ≤ 64	<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"/> </xsd:simpleType>
FIXED DECIMAL(*n*,*m*)	<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*n*"/> <xsd:fractionDigits value="*m*"/> </xsd:restriction> </xsd:simpleType>
BIT(*n*) ここで、*n* は 8 の倍数です。この他の値はサポートされません。	<xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="*m*"/> </xsd:restriction> </xsd:simpleType> ここで *m* = *n*/2
CHARACTER(*n*)	<xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:maxLength value="*n*"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType>
GRAPHIC(*n*) WIDECHAR(*n*)	<xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <xsd:length value="*m*"/> </xsd:restriction> </xsd:simpleType> ここで *m* = 2*n*
PICTURE '*a*' ここで、*a* は 9、99、999、... です。	<xsd:simpleType> <xsd:restriction base="xsd:long"> <xsd:maxInclusive value="*a*"/> <xsd:minInclusive value="-*a*"/> </xsd:restriction> </xsd:simpleType>
ORDINAL	<xsd:simpleType> <xsd:restriction base="xsd:int"/> </xsd:simpleType>

XML スキーマから Enterprise PL/I へのマッピング

DFHWS2LS は、次の表に従って、スキーマ・エレメントを Enterprise PL/I のデータ・タイプにマップします。

CICS Web サービス・アシスタントは、次の規則を使用してスキーマ・エレメント名から Enterprise PL/I 変数の固有の名前を生成します。

1. スキーマ・エレメント名が 24 文字を超えた場合は、この長さに切り捨てられます。
2. 必要に応じて、名前を固有にするため、2 文字が追加されます。
3. スキーマで変数が変化する基数を持つように指定する場合 (<xsd:minOccurs> および <xsd:maxOccurs> を指定する場合) に使用されるストリング -cont または -num 用に、5 文字が予約されます。

結果として生成される名前の全長は、31 文字以下になります。

スキーマ	Enterprise PL/I
<xsd:simpleType> <xsd:restriction base="xsd:*type*"> </xsd:restriction> </xsd:simpleType> ここで、*type* は以下のいずれかです。 anyType anySimpleType float	サポートされていない
<xsd:simpleType> <xsd:restriction base="xsd:type"> <xsd:maxLength value="*z*"/> <xsd:whiteSpace value="preserve"/> </xsd:restriction> </xsd:simpleType> ここで、*type* は以下のいずれかです。 string normalizedString token Name NMTOKEN language NCName ID IDREF ENTITY	CHARACTER(*z*)
<xsd:simpleType> <xsd:restriction base="xsd:*type*"> </xsd:restriction> </xsd:simpleType> ここで、*type* は以下のいずれかです。 duration date datetime time gDay gMonth gYear gMonthDay gYearMonth	CHAR(32)
<xsd:simpleType> <xsd:restriction base="xsd:hexBinary"> <length value="*y*"/> </restriction> </simpleType>	BIT(*z*) ここで $z = 8 \times y$

スキーマ	Enterprise PL/I
<xsd:simpleType> <xsd:restriction base="xsd:byte"> </xsd:restriction> </xsd:simpleType>	SIGNED FIXED BINARY (7)
<xsd:simpleType> <xsd:restriction base="xsd:unsignedByte"> </xsd:restriction> </xsd:simpleType>	SIGNED FIXED BINARY (8)
<xsd:simpleType> <xsd:restriction base="xsd:short"> </xsd:restriction> </xsd:simpleType>	SIGNED FIXED BINARY (15)
<xsd:simpleType> <xsd:restriction base="xsd:unsignedShort"> </xsd:restriction> </xsd:simpleType>	UNSIGNED FIXED BINARY(16)
<xsd:simpleType> <xsd:restriction base="xsd:int"> </xsd:restriction> </xsd:simpleType>	SIGNED FIXED BINARY(31)
<xsd:simpleType> <xsd:restriction base="xsd:unsignedInt"> </xsd:restriction> </xsd:simpleType>	UNSIGNED FIXED BINARY(32)
<xsd:simpleType> <xsd:restriction base="xsd:long"> </restriction> </simpleType>	SIGNED FIXED BINARY(63)
<xsd:simpleType> <xsd:restriction base="xsd:unsignedLong"> </restriction> </simpleType>	UNSIGNED FIXED BINARY(64)
<xsd:simpleType> <xsd:restriction base="xsd:boolean"> </restriction> </simpleType>	BINARY (7)
<xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:totalDigits value="*n*"/> <xsd:fractionDigits value="*m*"/> </xsd:restriction> </xsd:simpleType>	FIXED DECIMAL(*n*,*m*)

変化するエレメントの配列

SOAP メッセージは、エレメント数が変化する配列を持つことができます。極めて変化しやすい配列を効率よく 1 つの高水準言語データ構造にマップすることはできないため、CICS は一続きのコンテナーとしてアプリケーションに渡される結合された一続きのデータ構造を使用します。

エレメント数が変化する配列は、XML スキーマ内でエレメント宣言の minOccurs 属性および maxOccurs 属性を使用して表現されます。 例を次に示します。

```
<xsd:element name="component" minOccurs="0" maxOccurs="1">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

これは、SOAP メッセージ内でオプション、つまりゼロ回または 1 回発生が可能な 8 バイトのストリングを示します。

次の例は、1 回以上発生する必要のある 8 バイトのストリングを示しています。

```
<xsd:element name="component" minOccurs="1" maxOccurs="unbounded">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

一般には、次のようになります。

minOccurs 属性は、エレメントが発生できる最小の回数を指定します。この属性には、値 0 または任意の正の整数を指定できます。

maxOccurs 属性は、エレメントが発生できる最大の回数を指定します。この属性には、minOccurs 属性の値以上の任意の正の整数値を指定することができます。エレメントが発生できる回数に上限がないことを示す値 unbounded を指定することもできます。

これらの属性のデフォルト値は両方とも 1 です。

一般に、エレメント数が変化する SOAP メッセージを 1 つの高水準言語データ構造に効率よくマップすることはできません。したがって、このような場合に対処するため、CICS は一続きのコンテナーとしてアプリケーションに渡される結合された一続きのデータ構造を使用します。これを行う方法を、一連の例を使用して最もよく説明しています。これらの例では、単純な 8 バイト・フィールドの配列を使用しています。ただし、モデルでは複合データ・タイプの配列、および他の配列を含むデータ・タイプの配列をサポートしています。

固定のエレメント数

最初の例では、ちょうど 3 回発生するエレメントを示しています。

```
<xsd:element name="component" minOccurs="3" maxOccurs="3">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

この例では、エレメントが発生する回数があらかじめ分かっているため、次のように単純な COBOL 宣言 (または他の言語のこれに相当するもの) 内の固定長配列として表現することができます。

```
05 component PIC X(8) OCCURS 3 TIMES
```

オプションの 1 つのエレメント

2 番目の例では、発生する場合は 1 回発生するオプションのエレメントを示しています。

```
<xsd:element name="component" minOccurs="0" maxOccurs="1">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

この例では、主データ構造に配列の宣言が含まれていません。 その代わりに、次の 2 つのフィールドの宣言が格納されます。

```
05 #component PIC S9(9) COMP-4
05 component-container PIC X(16)
```

実行時に、最初のフィールド (#component) に、エレメントが SOAP メッセージ内に現れる回数 (この場合はゼロ回または 1 回) が格納され、2 番目のフィールドにコンテナーの名前が格納されます。

2 番目のデータ構造には、次のようなエレメントそのものの宣言が格納されます。

```
01 DFHWS-component
  02 component PIC X(8)
```

したがって、アプリケーション・プログラムでデータ構造を処理するには、#component の値を検査する必要があります。この値がゼロの場合、メッセージ内に component エレメントはありません。この値がゼロでない場合 (つまり、この値が値 1 を持つ場合)、component エレメントは component-container で指定されたコンテナー内にあります。コンテナーの内容は、DFHWS-component データ構造によってマップされます。

変化するエレメント数

3 番目の例では、1 回から 5 回まで発生できる必須エレメントを示しています。

```
<xsd:element name="component" minOccurs="1" maxOccurs="5">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="8"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

データ構造は、オプションの 1 つのエレメントとまったく同じです。主データ構造には、以下のものが格納されます。

```
05 #component PIC S9(9) COMP-4
05 component-container PIC X(16)
```

2 番目のデータ構造には、以下のものが格納されます。

```
01 DFHWS-component
  02 component PIC X(8)
```

主データ構造の処理は、前の例と同様です。エレメントが発生する回数を割り出すには、#component の値を検査する必要があります (この例では、 1 から 5 までの範囲の値が入ります)。エレメント・コンテントは、component-container で指定さ

れたコンテナー内にあります。違いは、この例ではコンテナーがエレメントの配列を含み、各エレメントが DFHWS-component データ構造によってマップされることです。

要約

最後の 2 つのケースは非常に似ています。実際に 2 つのケースはどちらも同じ一般モデルの例です。規則を次のようにまとめることができます。

- 変化する配列を主データ構造で表現できない場合、配列の各エレメントが 2 番目のデータ構造で表現されます。
- 主データ構造には、配列のエレメント数とエレメントの配列を含むコンテナーの名前が格納されます。
- 配列の各エレメントは、配列に関連付けられた 2 次データ構造でマップされます。

第 8 章 サービス・プロバイダーとサービス・リクエスター・アプリケーションの接続

CICS の Web サービス・サポートと対話するには、サービス・アプリケーションとプロバイダー・アプリケーション (またはラッパー・プログラム) をコーディングする必要があります。これを行う方法は、サービス・プロバイダー・アプリケーションを開発するか、サービス・リクエスターを開発するか、および CICS Web サービス・アシスタントを使用してアプリケーションを配置するかどうかによって異なります。

サービス・プロバイダーでのアプリケーションの呼び出し方法

サービス・プロバイダーでのアプリケーション・プログラム (またはラッパー・プログラム) の呼び出し方法は、アプリケーションの配置に Web サービス・アシスタントを使用するかどうかによって異なります。

CICS による、Web サービス・アシスタントを使用して配置したサービス・プロバイダー・プログラムの呼び出し方法

CICS Web サービス・アシスタントを使用して配置したサービス・プロバイダー・アプリケーションが呼び出されると、CICS は COMMAREA またはチャネルを使用して、そのサービス・プロバイダー・アプリケーションにリンクします。

プログラム DFHWS2LS または DFHLS2WS を PGMINT パラメーターを指定して実行したときに使用されるインターフェースの種類を指定します。チャネルを指定する場合は、 CONTID パラメーターでチャネルの名前を指定することができます。

- プログラムが COMMAREA インターフェースで呼び出される場合、COMMAREA には CICS が SOAP 要求から作成した最上位のデータ構造が格納されます。
- プログラムがチャネル・インターフェースで呼び出される場合は、DFHWS2LS または DFHLS2WS の CONTID パラメーターに指定されたプログラムのコンテナーに最上位のデータ構造が渡されます。CONTID パラメーターを指定しなかった場合、データはコンテナー DFHWS-DATA に渡されます。チャネル・インターフェースでは、一続きのコンテナー内にある結合された一続きのデータ構造として表現される、エレメント数が変化する配列をサポートします。これらのコンテナーも存在します。

プログラムは要求の処理を終了すると、同じメカニズムを使用して応答を戻す必要があります。要求が COMMAREA で受信されている場合は、応答を COMMAREA に戻す必要があります。要求がコンテナーで受信されている場合は、応答を同じコンテナーに戻す必要があります。

プログラムが提供する Web サービスが応答を戻す設計になっていない場合、プログラムが応答を戻しても CICS は COMMAREA またはコンテナー内の情報を無視します。

CICS による他のサービス・プロバイダー・プログラムの呼び出し方法

CICS Web サービス・アシスタントを使用して配置したものでないサービス・プロバイダー・アプリケーションが呼び出されると、CICS はチャネルを使用してそのサービス・プロバイダー・アプリケーションにリンクします。

要求を格納する以下のコンテナーが使用できます。

コンテナー名
内容

DFHWS-BODY
パイプラインが CICS 提供の SOAP メッセージ・ハンドラーを含むときのインバウンド SOAP 要求の場合、SOAP 本体の内容。

DFHREQUEST
パイプラインから受信した完全な要求 (SOAP 応答のエンベロープを含む)。

DFHWS-XMLNS
ネーム・スペースの接頭部を要求の XML の内容のためのネーム・スペースにマップする名前と値のペアのリスト。

DFHWS-SOAPACTION
コンテナー DFHWS-BODY 内の SOAP メッセージに関連付けられた SOAPAction ヘッダー。

パイプライン内のメッセージ・ハンドラーで使用可能であった以下のコンテナーも、ユーザーが作成したこの他のコンテナーと同様、プログラムで使用可能です。

DFHFUNCTION
DFHWS-SOAPLEVEL
DFHWS-URI
DFHWS-TRANID
DFHWS-USERID
DFHWS-APPHANDLER
DFH-SERVICEPLIST
DFHHANDLERPLIST
DFHWS-PIPELINE

プログラムは要求の処理を終了すると、次のコンテナーに応答を戻します。

コンテナー名
内容

DFHRESPONSE
パイプラインに渡される完全な応答メッセージ。メッセージに SOAP を使用しない場合、またはプログラム内で完全な SOAP メッセージ (エンベロープを含む) を作成する場合、このコンテナーを使用します。

コンテナー DFHWS-BODY に SOAP 本体を指定する場合、DFHRESPONSE を空にする必要があります。DFHWS-BODY および DFHRESPONSE の両方に内容を指定した場合、CICS は DFHRESPONSE を使用します。

DFHWS-BODY
アウトバウンド SOAP 応答の場合は、SOAP 本体の内容。パイプラインの端末ハンドラーが CICS 提供の SOAP メッセージ・ハンドラーである場合、このコンテナーを指定します。メッセージ・ハンドラーは、本体を含む完全な SOAP メッセージを作成します。

この他のいずれかのコンテナーを使用して、アウトバウンド応答を処理するためにパイプラインが必要とする情報を渡すことができます。

パイプラインの端末ハンドラーが CICS 提供の SOAP メッセージ・ハンドラーであり、かつ Web サービスが応答を戻さない場合は、応答がないことを示すコンテナー DFHNORESPONSE を戻す必要があります。メッセージ・ハンドラーはコンテナーが存在するかどうかのみをチェックするため、コンテナーの内容は重要ではありません。

SOAP メッセージ・ハンドラーを使用しない場合は、コンテナー DFHRESPONSE を戻さないことによって応答がないことを示します。

CICS プログラムからの Web サービスの呼び出し

アプリケーション・プログラム (またはラッパー・プログラム) からの Web サービスの呼び出し方法は、アプリケーションの配置に Web サービス・アシスタントを使用するかどうかによって異なります。

Web サービス・アシスタントを使用して配置したアプリケーションからの Web サービスの呼び出し

Web サービス・アシスタントを使用して配置したサービス・リクエスター・アプリケーションは、EXEC CICS INVOKE WEBSERVICE コマンドを使用して Web サービスを呼び出します。 要求と応答がコンテナー DFHWS-DATA のデータ構造にマップされます。

1. チャネルを作成して、チャネルにコンテナーを取り込みます。 少なくとも、コンテナー DFHWS-DATA が存在していることが必要です。このコンテナーは、CICS が SOAP 要求に変換する最上位のデータ構造を格納します。SOAP 要求にエレメント数が変化する配列が含まれている場合、このような配列は、一続きのコンテナー内の結合された一続きのデータ構造として表現されます。チャネル内にこれらのコンテナーも存在することが必要です。
2. ターゲット Web サービスを呼び出します。 次のコマンドを使用します。

```
EXEC CICS INVOKE WEBSERVICE(webservice)
                 CHANNEL(userchannel)
                 OPERATION(operation)
```

ここで、

webservice は、呼び出す Web サービスを定義する WEBSERVICE リソースの名前です。WEBSERVICE リソースは、Web サービス記述の場所を指定し、CICS が Web サービスと通信するときに使用する Web サービス・バインディング・ファイルの場所を指定します。

userchannel は、 コンテナー DFHWS-DATA およびアプリケーションのデータ構造に関連付けられているその他のコンテナーを持つチャネルです。

operation は、ターゲット Web サービスで呼び出す操作の名前です。

URI(*uri*) を指定することもできます。*uri* は、呼び出す Web サービスの URI です。 このオプションを省略する場合は、 WEBSERVICE リソース定義に関連付けられている Web サービス・バインディング・ファイルにプロバイダーの URI (DFHWS2LS により Web サービス記述から取得される) またはプロバイダーのアプリケーション名 (DFHWS2LS のパラメーターとして指定される) のいずれかが含まれていることが必要です。このオプションを指定した場合、Web サービス・バインディング・ファイルに指定された URI またはプロバイダー・アプリケーション名の代わりにこのオプションが使用されます。 WEBSERVICE リソースに関連付けられた Web サービス・バインディング・ファイルのプロバイダー・アプリケーション名は、Web サービス要求のローカルでの最適化を可能にする目的で使用されます。この最適化を使用すると、EXEC CICS INVOKE WEBSERVICE コマンドが EXEC CICS LINK コマンドに最適化されます。この最適化は、Web サービスからの応答の送信が期待できない場合の EXEC CICS INVOKE WEBSERVICE コマンドの動作に次のような影響があります。

- 最適化が行われない場合、要求メッセージが送信されるとすぐに EXEC CICS INVOKE WEBSERVICE コマンドから制御が戻ります。
- 最適化が行われる場合は、ターゲット・プログラムが終了した場合にのみ EXEC CICS INVOKE WEBSERVICE コマンドから制御が戻ります。

Web サービスからの応答の送信が期待できる場合は、応答が提供可能であるとき、コマンドから制御が戻ります。

3. コマンドが正常に実行された場合は、チャネルから応答コンテナーを取り出します。 少なくとも、コンテナー DFHWS-DATA は存在します。このコンテナーは、 CICS が SOAP 応答から作成した最上位のデータ構造を格納します。応答にエレメント数が変化する配列が含まれている場合、このような配列は、一続きのコンテナー内の結合された一続きのデータ構造として表現されます。チャネル内にこれらのコンテナーも存在します。

その他のアプリケーションからの Web サービスの呼び出し

Web サービス・アシスタントを使用して配置されたものでないサービス・リクエスター・アプリケーションは、プログラム DFHPIRT にリンクして、Web サービスを呼び出します。 CICS が要求の処理に使用する要求メッセージ、応答メッセージ、およびその他の情報は、一連のコンテナーに格納されます。

1. チャネルを作成して、チャネルにコンテナーを取り込みます。 各コンテナーに次の情報を指定します。

コンテナー名
内容

DFHWS-PIPELINE
アウトバウンド要求に使用される PIPELINE リソースの名前。

DFHWS-URI
ターゲット Web サービスの URI。

DFHWS-BODY
アウトバウンド SOAP 要求の場合は、SOAP 本体の内容。パイプラインが CICS 提供の SOAP メッセージ・ハンドラーを含むとき、このコンテナーを指定します。メッセージ・ハンドラーは、本体を含む完全な SOAP メッセージを作成します。

DFHREQUEST
パイプラインに渡される完全な要求メッセージ。メッセージに SOAP を使用しない場合、またはプログラム内で完全な SOAP メッセージ (エンベロープを含む) を作成する場合、このコンテナーを使用します。

コンテナー DFHWS-BODY に SOAP 本体を指定する場合、DFHREQUEST を空にする必要があります。DFHWS-BODY および DFHREQUEST の両方に内容を指定した場合、CICS は DFHREQUEST を使用します。

DFHWS-XMLNS
ネーム・スペースの接頭部を要求の XML の内容のためのネーム・スペースにマップする名前と値のペアのリスト。

DFHWS-SOAPACTION
コンテナー DFHWS-BODY に指定された SOAP メッセージに追加される SOAPAction ヘッダー。

2. プログラム DFHPIRT にリンクします。 次のコマンドを使用します。

```
EXEC CICS LINK PROGRAM(DFHPIRT) CHANNEL(userchannel)
```

ここで、*userchannel* はコンテナーを含むチャネルです。

3. 同じチャネルからの応答を格納するコンテナーを取り出します。 次のコンテナーに完全な応答が格納されます。

コンテナー名
内容

DFHRESPONSE
パイプラインから受信した完全な応答 (SOAP 応答のエンベロープを含む)。

DFHWS-BODY
パイプラインが CICS 提供の SOAP メッセージ・ハンドラーを含む場合は、SOAP 本体の内容。

DFHERROR
パイプラインからのエラー情報。

第 9 章 パイプライン構成ファイル

Web サービス要求を処理するときに使用する、パイプライン構成ファイルと呼ばれるパイプラインの構成は、XML 文書で指定します。このファイルの名前は、PIPELINE 定義の CONFIGFILE 属性で指定します。パイプライン構成ファイルを使用する場合は、適切な XML エディターまたはテキスト・エディターを使用してください。

CICS は、Web サービス要求を処理する場合、1 つ以上のメッセージ・ハンドラーからなるパイプラインを使用して Web サービス要求を処理します。パイプラインの構成はアプリケーションの必要性に依存するため、多数のサービス・プロバイダー・アプリケーションまたはサービス・リクエスター・アプリケーションが存在する CICS 領域には、いくつかの異なるパイプライン構成が必要になります (ただし、異なるアプリケーションの要件が類似する場合、これらのアプリケーションが同じパイプライン構成を共有することが可能です)。

パイプライン構成は、2 種類あります。1 つはサービス・プロバイダー・パイプラインを記述し、もう 1 つはサービス・リクエスター・パイプラインを記述します。それぞれが独自のスキーマによって定義され、異なるルート・エレメントを持っています。

パイプライン	スキーマ	ルート・エレメント
サービス・プロバイダー	Provider.xsd	<provider_pipeline>
サービス・リクエスター	Requester.xsd	<requester_pipeline>

使用される XML エレメントの多くは両方のパイプライン構成に共通ですが、片方にのみ使用されるエレメントもあるため、プロバイダーとリクエスターの両方に同じ構成ファイルを使用することはできません。

制約事項: パイプライン構成ファイルでは、ネーム・スペースで修飾されたエレメント名はサポートされません。

<provider_pipeline> および <requester_pipeline> の隣接したサブエレメントは、次のとおりです。

- 必須の <service> エレメント。これは、すべての要求に対して呼び出されるメッセージ・ハンドラーを指定します。
- オプションの <transport> エレメント。これは、メッセージ・トランスポートに使用されるリソースに基づいて、実行時に選択されるメッセージ・ハンドラーを指定します。
- <provider_pipeline> のみの場合は、<apphandler> エレメント。これは、一部の例ではサービスを提供するターゲット・アプリケーション (またはラッパー・プログラム) の指定に使用されます。
- オプションの <service_parameter_list> エレメント。パイプライン内のメッセージ・ハンドラーで使用可能なパラメーターを指定します。

パイプライン構成ファイルに関連しているのは、PIPELINE リソースです。 この属性には、HFS にあるパイプライン構成ファイルの名前を指定する CONFIGFILE が

あります。 PIPELINE 定義をインストールすると、CICS は、パイプラインを構成するために必要な情報をこのファイルから読み取ります。

CICS は、即時の構成ファイルを作成するための基本として使用できる構成ファイルのサンプルを提供します。サンプルの内容は次のとおりです。

basicsoap11provider.xml
: CICS 提供の SOAP 1.1 ハンドラーを使用するサービス・プロバイダーのパイプライン定義。アプリケーションが CICS Web サービス・アシスタントを使用して配置されているときに使用されます。

パイプライン構成ファイルの例

以下に、単純なパイプライン構成ファイルの例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline
   xmlns="http://www.ibm.com/software/htp/cics/pipeline"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline provider.xsd">
  <service>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

トランスポート関連ハンドラー

1 つのパイプライン構成ファイルに、複数のメッセージ・ハンドラーの集合を指定することができます。CICS は、実行時に、メッセージのトランスポートに使用されるリソースに基づいて、呼び出されるメッセージ・ハンドラーを選択します。

サービス・プロバイダー、およびサービス・リクエスターで、特定のトランスポート (HTTP または MQ) が使用中の場合にのみいくつかのメッセージ・ハンドラーを呼び出すように指定することができます。サービス・プロバイダーでは、メッセージ・ハンドラーの実行を、指定されたリソース (HTTP トランスポートの場合は TCPIPSERVICE、MQ トランスポートの場合は QUEUE) と関連付けることができます。

この機能をどのように使用できるかを示す例として、従業員に対して使用可能にする Web サービスを考えてみます。会社で働く従業員は、保護された社内ネットワーク上で MQ トランスポートを使用してサービスにアクセスします。しかし、自宅やホテルの部屋で作業する従業員はインターネットを介して HTTP トランスポートを使用してサービスにアクセスします。このような状況では、情報の機密性という性質から、HTTP トランスポートを使用するときはメッセージ・ハンドラーを使用して、メッセージの一部を暗号化することが必要になります。

これを可能にするには、パイプライン構成ファイルの次の 2 つの別個の部分にメッセージ・ハンドラーを指定します。

サービス・セクション
: パイプラインを実行するたびに実行されるメッセージ・ハンドラーを指定します。

トランスポート・セクション
実行時に、使用するトランスポート・リソースに基づいて実行が選択されるメッセージ・ハンドラーを指定します。

要確認: 実行時に、メッセージ・ハンドラーがパイプラインの実行を省略することを選択できます。したがって、CICS がパイプライン構成ファイルの内容に基づいて特定のメッセージ・ハンドラーを実行するように選択している場合でも、これより前のメッセージ・ハンドラーによってこの選択を無効にすることができます。

トランスポート・セクションに指定されたメッセージ・ハンドラー (トランスポート関連ハンドラー) は、いくつかのリストにまとめられます。CICS は、実行時に、使用されているトランスポート・リソースに基づいて、これらのうちの 1 つだけのリストから実行するハンドラーを選択します。使用されているトランスポート・リソースに対応するリストが複数ある場合、CICS は最も選択的なリストを使用します。以下に、サービス・プロバイダーとサービス・リクエスターの両方のパイプラインで使用されるリストを示します。

<default_transport_handler_list>
これはトランスポート関連ハンドラーのリストの中で最も選択的でないリストです。このリストに指定されているハンドラーは、以下に示すどのリストも使用されているトランスポート・リソースに対応しない場合に呼び出されます。

<default_http_transport_handler_list>
サービス・リクエスター・パイプラインでは、HTTP トランスポートが使用されているとき、このリストにあるハンドラーが呼び出されます。

サービス・プロバイダー・パイプラインでは、HTTP トランスポートが使用されており、<named_transport_entry> に TCP/IP 接続に TCPIPSERVICE が指定されていないときに、このリストにあるハンドラーが呼び出されます。

<default_mq_transport_handler_list>
サービス・リクエスター・パイプラインでは、WebSphere MQ トランスポートが使用されているとき、このリストにあるハンドラーが呼び出されます。

サービス・プロバイダー・パイプラインでは、WebSphere MQ トランスポートが使用されており、<named_transport_entry> にインバウンド・メッセージを受信するメッセージ・キューが指定されていないときに、このリストにあるハンドラーが呼び出されます。

以下のメッセージ・ハンドラーのリストは、サービス・プロバイダー・パイプライン用の構成ファイルでのみ使用されます。

<named_transport_entry>
<named_transport_entry> は、エレメントのリストだけでなく、リソースの名前とトランスポート・タイプを指定します。

- HTTP トランスポートの場合、リソース名がインバウンド TCP/IP 接続の TCPIPSERVICE の名前と一致したときに、このリストにあるハンドラーが呼び出されます。

- WebSphere MQ トランスポートの場合は、リソース名がインバウンド・メッセージを受信するメッセージ・キューの名前と一致したときに、このリストにあるハンドラーが呼び出されます。

例

以下に、サービス・プロバイダー・パイプライン用のパイプライン構成ファイルの <transport> エレメントの例を示します。

```
<transport>

  <!-- HANDLER1 and HANDLER2 are the default transport handlers -->
  <default_transport_handler_list>
    <handler>HANDLER1</handler>
    <handler>HANDLER2</handler>
  </default_transport_handler_list>

  <!-- HANDLER3 overrides defaults for MQ transport -->
  <default_mq_transport_handler_list>
    <handler>HANDLER3</handler>
  </default_mq_transport_handler_list>
  <!-- HANDLER4 overrides defaults for http transport with TCPIPSERVICE(WS00) -->
  <named_transport_entry transport="http">
    <name>WS00</name>
    <transport_handler_list>
      <handler>HANDLER4</handler>
    </transport_handler_list>
  </named_transport_entry>

</transport>
```

これらの定義の効果は、次のとおりです。

- <default_mq_transport_handler_list> は、 MQ トランスポートを使用するメッセージがハンドラー HANDLER3 によって処理されるように設定します。
- <named_transport_entry> は、 TCPIPSERVICE(WS00) に関連付けられた TCP/IP 接続を使用するメッセージがハンドラー HANDLER4 によって処理されるように設定します。
- <default_transport_handler_list> は、残りのすべてのメッセージ、つまり HTTP トランスポートを使用するが TCPISERVICE(WS00) を使用しないメッセージがハンドラー HANDLER1 と HANDLER2 によって処理されるように設定します。

要確認: トランスポート・セクションに指定されたハンドラーに加えて、パイプライン定義のサービス・セクションに指定されたハンドラーが呼び出されます。

サービス・プロバイダーのパイプライン定義

メッセージ・ハンドラーは、HFS に格納されている XML 文書で定義されます。この文書が格納されているファイルの名前は、PIPELINE 定義の CFGFILE 属性で指定します。

パイプライン構成文書のルート・エレメントは、<provider_pipeline> エレメントです。 この文書の上位構造を 89 ページの図 21 に示します。

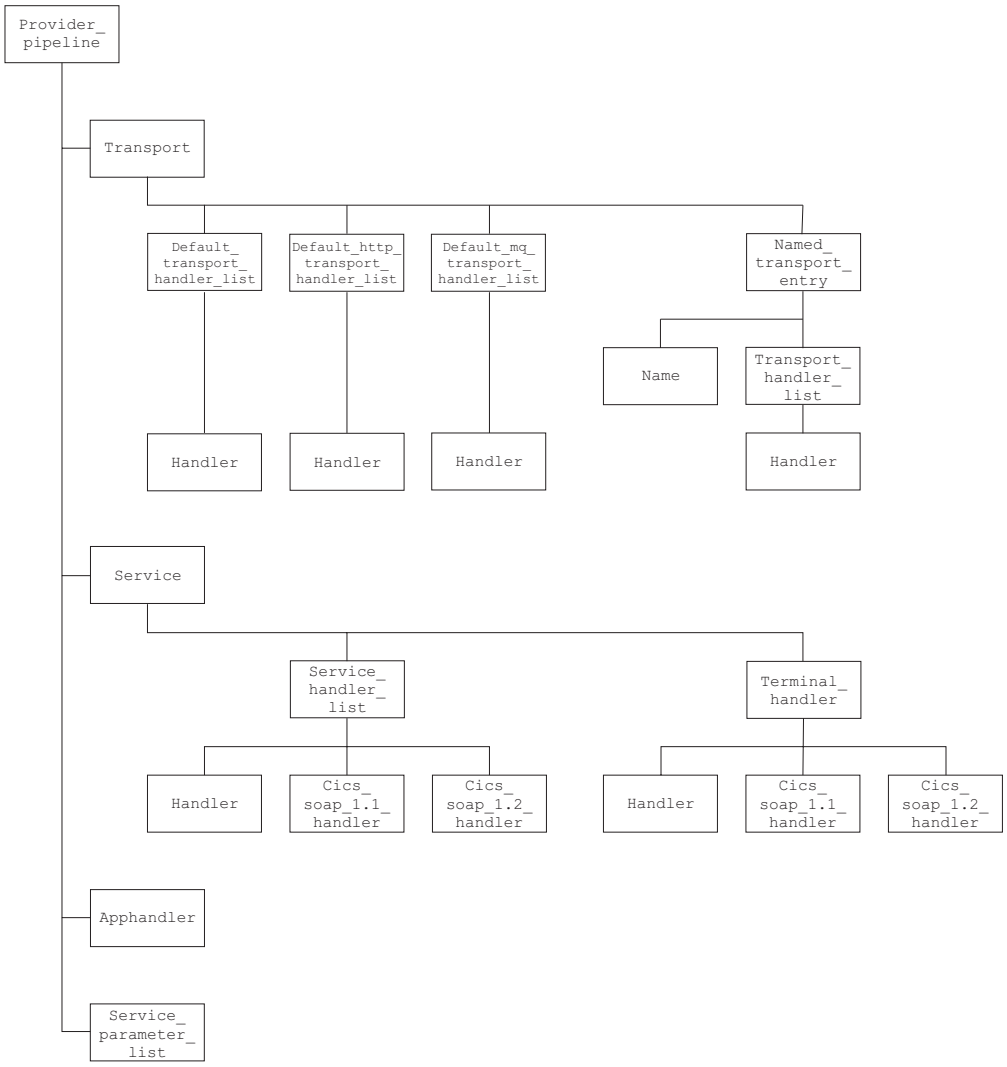


図 21. サービス・プロバイダーのパイプライン定義の構造 :

注: 図を簡略化するために、 <handler>、<cics_soap_1.1_handler>、および <cics_soap_1.2_handler> エレメントの子エレメントは表記していません。

以下のエレメントは、サービス・プロバイダーのパイプライン構成でのみ使用されます。

<named_transport_entry>

<terminal_handler>

その他のエレメントは、サービス・プロバイダーとサービス・リクエスターに共通です。

サービス・リクエスターのパイプライン定義

メッセージ・ハンドラーは、HFS に格納されている XML 文書で定義されます。この文書が格納されているファイルの名前は、PIPELINE 定義の CFGFILE 属性で指定します。

パイプライン構成文書のルート・エレメントは、<provider_pipeline> エレメントです。 この文書の上位構造を図 22 に示します。

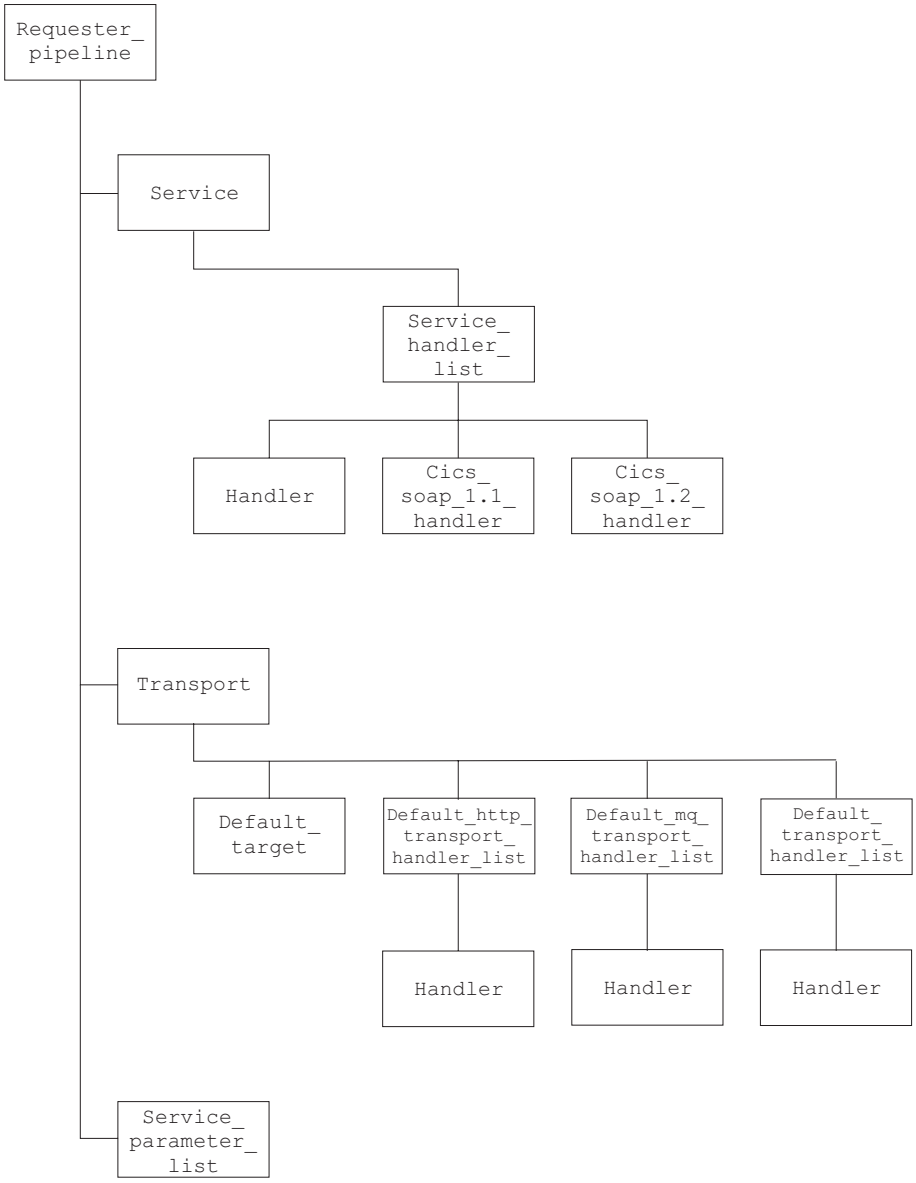


図 22. サービス・リクエスターのパイプライン定義の構造 :

注: 図を簡略化するために、 <handler>、<cics_soap_1.1_handler>、および <cics_soap_1.2_handler> エレメントの子エレメントは表記していません。

サービス・プロバイダーのパイプライン構成で使用されるエレメントは、サービス・リクエスターでも使用されます。

サービス・プロバイダーのみで使用されるエレメント

パイプライン構成ファイルで使用される XML エレメントの一部は、サービス・プロバイダー・パイプラインのみに適用されます。

<named_transport_entry> エレメント

指定のトランスポート・リソースがサービス・プロバイダーに使用されると呼び出されるハンドラーのリストが格納されます。

- MQ トランスポートの場合、指定のリソースは要求を受信するローカルの入力キューになります。
- HTTP トランスポートの場合、リソースは要求を受信したポートを定義する TCPIPSERVICE になります。

使用先

- サービス・プロバイダー

格納元

```
<transport>
```

属性:

名前	説明
transport	指定のリソースと関連したトランスポート機構 **mq** 指定のリソースはキューです。 **http** 指定のリソースは TCPIPSERVICE です。

内容

1. <name> エレメント。リソースの名前が格納されます。
2. オプションの <transport_handler_list> エレメント。 各 <transport_handler_list> には、1 つ以上の <handler> エレメントが格納されます。

   <transport_handler_list> エレメントを記述しなかった場合、指定のトランスポートが使用されると呼び出される唯一のメッセージ・ハンドラーは、<service> エレメントに指定されているメッセージ・ハンドラーになります。

例

```
<named_transport_entry transport="http">
  <name>PORT80</name>
  <transport_handler_list>
    <handler>HANDLER1</handler>
    <handler>HANDLER2</handler>
  </transport_handler_list>
</named_transport_entry>
```

この例では、指定されたメッセージ・ハンドラー (HANDLER1 および HANDLER2) は、PORT80 という名前で TCPIPSERVICE で受信したメッセージに対して呼び出されます。

<provider_pipeline> エレメント

Web サービス・プロバイダーの CICS パイプラインの構成を記述する XML 文書のルート・エレメント。

使用先

- サービス・プロバイダー

内容

1. オプションの <transport> エレメント
2. <service> エレメント
3. オプションの <apphandler> エレメント。パイプラインの端末ノードがデフォルトでリンクする CICS プログラムの名前を指定します。

   <apphandler> は、端末ハンドラーが CICS 提供の SOAP メッセージ・ハンドラーのうちの 1 つであるとき、つまり、<terminal_handler> エレメントに <cics_soap_1.1_handler> エレメントまたは <cics_soap_1.2_handler> エレメントが含まれているときに使用されます。

   メッセージ・ハンドラーは実行時に別のプログラムを指定できるため、ここに記述された名前が必ずしもリンク先のプログラムになるわけではありません。<apphandler> エレメントを記述しなかった場合、いずれかのメッセージ・ハンドラーが DFHWS-APPHANDLER コンテナーを使用して、実行時にプログラムの名前を**指定する必要があります**。

   重要: CICS Web サービス・アシスタントを使用して、サービス・プロバイダーを配置する場合は、<apphandler> エレメント (または DFHWS-APPHANDLER コンテナー) で、ターゲット・アプリケーションやラッパー・プログラムの名前ではなく、DFHPITP を指定する必要があります。この場合、 DFHWS2LS または DFHLS2WS の実行時に PGMNAME パラメーターにプログラムの名前を指定します。
4. オプションの <service_parameter_list> エレメント。コンテナー DFH-SERVICEPLIST のパイプラインに存在するすべてのメッセージ・ハンドラーで使用可能になる XML エレメントが格納されます。

例

```
<provider_pipeline>
  <service>
    ...
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

<terminal_handler> エレメント

サービス・プロバイダー・パイプラインの端末メッセージ・ハンドラーの定義が格納されます。

使用先

- サービス・プロバイダー

格納元

- <service> エレメント

内容

以下のいずれかのエレメント

<handler>

<cics_soap_1.1_handler>

<cics_soap_1.2_handler>

ただし、<cics_soap_1.1_handler> エレメントと <cics_soap_1.2_handler> エレメントは同じパイプラインに定義しないでください。パイプラインが SOAP 1.1 メッセージと SOAP 1.2 メッセージの両方を処理することを期待する場合は、CICS 提供の SOAP 1.2 メッセージ・ハンドラーを使用してください。

要確認: サービス・プロバイダーでは、<terminal_handler> エレメントの場合と同様に、以下のハンドラーを <service_handler_list> エレメントに指定できます。

例

```
<terminal_handler>
  <cics_soap_1.1_handler>
    ...
  </cics_soap_1.1_handler>
<service_handler_list>
```

<transport_handler_list> エレメント

指定のリソースが使用されると呼び出されるメッセージ・ハンドラーのリストが格納されます。

- MQ トランスポートの場合、指定のリソースは、... キューになります。
- HTTP トランスポートの場合、リソースは要求を受信したポートを定義する TCPIPSERVICE になります。

使用先

- サービス・プロバイダー

格納元

- <named_transport_entry> エレメント

内容

- 1 つ以上の <handler> エレメント。

例

```
<transport_handler_list>
  <handler>
    ...
  </handler>
  <handler>
    ...
  </handler>
<transport_handler_list>
```

サービス・リクエスターのみで使用されるエレメント

パイプライン構成ファイルで使用される XML エレメントの一部は、サービス・リクエスター・パイプラインのみに適用されます。

<requester_pipeline> エレメント

サービス・リクエスターのパイプラインの構成を記述する XML 文書のルート・エレメント。

使用先

- サービス・リクエスター

内容

1. オプションの <service> エレメント
2. オプションの <transport> エレメント
3. オプションの <service_parameter_list> エレメント。コンテナー DFH-SERVICEPLIST のメッセージ・ハンドラーで使用可能になる XML エレメントが格納されます。

例

```
<requester_pipeline>
  ...
</requester_pipeline>
```

サービス・プロバイダーおよびサービス・リクエスターで使用されるエレメント

パイプライン構成ファイルで使用される XML エレメントの一部は、サービス・プロバイダーとサービス・リクエスターの両方のパイプラインに適用されます。

<cics_soap_1.1_handler> エレメント

SOAP 1.1 メッセージに対応する CICS 提供のハンドラー・プログラムの属性を定義します。

使用先

- サービス・リクエスター
- サービス・プロバイダー

格納元

<service_handler_list> エレメント

<terminal_handler> エレメント

内容

存在しないか 1 つ以上の <headerprogram> エレメント。 各 <headerprogram> の内容は、以下のとおりです。

1. <programname> エレメント。ヘッダー処理プログラムの名前が格納されています。

2. <namespace> エレメント。ヘッダー処理プログラムが処理する SOAP メッセージのヘッダー・ブロックを調べるときに、次の <localname> エレメントと組み合わせて使用します。 <namespace> エレメントには、ヘッダー・ブロックのネーム・スペースの URI (Universal Resource Identifier) が格納されます。
3. <localname> エレメント。ヘッダー処理プログラムが処理する SOAP メッセージのヘッダー・ブロックを調べるときに、前の <namespace> エレメントと組み合わせて使用します。 <localname> には、ヘッダー・ブロックのエレメント名が格納されます。

   例えば、次のヘッダー・ブロックの場合を考えます。

   ```
   <t:myheaderblock xmlns:t="http://mynamespace" ...> .... </myheaderblock>
   ```

   - ネーム・スペース名は http://mynamespace です。
   - エレメント名は myheaderblock です。

   ヘッダー・プログラムをこのヘッダー・ブロックに一致させるには、<namespace> エレメントおよび <localname> エレメントを次のように記述します。

   ```
   <namespace>http://mynamespace</namespace>
   <localname>myheaderblock</localname>
   ```

   <localname> エレメントに * を記述すると、特定の文字ストリングで始まる名前を持つネーム・スペース内のすべてのヘッダー・ブロックを処理するように指定できます。例を次に示します。

   ```
   <namespace>http://mynamespace</namespace>
   <localname>myhead*</localname>
   ```

   <localname> エレメントに * を指定すると、メッセージ内のヘッダーで複数の <headerprogram> エレメントを一致させることができます。例えば、次のヘッダー・ブロックの場合を考えます。

   ```
   <t:myheaderblock xmlns:t="http://mynamespace" ...> .... </myheaderblock>
   ```

   は、次のフラグメント内のすべての <headerprogram> エレメントと一致します。

   ```
   <cics_soap_1.1_handler>
     <headerprogram>
       <programname>HDRPROG1</programname>
       <namespace>http://mynamespace</namespace>
       <localname>*</localname>
       <mandatory>false</mandatory>
     </headerprogram>
     <headerprogram>
       <programname>HDRPROG2</programname>
       <namespace>http://mynamespace</namespace>
       <localname>myhead*</localname>
       <mandatory>false</mandatory>
     </headerprogram>
     <headerprogram>
       <programname>HDRPROG3</programname>
       <namespace>http://mynamespace</namespace>
       <localname>myheaderblock</localname>
       <mandatory>false</mandatory>
     </headerprogram>
   </cics_soap_1.1_handler>
   ```

この場合、実行されるヘッダー・プログラムは、<headerprogram> エレメントで指定されているヘッダー・プログラムの中で、ヘッダー・ブロックのエレメント名が最も正確に指定されているものです。この例では、 HDRPROG3 です。

SOAP メッセージに複数のヘッダーが含まれる場合は、一致するヘッダーがあるたびにヘッダー処理プログラムが 1 回呼び出されますが、ヘッダーの処理順序は定義されていません。

<namespace> と <localname> は同じであるが、異なるヘッダー・プログラムを指定する <headerprogram> エレメントを 2 つ以上記述した場合、*1* つのヘッダー・プログラムだけが実行されますが、どのプログラムが実行されるかは定義されていません。

4. <mandatory> エレメント。XML ブール値 (true、false、1、または 0) が格納されています。

   true

   1 SOAP メッセージに <uri> エレメントと一致するヘッダーが存在しない場合でも、ヘッダー処理プログラムが 1 回以上呼び出されます。

   - 一致するヘッダーがない場合、ヘッダー処理プログラムは 1 回呼び出されます。
   - いずれかのヘッダーが一致すると、ヘッダー処理プログラムは、一致したヘッダーごとに 1 回呼び出されます。

   CICS が作成する SOAP メッセージには最初はヘッダーはありません。メッセージにヘッダーを追加したい場合は、<mandatory>true</mandatory> または <mandatory>1</mandatory> を指定することにより、少なくとも 1 つのヘッダー処理プログラムが呼び出されるようにする必要があります。

   false

   0 SOAP メッセージに <uri> エレメントと一致するヘッダーが 1 つ以上存在する場合にのみ、ヘッダー処理プログラムが呼び出されます。

   - 一致するヘッダーがない場合、ヘッダー処理プログラムは呼び出されません。
   - いずれかのヘッダーが一致すると、ヘッダー処理プログラムは、一致したヘッダーごとに 1 回呼び出されます。

例

```
<cics_soap_1.1_handler>
  <headerprogram>
    <programname> ... </programname>
    <namespace>...</namespace>
    <localname>...</localname>
    <mandatory>true</mandatory>
  </headerprogram>
</cics_soap_1.1_handler>
```

<cics_soap_1.2_handler> エレメント

CICS 提供の SOAP 1.2 メッセージ・ハンドラー・プログラムの属性を定義します。

使用先

- サービス・リクエスター
- サービス・プロバイダー

格納元

<service_handler_list> エレメント

<terminal_handler> エレメント

内容

存在しないか 1 つ以上の <headerprogram> エレメント。 各 <headerprogram> の内容は、以下のとおりです。

1. <programname> エレメント。ヘッダー処理プログラムの名前が格納されています。
2. <namespace> エレメント。ヘッダー処理プログラムが処理する SOAP メッセージのヘッダー・ブロックを調べるときに、次の <localname> エレメントと組み合わせて使用します。 <namespace> エレメントには、ヘッダー・ブロックのネーム・スペースの URI (Universal Resource Identifier) が格納されます。
3. <localname> エレメント。ヘッダー処理プログラムが処理する SOAP メッセージのヘッダー・ブロックを調べるときに、前の <namespace> エレメントと組み合わせて使用します。 <localname> には、ヘッダー・ブロックのエレメント名が格納されます。

   例えば、次のヘッダー・ブロックの場合を考えます。

   ```
   <t:myheaderblock xmlns:t="http://mynamespace" ...> .... </myheaderblock>
   ```

   - ネーム・スペース名は http://mynamespace です。
   - エレメント名は myheaderblock です。

   ヘッダー・プログラムをこのヘッダー・ブロックに一致させるには、<namespace> エレメントおよび <localname> エレメントを次のように記述します。

   ```
   <namespace>http://mynamespace</namespace>
   <localname>myheaderblock</localname>
   ```

   <localname> エレメントに * を記述すると、特定の文字ストリングで始まる名前を持つネーム・スペース内のすべてのヘッダー・ブロックを処理するように指定できます。例を次に示します。

   ```
   <namespace>http://mynamespace</namespace>
   <localname>myhead*</localname>
   ```

   <localname> エレメントに * を指定すると、メッセージ内のヘッダーで複数の <headerprogram> エレメントを一致させることができます。例えば、次のヘッダー・ブロックの場合を考えます。

   ```
   <t:myheaderblock xmlns:t="http://mynamespace" ...> .... </myheaderblock>
   ```

   は、次のフラグメント内のすべての <headerprogram> エレメントと一致します。

```
<cics_soap_1.1_handler>
  <headerprogram>
    <programname>HDRPROG1</programname>
    <namespace>http://mynamespace</namespace>
    <localname>*</localname>
    <mandatory>false</mandatory>
  </headerprogram>
  <headerprogram>
    <programname>HDRPROG2</programname>
    <namespace>http://mynamespace</namespace>
    <localname>myhead*</localname>
    <mandatory>false</mandatory>
  </headerprogram>
  <headerprogram>
    <programname>HDRPROG3</programname>
    <namespace>http://mynamespace</namespace>
    <localname>myheaderblock</localname>
    <mandatory>false</mandatory>
  </headerprogram>
</cics_soap_1.1_handler>
```

この場合、実行されるヘッダー・プログラムは、<headerprogram> エレメントで指定されているヘッダー・プログラムの中で、ヘッダー・ブロックのエレメント名が最も正確に指定されているものです。この例では、 HDRPROG3 です。

SOAP メッセージに複数のヘッダーが含まれる場合は、一致するヘッダーがあるたびにヘッダー処理プログラムが 1 回呼び出されますが、ヘッダーの処理順序は定義されていません。

<namespace> と <localname> は同じであるが、異なるヘッダー・プログラムを指定する <headerprogram> エレメントを 2 つ以上記述した場合、*1* つのヘッダー・プログラムだけが実行されますが、どのプログラムが実行されるかは定義されていません。

4. <mandatory> エレメント。XML ブール値 (true、false、1、または 0) が格納されています。

   true

   1 SOAP メッセージに <uri> エレメントと一致するヘッダーが存在しない場合でも、ヘッダー処理プログラムが 1 回以上呼び出されます。

   - 一致するヘッダーがない場合、ヘッダー処理プログラムは 1 回呼び出されます。
   - いずれかのヘッダーが一致すると、ヘッダー処理プログラムは、一致したヘッダーごとに 1 回呼び出されます。

   CICS が作成する SOAP メッセージには最初はヘッダーはありません。メッセージにヘッダーを追加したい場合は、<mandatory>true</mandatory> または <mandatory>1</mandatory> を指定することにより、少なくとも 1 つのヘッダー処理プログラムが呼び出されるようにする必要があります。

   false

   0 SOAP メッセージに <uri> エレメントと一致するヘッダーが 1 つ以上存在する場合にのみ、ヘッダー処理プログラムが呼び出されます。

   - 一致するヘッダーがない場合、ヘッダー処理プログラムは呼び出されません。

- いずれかのヘッダーが一致すると、ヘッダー処理プログラムは、一致したヘッダーごとに 1 回呼び出されます。

例

```
<cics_soap_1.2_handler>
  <headerprogram>
    <programname> ... </programname>
    <namespace>...</namespace>
    <localname>...</localname>
    <mandatory>true</mandatory>
  </headerprogram>
</cics_soap_1.2_handler>
```

<default_http_transport_handler_list> エレメント

HTTP トランスポートが使用中の場合、デフォルトで呼び出されるメッセージ・ハンドラーを指定します。

サービス・プロバイダーの場合、このリストで指定されているメッセージ・ハンドラーが呼び出されるのは、<named_transport_entry> エレメントに定義されているハンドラーの特定のリストが存在しなくなった場合のみです。

使用先

- サービス・プロバイダー
- サービス・リクエスター

格納元

- <transport> エレメント

内容

- 1 つ以上の <handler> エレメント。

例

```
<default_http_transport_handler_list>
  <handler>
    ...
  </handler>
  <handler>
    ...
  </handler>
</default_http_transport_handler_list>
```

<default_mq_transport_handler_list> エレメント

MQ トランスポートが使用中の場合、デフォルトで呼び出されるメッセージ・ハンドラーを指定します。

目的

サービス・プロバイダーの場合、このリストで指定されているメッセージ・ハンドラーが呼び出されるのは、<named_transport_entry> エレメントに定義されているハンドラーの特定のリストが存在しなくなった場合のみです。

使用先

- サービス・プロバイダー
- サービス・リクエスター

格納元

- <transport> エレメント

内容

- 1 つ以上の <handler> エレメント。

例

```
<default_mq_transport_handler_list>
  <handler>
    ...
  </handler>
  <handler>
    ...
  </handler>
</default_mq_transport_handler_list>
```

<default_transport_handler_list> エレメント

いずれかのトランスポートが使用中の場合、デフォルトで呼び出されるメッセージ・ハンドラーを指定します。

目的

サービス・プロバイダーでは、以下のいずれかのエレメントに定義された特定のハンドラー・リストがなくなった場合に、このリストに指定されたメッセージ・ハンドラーが呼び出されます。

<default_http_transport_handler_list>

<default_mq_transport_handler_list>

<named_transport_entry>

使用先

- サービス・プロバイダー
- サービス・リクエスター

格納元

- <transport> エレメント

内容

- 1 つ以上の <handler> エレメント。

例

```
<default_transport_handler_list>
  <handler>HANDLER1</handler>
  <handler>HANDLER2</handler>
</default_transport_handler_list>
```

<handler> エレメント

メッセージ・ハンドラー・プログラム (CICS 提供の SOAP メッセージ・ハンドラー・プログラム以外) の属性を定義します。

CICS 提供の SOAP メッセージ・ハンドラー・プログラム。これらは、<cics_soap_1.1_handler> エレメントおよび <cics_soap_1.2_handler> エレメントを使用して定義されます。

使用先

- サービス・プロバイダー
- サービス・リクエスター

格納元

<default_transport_handler_list>

<transport_handler_list>

<service_handler_list>

<terminal_handler>

内容

1. <program> エレメント。ハンドラー・プログラムの名前が格納されます。
2. <handler_parameter_list> エレメント。コンテナー DFH-HANDLERPLIST のメッセージ・ハンドラーで使用可能になるXML エレメントが格納されます。

例

```
<handler>
  <program>MYPROG</program>
  <handler_parameter_list><output print="yes"/></handler_parameter_list>
</handler>
```

この例では、ハンドラー・プログラムは MYPROG です。ハンドラー・パラメーター・リストは、単一の <output> エレメントで構成されます。このパラメーター・リストの内容は、MYPROG に認識されます。

<service> エレメント

すべての要求に対して呼び出されるメッセージ・ハンドラーを指定します。

使用先

- サービス・プロバイダー
- サービス・リクエスター

格納元

<provider_pipeline>

<requester_pipeline>

内容

1. <service_handler_list> エレメント
2. サービス・プロバイダーの場合に限り、<terminal_handler> エレメント

例

```
<service>
  <service_handler_list>
  ...
  </service_handler_list>
  <terminal_handler>
  ...
  <terminal_handler>
</service>
```

<service_handler_list>

すべての要求に対して呼び出されるメッセージ・ハンドラーをのリストを指定します。

目的

ハンドラー・エレメントのリストを格納します。

使用先

- サービス・プロバイダー
- サービス・リクエスター

格納元

- <service> エレメント

内容

以下のエレメントのうち 1 つ以上のエレメント

<handler>

<cics_soap_1.1_handler>

<cics_soap_1.2_handler>

ただし、<cics_soap_1.1_handler> エレメントと <cics_soap_1.2_handler> エレメントは同じパイプラインに定義しないでください。パイプラインが SOAP 1.1 メッセージと SOAP 1.2 メッセージの両方を処理することを期待する場合は、CICS 提供の SOAP 1.2 メッセージ・ハンドラーを使用してください。

要確認: サービス・プロバイダーでは、<service_handler_list> エレメントの場合と同様に、以下のハンドラーを <terminal_handler> エレメントに指定できます。

例

```
<service_handler_list>
  <handler>
    ...
  </handler>
  <cics_soap_1.1_handler>
    ...
  </cics_soap_1.1_handler>
<service_handler_list>
```

<transport> エレメント

特定のトランスポートが使用中の場合にのみ呼び出されるハンドラーを指定します。

使用先

- サービス・プロバイダー
- サービス・リクエスター

格納元

<provider_pipeline>

<requester_pipeline>

内容

サービス・プロバイダーの場合

1. オプションの <default_transport_handler_list> エレメント
2. オプションの <default_http_transport_handler_list> エレメント
3. オプションの <default_mq_transport_handler_list> エレメント
4. 存在しないか 1 つ以上の <named_transport_entry> エレメント

サービス・リクエスターの場合

1. オプションの <default_target> エレメント。<default_target> には、サービス・リクエスター・アプリケーションによって URI が提供されない場合、ターゲット Web サービスの場所を特定するために CICS が使用する URI を指定します。多くの場合、ターゲットの URI はサービス・リクエスター・アプリケーションによって*提供される* ため、<default_target> に指定しても無視されます。例えば、 CICS Web サービス・アシスタントを使用して配置されるアプリケーションは、通常は Web サービス記述から URL を取得します。
2. オプションの <default_http_transport_handler_list> エレメント
3. オプションの <default_mq_transport_handler_list> エレメント
4. オプションの <default_transport_handler_list> エレメント

例

```
<transport>
  <default_transport_handler_list>
  ...
  </default_transport_handler_list>
</transport>
```

第 10 章 メッセージ・ハンドラー

メッセージ・ハンドラーとは、入力時に Web サービスの要求を処理し、出力時に応答を処理するために使用する CICS プログラムのことです。 メッセージ・ハンドラーは、メッセージ・ハンドラー同士の対話やシステムとの対話のために、チャネルおよびコンテナーを使用します。

メッセージ・ハンドラー・インターフェースを使用すると、メッセージ・ハンドラー・プログラム内部で以下のタスクを実行できます。

- XML 要求または応答の内容を、内容を変更せずに調べる
- XML 要求または応答の内容を変更する
- 端末以外のメッセージ・ハンドラーの場合は、XML 要求または応答をパイプライン内の次のメッセージ・ハンドラーに渡す
- 端末のメッセージ・ハンドラーの場合は、アプリケーション・プログラムを呼び出して、応答を生成する
- パイプラインの要求段階では、要求を吸収し、応答を生成することによって応答段階への移行を強制する
- ハンドル・エラー

ヒント: CICS 提供の SOAP 1.1 ハンドラーおよび SOAP 1.2 ハンドラーを使用して SOAP メッセージを処理することをお勧めします。これらのハンドラーを使用すると、SOAP メッセージ (SOAP ヘッダーおよび SOAP 本体) の主要なエレメントを直接処理できます。

メッセージ・ハンドラーとして使用されるすべてのプログラムは、同じインターフェースによって呼び出されます。これらのプログラムは、コンテナーの数を保持するチャネルによって呼び出されます。コンテナーは次のように分類できます。

制御コンテナー
これらのコンテナーは、パイプラインの運用に不可欠です。ノードは制御コンテナーを使用してノードの処理順序を変更できます。

コンテキスト・コンテナー
状況によっては、メッセージ・ハンドラー・プログラムが呼び出されるコンテキストについての情報がメッセージ・ハンドラー・プログラムに必要です。 CICS は、プログラムに渡される一連のコンテキスト・コンテナーにこの情報を提供します。

コンテキスト・コンテナーの一部には、メッセージ・ハンドラーで変更できる情報が格納されます。例えば、サービス・プロバイダー・パイプラインで、ターゲット・アプリケーション・プログラムのユーザー ID やトランザクション ID を変更するには、該当するコンテキスト・コンテナーの内容を変更します。

ユーザー・コンテナー
ここには、あるメッセージ・ハンドラーが別のメッセージ・ハンドラーに渡す必要がある情報が格納されます。ユーザー・コンテナーの使用は、全面的にメッセージ・ハンドラーに委ねられます。

制約事項: ユーザー・コンテナーでは、DFH で始まる名前を使用しないでください。

メッセージ・ハンドラー・プロトコル

パイプラインのメッセージ・ハンドラーは、要求メッセージと応答メッセージを処理します。 メッセージ・ハンドラーの動作は、特定の状況でメッセージ・ハンドラーが可能な動作を記述する一連のプロトコルによって管理されます。

パイプライン内の各非端末メッセージ・ハンドラーは、2 度呼び出されます。

1. 最初は、要求 (サービス・プロバイダー・パイプラインではインバウンド要求、サービス・リクエスターではアウトバウンド要求) を処理するために呼び出されます。
2. 2 度目は、以下の 3 つのいずれかの理由で呼び出されます。
   - 要求 (サービス・プロバイダー・パイプラインではアウトバウンド要求、サービス・リクエスターではインバウンド要求) を処理するため。
   - パイプラインの他の場所でのエラーの後のリカバリーのため。
   - 応答がない場合に必要なその後の処理を実行するため。

サービス・プロバイダー・パイプラインの端末メッセージ・ハンドラーは、要求の処理のため、1 度呼び出されます。

メッセージ・ハンドラーがパイプラインに用意される理由はさまざまであり、各ハンドラーが実行する処理は大幅に異なる場合があります。特に、以下の場合が該当します。

- 一部のメッセージ・ハンドラーはメッセージの内容を変更せず、パイプラインの通常の処理シーケンスも変更しません。
- 一部のメッセージ・ハンドラーは、メッセージの内容は変更しますが、パイプラインの通常の処理シーケンスは変更しません。
- 一部のメッセージ・ハンドラーは、パイプラインの処理シーケンスを変更します。

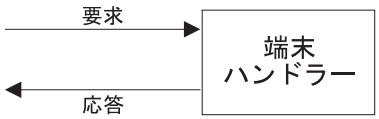
各ハンドラーは、実行できる処理を選択できます。 選択の内容は、以下の条件に依存します。

- ハンドラーの呼び出し元はサービス・プロバイダーとサービス・リクエスターのどちらであるか
- サービス・プロバイダーの場合、ハンドラーは端末ハンドラーかどうか
- ハンドラーの呼び出し対象は要求メッセージと応答メッセージのどちらであるか

端末ハンドラーのプロトコル

通常の要求および応答

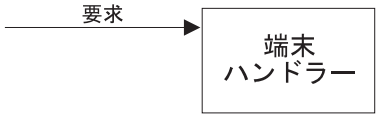
これは、端末ハンドラーの通常のプロトコルです。このハンドラーは、要求メッセージを対象として呼び出され、応答を作成します。



応答を作成するため、標準的な端末ハンドラーはターゲット・アプリケーション・プログラムにリンクしますが、これは必須ではありません。

通常の要求、応答なし

これは、端末ハンドラーのもう 1 つの一般的プロトコルです。



このプロトコルは、通常、ターゲット・アプリケーションが要求に対して応答がないと判断した場合に検出されます (ただし、判断は端末ハンドラーで実行される場合もあります)。

非端末ハンドラーのプロトコル

通常の要求および応答

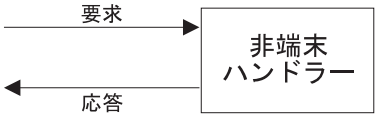
これは、非端末ハンドラーの通常のプロトコルです。このハンドラーは、要求メッセージと応答メッセージの両方を対象として呼び出されます。どちらの場合も、ハンドラーはメッセージを処理し、パイプラインの次のハンドラーにメッセージを渡します。

通常の要求、応答なし

これは、非端末ハンドラーのもう 1 つの一般的プロトコルです。このハンドラーは、要求メッセージを対象として呼び出され、メッセージの処理後、パイプラインの次のハンドラーに渡します。ターゲット・アプリケーション (または別のハンドラー) は、応答がないと判別します。 このハンドラーが 2 度目に呼び出されたときに、処理する応答メッセージはありません。

ハンドラーが応答を作成する

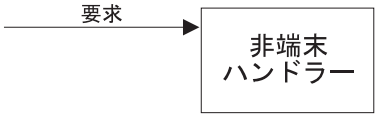
非端末ハンドラーは要求を次のハンドラーに渡さないため、このプロトコルは、通常、異常な状況で使用されます。その代わりに、このプロトコルでは応答が構成され、パイプラインに戻されます。



このプロトコルを使用できるのは、要求が何らかの意味で無効になり、要求がこれ以上処理されなくなるとハンドラーが判別した場合です。この状況では、ハンドラーが 2 回呼び出されることは**ありません**。

ハンドラーが応答を抑止する

非端末ハンドラーは要求を次のハンドラーに渡さないため、このプロトコルは、通常、異常な状況で使用されるプロトコルとは別のプロトコルです。このプロトコルの場合、ハンドラーは要求に対する応答がないと判別します。



このプロトコルを使用できるのは、元の要求に対して応答が期待できなくなった場合と、要求が何らかの意味で無効になり、要求がこれ以上処理されない場合です。

独自のメッセージ・ハンドラーの提供

サービス・リクエスターとサービス・プロバイダーとの間でやり取りされるメッセージに対して特殊な処理を実行する場合で、この要望を満たすメッセージ・ハンドラーが CICS から提供されていない場合は、独自のメッセージ・ハンドラーを用意する必要があります。

大半の状況では、CICS 提供のメッセージ・ハンドラーを使用することにより、必要なすべての処理を実行できます。 例えば、SOAP メッセージを処理するには、CICS 提供の SOAP 1.1 ノードおよび 1.2 ノードを使用できます。ただし、Web サービス要求および応答に対して、独自の特殊な操作を実行することが必要になる場合があります。 このためには、独自のメッセージ・ハンドラーを用意する必要があります。

1. メッセージ・ハンドラー・プログラムを作成します。 メッセージ・ハンドラーとは、チャネル・インターフェースを備えた CICS プログラムのことです。 プログラムは、CICS がサポートしている任意の言語で記述でき、プログラム内部の DPL サブセットには任意の CICS コマンドを使用できます。
2. 通常の方法で CICS システムにプログラムをインストールします。
3. パイプライン構成ファイルでプログラムを定義します。 メッセージ・ハンドラーを定義する場合は、`<handler>` エレメントを使用します。 `<handler>` エレメントの内部には、プログラムの名前を格納した `<program>` エレメントを記述します。

端末以外のメッセージ・ハンドラーでのメッセージの処理

標準的な端末以外のメッセージ・ハンドラーは、メッセージを処理してから、パイプラインに存在する次のメッセージ・ハンドラーに制御を渡します。

端末以外のメッセージ・ハンドラーの場合は、要求または応答を、その内容を変更するかまたは変更せずに処理し、次のメッセージ・ハンドラーに渡すことができます。

注: Web サービスでは、通常、XML を含む SOAP メッセージが使用されますが、メッセージ・ハンドラーは、その他のメッセージ・フォーマットの場合と同様に機能します。

1. コンテナー DFHFUNCTION の内容を使用して、このメッセージ・ハンドラーに渡されたメッセージが要求か応答かを判別します。

DFHFUNCTION	要求または応答	メッセージ・ハンドラーのタイプ	インバウンドまたはアウトバウンド
RECEIVE-REQUEST	要求	端末以外	インバウンド
SEND-RESPONSE	応答	端末以外	アウトバウンド
SEND-REQUEST	要求	端末以外	アウトバウンド
RECEIVE-RESPONSE	応答	端末以外	インバウンド

ヒント:

- DFHFUNCTION に PROCESS-REQUEST が格納されている場合、メッセージ・ハンドラーは端末メッセージ・ハンドラーであり、以下の手順は適用されません。
- DFHFUNCTION に HANDLER-ERROR が格納されている場合、ハンドラーはエラー処理のために呼び出され、以下の手順は適用されません。

2. 適切なコンテナーから要求または応答を取り出します。
   - メッセージが要求である場合、このメッセージはコンテナー DFHREQUEST に格納されてプログラムに渡されます。コンテナー DFHRESPONSE も存在しますが、長さはゼロです。
   - メッセージが応答である場合、このメッセージはコンテナー DFHRESPONSE に格納されてプログラムに渡されます。
3. 必要なメッセージの処理を実行します。 メッセージ・ハンドラーの目的に応じて、次のいずれかを実行できます。
   - 内容を変更せずにメッセージを調べ、パイプラインに存在する次のメッセージ・ハンドラーに渡します。
   - 要求を変更し、パイプラインに存在する次のメッセージ・ハンドラーに渡します。
   - メッセージが要求の場合は、パイプラインに存在する以降のメッセージ・ハンドラーをバイパスして、代わりに応答メッセージを作成できます。

   注: これは、どのメッセージ・ハンドラーが次に呼び出されるかを判別する情報をメッセージ・ハンドラーが戻すコンテナーの内容です。メッセージ・ハンドラーが何も処理を実行しないとエラーになります (つまり、渡されたコンテナーをまったく変更しなかった場合)。

パイプラインに存在する次のメッセージ・ハンドラーへのメッセージの引き渡し

標準的な端末以外のメッセージ・ハンドラーの場合は、要求または応答を、その内容を変更するかまたは変更せずに処理し、次のメッセージ・ハンドラーに渡します。

1. メッセージを (変更するか、未変更のまま) 適切なコンテナーにあるパイプラインに戻します。
   - メッセージが要求で、その内容を変更した場合は、そのメッセージをコンテナー DFHREQUEST に戻します。
   - メッセージが応答で、その内容を変更した場合は、そのメッセージをコンテナー DFHRESPONSE に書き込みます。
   - メッセージを変更しなかった場合、メッセージはすでに適切なコンテナーに格納されています。
2. メッセージが要求の場合は、コンテナー DFHRESPONSE を削除します。 要求を処理するためにメッセージ・ハンドラーが呼び出されると、コンテナー DFHREQUEST および DFHRESPONSE はプログラムに渡されます。 DFHRESPONSE の長さはゼロです。 ただし、DFHREQUEST と DFHRESPONSE の両方を戻すのは誤りです。

メッセージは、パイプラインに存在する次のメッセージ・ハンドラーに渡されます。

パイプラインの応答段階への強制的な移行

要求の処理中には、パイプラインに存在する次のメッセージ・ハンドラーに要求を渡すのではなく、応答を速やかに生成するタイミングがあります。

1. コンテナー DFHREQUEST を削除します。
2. 応答を作成して、コンテナー DFHRESPONSE に書き込みます。

応答は、パイプラインの応答段階で次のメッセージ・ハンドラーに渡されます。

応答の抑止

状況によっては、要求を受けるのみで応答を返信しないようにしたいことがあります。 このためには、コンテナー DFHREQUEST および DFHRESPONSE を削除します。

サービス・リクエスター・パイプラインでの片方向メッセージの処理

サービス・リクエスター・パイプラインがサービス・プロバイダーに要求を送信する場合、通常であれば、要求の送信後に応答があり、応答が到着すると、パイプライン内のメッセージ・ハンドラーが再び呼び出されるという期待があります。 一部の Web サービスでは応答が送信されないため、メッセージ・ハンドラーが応答によって活動化されることなく再び呼び出されるように、特別な対策を講じる必要があります。

このためには、コンテナー DFHNORESPONSE が要求段階のパイプライン処理の最後に存在することを確認します。通常、この処理はアプリケーション・レベルのコードで実行されます。これは、応答が期待されるかどうかの認識はアプリケーションにとどまるためです。

- CICS Web サービス・アシスタントによって配置されたアプリケーションの場合、CICS コードがコンテナーを作成します。
- Web サービス・アシスタントによって配置されなかったアプリケーションは、通常、アプリケーションを呼び出す前にコンテナーを作成します。

メッセージ・ハンドラーでコンテナー DFHNORESPONSE を作成または破棄する場合は、こうすることでサービス・リクエスターとサービス・プロバイダー間のメッセージ・プロトコルを妨害しないことを確認する必要があります。

端末メッセージ・ハンドラーでのメッセージの処理

標準的な端末ノードは、要求を処理し、アプリケーション・プログラムを呼び出して、応答を生成します。

注: Web サービスでは、通常、XML を含む SOAP メッセージが使用されますが、メッセージ・ハンドラーは、その他のメッセージ・フォーマットの場合と同様に機能します。

端末メッセージ・ハンドラーでは、要求を処理できます。また、必要に応じて応答を生成し、パイプラインをたどって戻すこともできます。 標準的な端末ノードでは、要求がアプリケーション・プログラムへの入力として使用され、アプリケーション・プログラムの応答を使用して応答が作成されます。

1. コンテナー DFHFUNCTION の内容を使用して、このノードに渡されたメッセージが要求であることと、このノードは端末ノードとして呼び出されていることを確認します。

DFHFUNCTION	要求または応答	ノードのタイプ	インバウンドまたはアウトバウンド
PROCESS-REQUEST	要求	端末	インバウンド

ヒント:

- DFHFUNCTION にその他の値が格納されている場合、このノードは端末ノードではなく、これらのステップは適用されません。

2. コンテナー DFHREQUEST から要求を取り出します。 コンテナー DFHRESPONSE も存在しますが、長さはゼロです。
3. 必要なメッセージの処理を実行します。 通常、端末ノードはアプリケーション・プログラムを呼び出します。
4. **任意:** 応答を作成して、コンテナー DFHRESPONSE に書き込みます。

応答は、パイプラインの応答段階で次のノードに渡されます。コンテナー DFHRESPONSE の長さがゼロである場合、メッセージ・ハンドラーは応答段階の間に呼び出されず、サービス・リクエスターには応答が送信されません。

エラーの処理

メッセージ・ハンドラーは、パイプラインで発生する可能性のあるエラーを処理する目的で設計する必要があります。

メッセージ・ハンドラー・プログラムでエラーが発生すると、このプログラムはエラー処理のためにもう一度呼び出されます。 パイプラインの応答段階では、エラー処理が必ず発生します。要求段階でエラーが発生すると、要求段階の後続のハンドラーはバイパスされます。

したがって大半の場合は、発生する可能性があるすべてのエラーを処理するためにハンドラー・プログラムを記述する必要があります。

1. コンテナー DFHFUNCTION に、エラー処理のためにメッセージ・ハンドラーが呼び出されたことを示す NODE_ERROR が格納されていることを確認します。

   ヒント:

   - DFHFUNCTION に他の値が格納されている場合は、このメッセージ・ハンドラーがエラー処理のために呼び出されることはなく、これらのステップは適用されません。

2. エラー情報を分析し、適切な応答を作成することにより、メッセージ・ハンドラーがエラー状態から復旧できるかどうかを判断します。 コンテナー DFHERROR には、以下のエラーに関する情報が保持されます。
   - エラーの理由 (異常終了またはハンドラーの障害)
   - エラーの発生先はサービス・リクエスターとサービス・プロバイダーのどちらであるか
   - 異常終了コード
   - エラーが発生したノード
   - コンテナー

   コンテナー DFHRESPONSE も存在しますが、長さはゼロです。
3. リカバリー処理を実行します。
   - メッセージ・ハンドラーが回復できる場合は、応答を作成して、コンテナー DFHRESPONSE に応答を戻します。
   - メッセージ・ハンドラーは回復できるが、応答は必要ない場合は、コンテナー DFHRESPONSE を削除し、代わりにコンテナー DFHNORESPONSE に戻ります。
   - メッセージ・ハンドラーが回復できない場合は、コンテナー DFHRESPONSE を未変更状態 (つまり、長さゼロ) に戻します。

メッセージ・ハンドラー・インターフェース

CICS パイプラインは、多数のコンテナーを内蔵するチャネルを使用して、メッセージ・ハンドラーにリンクします。 コンテナーには、オプションのコンテナーもあれば、すべてのメッセージ・ハンドラーが必要とするコンテナーもあります。また、一部のメッセージ・ハンドラーが使用し、それ以外のハンドラーは使用しないコンテナーもあります。

ハンドラーが呼び出される前に、一部またはすべてのコンテナーには、ハンドラーがその作業を実行するために使用できる情報が取り込まれます。後続の処理は、ハンドラーによって戻されたコンテナーによって決定し、このコンテナーはパイプラインのその後のハンドラーに渡されます。

第 11 章 SOAP メッセージ・ハンドラー

SOAP メッセージ・ハンドラーは、パイプラインに組み込んでSOAP 1.1 メッセージおよび SOAP 1.2 メッセージを処理できる、CICS 提供のメッセージ・ハンドラーです。 SOAP メッセージ・ハンドラーは、サービス・リクエスター・パイプラインまたはサービス・プロバイダー・パイプラインで使用できます。

入力では、SOAP メッセージ・ハンドラーがインバウンド SOAP メッセージを解析し、アプリケーション・プログラムによる使用に備えて SOAP <Body> エレメントを抽出します。出力では、アプリケーションによって提供される <Body> エレメントを使用して、ハンドラー完全な SOAP メッセージを作成します。

メッセージに SOAP ヘッダーを使用すると、SOAP ハンドラーは、インバウンド・メッセージでヘッダーを処理し、アウトバウンド・メッセージでヘッダーを追加できる、ユーザー作成のヘッダー処理プログラムを呼び出すことができます。

SOAP メッセージ・ハンドラーおよびすべてのヘッダー処理プログラムは、<cics_soap_1.1_handler> エレメントおよび <cics_soap_1.2_handler> エレメント、およびそのサブエレメントを使用して、パイプライン構成ファイルで指定されます。

通常、1 つのパイプラインに必要な SOAP ノードは 1 つだけです。ただし、状況によっては、複数の SOAP ノードが必要です。 例えば、複数の SOAP ノードを定義すれば、SOAP ヘッダーを特定の順序で処理できるようになります。

ヘッダー処理プログラム

ヘッダー処理プログラムとは、SOAP ヘッダー・ブロックを処理するために、CICS 提供の SOAP 1.1 ノードおよび SOAP 1.2 ノードにリンクされているユーザー作成 CICS プログラムのことです。

ヘッダー処理プログラムは、CICS がサポートしている任意の言語で記述でき、DPL サブセットに任意の CICS コマンドを使用できます。ヘッダー処理プログラムは、他の CICS プログラムとリンクできます。

ヘッダー処理プログラムには、チャネル・インターフェースがあります。コンテナーには、ヘッダー処理プログラムによって検査または変更できる以下の情報が保持されます。

- プログラムの呼び出し対象となる SOAP ヘッダー・ブロック
- SOAP メッセージの本文

別のコンテナーには、ヘッダー・プログラムの呼び出し元の環境について以下のような情報が保持されます。

- ヘッダー・プログラムの呼び出しに使用されたトランザクション ID
- プログラムの呼び出し元がサービス・プロバイダーと要求側パイプラインのいずれであるか
- 処理対象のメッセージは要求と応答のいずれであるか

SOAP 要求に対するヘッダー処理プログラムの呼び出し方法

パイプライン構成内の <cics_soap_1.1_ handler> エレメントおよび <cics_soap_1.2_ handler> エレメントには、0 または 1 つ以上の <headerprogram> エレメントが格納されており、このエレメントのそれぞれには、以下の子が格納されています。

<program_name>

<namespace>

<localname>

<mandatory>

パイプラインがインバウンド SOAP メッセージ (サービス・プロバイダーでは要求、サービス・リクエスターでは応答) を処理している場合、<program_name> エレメントで指定されたヘッダー・プログラムが呼び出されるかどうかは、以下の内容によって決まります。

- <namespace>、<localname>、<mandatory> エレメントの内容
- SOAP ヘッダー自体のルート・エレメントの特定の属性の値 (SOAP 1.1 の場合は **actor** 属性、SOAP 1.2 の場合は **role** 属性)

以下の規則では、ヘッダー・プログラムが特定の場合に呼び出されるかどうかが判別されます。

パイプライン構成ファイル内の <mandatory> エレメント
エレメントに true (つまり 1) が含まれる場合は、その他の規則によって選択される SOAP メッセージが存在しない場合でも、ヘッダー処理プログラムが 1 回以上呼び出されます。

- 選択されたヘッダー・ブロックがない場合、ヘッダー処理プログラムは 1 回呼び出されます。
- いずれかのヘッダー・ブロックが選択されると、ヘッダー処理プログラムは、選択されたヘッダーごとに 1 回呼び出されます。

SOAP ヘッダー・ブロック内の属性
SOAP 1.1 の場合、ヘッダー・ブロックは、**actor** 属性が存在しない場合、または http://schemas.xmlsoap.org/soap/actor/next の値が存在する場合にのみ、処理のために選択されます。

SOAP 1.2 の場合、ヘッダー・ブロックは、**role** 属性が存在しない場合、または以下のいずれかの値が存在する場合にのみ、処理のために選択されます。

http://www.w3.org/2003/05/soap-envelope/role/next

http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver

パイプライン構成ファイル内の <namespace> エレメントおよび <localname> エレメント ヘッダー・ブロックが処理のために選択されるのは、以下の場合に限られます。

- ヘッダー・ブロックのルート・エレメントの名前が、パイプライン構成ファイルの <localname> エレメントと一致する場合
- かつ ルート・エレメントのネーム・スペースが、パイプライン構成ファイルの <namespace> エレメントと一致する場合

例えば、次のヘッダー・ブロックの場合を考えます。

```
<t:myheaderblock xmlns:t="http://mynamespace" ...> .... </myheaderblock>
```

パイプライン構成ファイルに以下のコードを記述すると、他の規則に従って、処理のためにヘッダー・ブロックが選択されます。

```
<namespace>http://mynamespace</namespace>
<localname>myheaderblock</localname>
```

<localname> に * を記述すると、ネーム・スペース内のすべてのヘッダー・ブロックを処理することを指定できます。したがって、次のコードを記述すると、同じヘッダー・ブロックを選択できます。

```
<namespace>http://mynamespace</namespace>
<localname>*</localname>
```

SOAP メッセージに複数のヘッダーが含まれる場合は、一致するヘッダーがあるたびにヘッダー処理プログラムが 1 回呼び出されますが、ヘッダーの処理順序は定義されていません。

CICS 提供の SOAP メッセージ・ハンドラーは、SOAP メッセージを受信すると、その内部に存在するヘッダー・ブロックに基づいて呼び出されるヘッダー処理プログラムを選択します。従って、ヘッダー処理プログラムは、SOAP メッセージ・ハンドラー内にあるメッセージに追加されるヘッダー・ブロックの結果として呼び出されることはありません。パイプライン内で新規のヘッダー (または任意の変更済みヘッダー) を処理する場合は、パイプライン内に別の SOAP メッセージ・ハンドラーを定義する必要があります。

アウトバウンド・メッセージの場合 (サービス・リクエスターでは要求、サービス・プロバイダーでは応答)、CICS 提供の SOAP メッセージ・ハンドラーはヘッダーを含まない SOAP メッセージを作成します。メッセージに 1 つ以上のヘッダーを追加するためには、ヘッダー・ハンドラー・プログラムを記述してヘッダーを追加する必要があります。このヘッダー・ハンドラーを確実に呼び出すようにするには、パイプライン構成ファイルでヘッダー・ハンドラーを定義し、<mandatory>true</mandatory> を指定する必要があります。

パイプラインの要求段階でヘッダー・ハンドラーが呼び出される場合、応答段階で出されるメッセージに、対応するヘッダーが含まれていない場合でも、応答段階でこのヘッダー・ハンドラーが再度呼び出されます。

ヘッダー処理プログラム・インターフェース

CICS 提供の SOAP 1.1 および SOAP 1.2 メッセージ・ハンドラーは、チャネル DFHHHC-V1 を使用してヘッダー処理プログラムにリンクします。チャネル上で渡されるコンテナーには、いくつかのヘッダー処理プログラム・インターフェースに固有のコンテナーと、パイプライン内のすべてのヘッダー処理プログラムおよびメッセージ・ハンドラーでアクセス可能なコンテキスト・コンテナーおよびユーザー・コンテナーがあります。

コンテナー DFHHEADER は、ヘッダー・プログラミング処理インターフェースに固有のコンテナーです。その他のコンテナーは、パイプライン内の他の場所で使用

することができますが、ヘッダー処理プログラムでは特定の使用法があります。このカテゴリーのコンテナーには、次のものがあります。

DFHWS-XMLNS

DFHWS-BODY

コンテナー DFHHEADER

ヘッダー処理プログラムが呼び出された場合、DFHHEADER には、ヘッダー処理プログラムを駆動する単一のヘッダー・ブロックが格納されています。ヘッダー・プログラムを、パイプライン構成ファイルで `<mandatory>true</mandatory>` または `<mandatory>1</mandatory>` と共に指定すると、SOAP メッセージ内に一致するヘッダー・ブロックがない場合でも、このヘッダー・プログラムが呼び出されます。この場合、コンテナー DFHHEADER の長さはゼロになります。新しい SOAP メッセージにヘッダー・ブロックを追加するためにヘッダー処理プログラムを呼び出す場合がこの例になります。新規に作成された SOAP メッセージには、ヘッダー・ブロックはありません。

CICS が作成する SOAP メッセージには最初はヘッダーはありません。メッセージにヘッダーを追加したい場合は、 `<mandatory>true</mandatory>` または `<mandatory>1</mandatory>` を指定することにより、少なくとも 1 つのヘッダー処理プログラムが呼び出されるようにする必要があります。

ヘッダー・プログラムが戻った場合、コンテナー DFHHEADER には、以下に示すようにヘッダー・ブロックがゼロまたは 1 つ以上格納されている必要があります。このヘッダー・ブロックは、元のヘッダー・ブロックの代わりに CICS が SOAP メッセージに挿入したものです。

- 元のヘッダー・ブロックを未変更のまま戻すことができます。
- ヘッダー・ブロックの内容を変更できます。
- 元のヘッダー・ブロックに 1 つ以上の新規ヘッダー・ブロックを追加できます。
- 元のヘッダー・ブロックを、1 つ以上の異なるブロックと置換できます。
- ヘッダー・ブロックを完全に削除できます。

コンテナー DFHWS-XMLNS

ヘッダー処理プログラムが呼び出された場合、DFHWS-XMLNS には、SOAP エンベロープ内で宣言された XML ネーム・スペースに関する情報が格納されています。 ヘッダー・プログラムは、以下のことを目的としてこの情報を使用できます。

- ヘッダー・ブロック内で検出した修飾名を解決する
- 新規または変更したヘッダー・ブロックで修飾名を構成する

ネーム・スペース情報は、ネーム・スペースを宣言するための標準の XML 表記を使用している、ネーム・スペース宣言のリストで構成されます。DFHWS-XMLNS のネーム・スペース宣言は、スペースで区切られます。例を次に示します。

```
xmlns:na='http://abc.example.org/schema' xmlns:nx='http://xyz.example.org/schema'
```

ネーム・スペース宣言を DFHWS-XMLN の内容に追加すれば、ネーム・スペース宣言を SOAP エンベロープにさらに追加できます。ただし、有効範囲が SOAP ヘッダー・ブロックまたは SOAP 本体であるネーム・スペースは、それぞれヘッダー・

ブロックまたは本体で宣言するのが最適です。ヘッダー処理プログラムでは、コンテナー DFHWS-XMLNS からネーム・スペース宣言を削除しないことをお勧めします。このプログラムでは表示されない XML エレメントがネーム・スペース宣言に依存する場合があるからです。

コンテナー DFHWS-BODY

SOAP エンベロープの本体部分を格納します。 ヘッダー・ハンドラーは、内容を変更する場合があります。

ヘッダー処理プログラムが呼び出された場合、DFHWS-BODY には、SOAP <Body> エレメントが格納されています。

ヘッダー・プログラムが戻った場合、コンテナー DFHWS-BODY には、以下に示すように有効な SOAP <Body> が再度格納されている必要があります。この SOAP 本体は、元の SOAP 本体の代わりに CICS が SOAP メッセージ内に挿入したものです。

- 元の本体を未変更のまま戻すことができます。
- 本体の内容を変更できます。

各 SOAP メッセージには <Body> エレメントが格納されている必要があるため、SOAP 本体を完全に削除することはできません。

コンテキスト・コンテナーおよびユーザー・コンテナー

ここで説明したコンテナーと同様に、インターフェースは、チャネル DFHHHC-V1 上で 制御コンテナー、コンテキスト・コンテナー、およびユーザー・コンテナーを渡します。

SOAP ハンドラー・インターフェース

SOAP ハンドラーには、ユーザー作成プログラムを接続する 2 つのインターフェースがあります。ヘッダー・ハンドラー・インターフェースは、SOAP ハンドラーとヘッダー・ハンドラー・プログラム間で情報を受け渡します。アプリケーション・ハンドラー・インターフェースは、SOAP ハンドラーとアプリケーション・ハンドラー間で情報を受け渡します。

アプリケーション・ハンドラー・インターフェース

アプリケーション・ハンドラー・インターフェースは、SOAP ハンドラーとアプリケーション・ハンドラー・プログラム間のチャネルです。

アプリケーション・ハンドラー・インターフェースが使用するチャネル (DFHAHC) は、以下のコンテナーを受け渡します。

DFHWS-XMLNS

ネーム・スペースの接頭部をネーム・スペースにマップする名前と値のペアのリストを格納します。

- 入力時には、このリストに有効範囲にある SOAP エンベロープ内のネーム・スペースが格納されています。

- 出力時には、このリストにエンベロープ・タグ内にあると見なされるネーム・スペース・データが格納されています。

DFHWS-BODY
SOAP エンベロープの本体部分を格納します。通常、アプリケーション・ハンドラーは内容を変更します。

DFHNORESPONSE
アプリケーション・ハンドラーがコンテナーを戻す場合、SOAP ハンドラーは応答を生成しません。コンテナーの内容は定義されていません。

チャネルは、呼び出し側ハンドラーに渡されたすべてのコンテキスト・コンテナーも同様に渡します。ヘッダー・ハンドラーはチャネルにコンテナーを追加することができます。追加されたコンテナーはユーザー・コンテナーとして、パイプライン内の次の処理ハンドラーに渡されます。

第 12 章 パイプラインで使用されるコンテナー

一般に、パイプラインは、いくつかのメッセージ・ハンドラー・プログラムから構成されます。CICS 提供の SOAP メッセージ・ハンドラーが使用される場合は、いくつかのヘッダー処理プログラムから構成されます。CICS は、コンテナーを使用してこれらのプログラムとの間で情報を受け渡しします。 各プログラムは、パイプライン内の他のプログラムとの通信にもコンテナーを使用します。

CICS のパイプラインは、いくつかのコンテナーから成るチャネルを使用して、メッセージ・ハンドラーや、ヘッダー処理プログラムとリンクします。コンテナーには、オプションのコンテナーもあれば、すべてのメッセージ・ハンドラーが必要とするコンテナーもあります。また、一部のメッセージ・ハンドラーが使用し、それ以外のハンドラーは使用しないコンテナーもあります。

ハンドラーが呼び出される前に、一部またはすべてのコンテナーには、ハンドラーがその作業を実行するために使用できる情報が取り込まれます。後続の処理は、ハンドラーによって戻されたコンテナーによって決定し、このコンテナーはパイプラインのその後のハンドラーに渡されます。

コンテナーは次のように分類できます。

制御コンテナー
これらのコンテナーは、パイプラインの運用に不可欠です。ハンドラーは制御コンテナーを使用してハンドラーの処理順序を変更できます。制御コンテナーの名前は CICS によって定義されます。名前が DFH という文字で始まります。

コンテキスト・コンテナー
ここには、ハンドラーが呼び出された環境に関する情報が格納されます。CICS はこれらのコンテナーに情報を書き込んでから最初のメッセージ・ハンドラーを呼び出しますが、場合によっては、ハンドラーが内容の変更やコンテナーの削除を自由に実行できます。コンテキスト・コンテナーの変更が、ハンドラーの呼び出し順序に直接影響することはありません。制御コンテナーの名前は CICS によって定義されます。名前が DFH という文字で始まります。

ヘッダー処理コンテナー
このコンテナーには、CICS 提供の SOAP 処理ヘッダー・ハンドラーから呼び出されたヘッダー処理プログラムが使用する情報が格納されます。

ユーザー・コンテナー
ここには、あるメッセージ・ハンドラーが別のメッセージ・ハンドラーに渡す必要がある情報が格納されます。ユーザー・コンテナーの使用は、全面的にメッセージ・ハンドラーに委ねられます。これらのコンテナーには独自の名前を選択することができますが、 DFH で始まる名前は使用できません。

制御コンテナー

制御コンテナーは、パイプラインの操作に不可欠です。ハンドラーは制御コンテナーを使用してハンドラーの処理順序を変更できます。

コンテナー DFHFUNCTION

DFHFUNCTION は、パイプライン内のどこでプログラムが呼び出されるかを示す 16 文字のストリングを格納します。

このストリングには、次のいずれかの値が含まれます。右端の文字位置は、ブランク文字で埋め込まれます。

RECEIVE-REQUEST
このハンドラーはサービス・プロバイダー・パイプラインの端末以外のハンドラーで、インバウンド要求メッセージを処理するときに呼び出されます。ハンドラーへの入力時は、このメッセージは制御コンテナー DFHREQUEST に格納されています。

SEND-RESPONSE
このハンドラーはサービス・プロバイダー・パイプラインの端末以外のノードで、アウトバウンド応答メッセージを処理するときに呼び出されます。ハンドラーへの入力時は、このメッセージは制御コンテナー DFHRESPONSE に格納されています。

SEND-REQUEST
このハンドラーは、要求を送信しているパイプラインによって呼び出されます。つまり、アウトバウンド・メッセージを処理しているサービス・リクエスターに存在します。

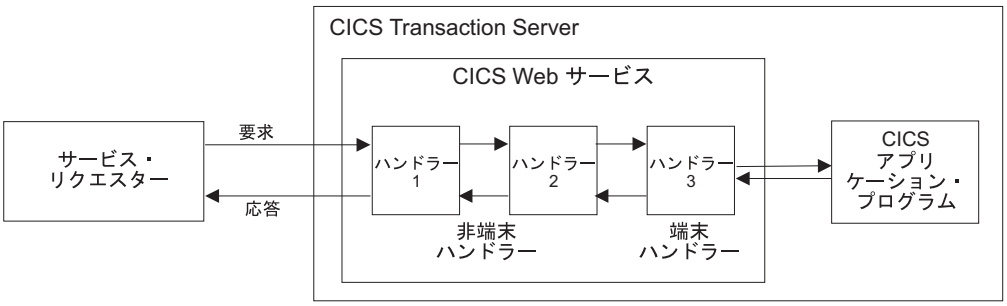
RECEIVE-RESPONSE
このハンドラーは、応答を受信しているパイプラインによって呼び出されます。つまり、インバウンド・メッセージを処理しているサービス・リクエスターに存在します。

PROCESS-REQUEST
このハンドラーは、サービス・プロバイダー・パイプラインの端末ハンドラーとして呼び出されます。

NO-RESPONSE
このハンドラーは、処理の対象となる応答が存在しない場合、要求の処理後に呼び出されます。

HANDLER-ERROR
このハンドラーは、エラーが検出されたために呼び出されます。

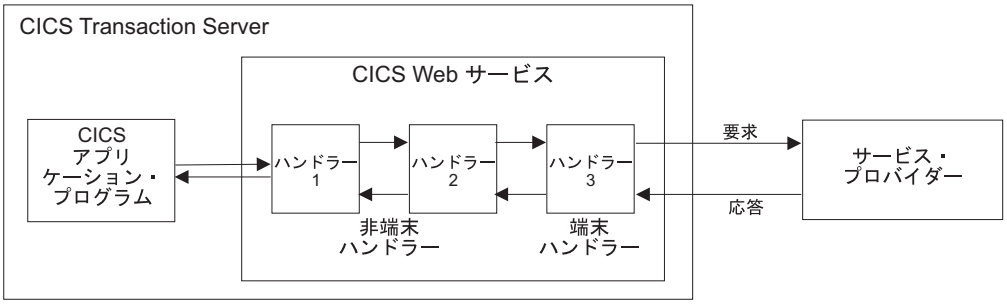
要求を処理し応答を戻すサービス・プロバイダー・パイプラインでは、 発生する DFHFUNCTION の値は RECEIVE-REQUEST、PROCESS-REQUEST、および SEND-RESPONSE です。 123 ページの図 23 には、ハンドラーが呼び出される順序と、各ハンドラーに渡される DFHFUNCTION の値が示されています。



順序	ハンドラー	DFHFUNCTION
1	ハンドラー 1	RECEIVE-REQUEST
2	ハンドラー 2	RECEIVE-REQUEST
3	ハンドラー 3	PROCESS-REQUEST
4	ハンドラー 2	SEND-RESPONSE
5	ハンドラー 1	SEND-RESPONSE

図 23. サービス・プロバイダー・パイプラインでのハンドラーの順序

要求を送信し応答を受信するサービス・リクエスター・パイプラインでは、 発生する DFHFUNCTION の値は SEND-REQUEST および RECIEVE-RESPONSE です。図 24 には、ハンドラーが呼び出される順序と、各ハンドラーに渡される DFHFUNCTION の値が示されています。



順序	ハンドラー	DFHFUNCTION
1	ハンドラー 1	SEND-REQUEST
2	ハンドラー 2	SEND-REQUEST
3	ハンドラー 3	SEND-REQUEST
4	ハンドラー 3	RECEIVE-RESPONSE
5	ハンドラー 2	RECEIVE-RESPONSE
6	ハンドラー 1	RECEIVE-RESPONSE

図 24. サービス・リクエスター・パイプラインでのハンドラーの順序

特定のメッセージ・ハンドラーで検出できる DFHFUNCTION の値は、パイプラインがプロバイダーであるかリクエスターであるか、パイプラインの要求段階であるか応答段階であるか、およびハンドラーが端末ハンドラーであるか端末以外のハンドラーであるかによって異なります。次の表に、それぞれの値が発生する場合をま

とめます。

DFHFUNCTION の値	プロバイダー・パイプラインであるか、リクエスター・パイプラインであるか	パイプラインの段階	端末ハンドラーであるか、端末以外のハンドラーであるか
RECEIVE-REQUEST	プロバイダー	要求段階	端末以外
SEND-RESPONSE	プロバイダー	応答段階	端末以外
SEND-REQUEST	リクエスター	要求段階	端末以外
RECEIVE-RESPONSE	リクエスター	応答段階	端末以外
PROCESS-REQUEST	プロバイダー	要求段階	端末
NO-RESPONSE	両方	応答段階	端末以外
HANDLER-ERROR	両方	両方	両方

コンテナー DFHREQUEST

DFHREQUEST は、パイプラインの要求段階で処理される要求メッセージを格納します。メッセージが CICS 提供の SOAP メッセージ・ハンドラーによって作成され、その後変更されていない場合、DFHREQUEST には完全な SOAP エンベロープとその UTF-8 コード・ページのすべての内容が格納されます。

コンテナー DFHREQUEST は、メッセージ・ハンドラーが呼び出されるときに要求に存在し、コンテナー DFHFUNCTION には RECEIVE-REQUEST または SEND-REQUEST が格納されます。

この状態の通常のプロトコルでは、 DFHREQUEST を同じ内容または変更した内容でパイプラインに戻します。パイプラインの要求段階の処理が正常に続行し、パイプライン内の次のメッセージ・ハンドラー・プログラム (存在する場合) の処理が引き続き実行されます。

これに代わる方法として、メッセージ・ハンドラーでコンテナー DFHREQUEST を削除し、コンテナー DFHRESPONSE に応答を書き込むことができます。これを行った場合、通常の順序の逆に処理が実行され、パイプラインの応答段階の処理が続行されます。

コンテナー DFHRESPONSE

DFHRESPONSE は、パイプラインの応答段階で処理される応答メッセージを格納します。メッセージが CICS 提供の SOAP メッセージ・ハンドラーによって作成され、その後変更されていない場合、DFHRESPONSE には完全な SOAP エンベロープとその UTF-8 コード・ページのすべての内容が格納されます。

コンテナー DFHRESPONSE は、メッセージ・ハンドラーが呼び出されるときに存在し、コンテナー DFHFUNCTION には SEND-RESPONSE または RECEIVE-RESPONSE が格納されます。

この状態の通常のプロトコルでは、 DFHRESPONSE を同じ内容または変更した内容でパイプラインに戻します。パイプラインの処理は正常に続行し、パイプライン内の次のメッセージ・ハンドラー・プログラム (存在する場合) の処理が引き続き実行されます。

コンテナー DFHFUNCTION に RECEIVE-REQUEST、SEND-REQUEST、PROCESS-REQUEST、または HANDLER-ERROR が格納されているとき、コンテナー DFHRESPONSE も存在しますが、長さはゼロです。

コンテナー DFHNORESPONSE

サービス・リクエスター・パイプラインの要求段階では (DFHFUNCTION に SEND-REQUEST が格納されている場合)、コンテナー DFHNORESPONSE が存在することはサービス・プロバイダーが応答を戻すことを期待できないことを示します。

コンテナー DFHNORESPONSE の内容は定義されていません。サービス・プロバイダーが応答を戻すことが見込まれるかどうかを認識する必要のあるメッセージ・ハンドラーは、コンテナーが存在するかどうかのみを判別する必要があります。

この情報は、サービス・プロバイダーと組み合わせて使用するプロトコルに基づいて、最初はサービス・リクエスターのアプリケーションから伝達されます。したがって、メッセージ・ハンドラーに存在するこのコンテナーを削除すること (存在しない場合は、作成すること) は、エンドポイント間のプロトコルを乱す場合があるため、お勧めできません。

サービス・リクエスター・パイプラインの要求段階以外では、このコンテナーの使用は定義されていません。

コンテナー DFHERROR

コンテナー DFHERROR は、パイプラインのエラーに関する情報を他のメッセージ・ハンドラーに伝達します。

あるハンドラーのエラーに関する情報を後続のハンドラーに伝達します。

コンテナーの構造の COBOL 宣言は、次のとおりです。

```
01 PIISNEB.
  02 PIISNEB-MAJOR-VERSION PIC X(1).
  02  PIC X(1).
  02 PIISNEB-ERROR-TYPE PIC X(1).
  02  PIC X(1).
  02 PIISNEB-ABCODE PIC X(4).
  02 PIISNEB-ERROR-CONTAINER1 PIC X(16).
  02 PIISNEB-ERROR-CONTAINER2 PIC X(16).
  02 PIISNEB-ERROR-NODE PIC X(8).
```

コンテキスト・コンテナー

状況によっては、ユーザー作成のメッセージ・ハンドラー・プログラム、およびヘッダー処理プログラムが呼び出されるコンテキストについての情報がこれらのプログラムに必要です。CICS は、プログラムに渡される一連のコンテキスト・コンテナーにこの情報を提供します。

CICS は、各コンテキスト・コンテナーの内容を初期化しますが、場合によっては、メッセージ・ハンドラー・プログラムやヘッダー・ハンドラー・プログラムの内容を変更できます。例えば、端末ハンドラーが CICS 提供の SOAP ハンドラーの 1 つであるサービス・プロバイダー・パイプラインで、ターゲット・アプリケーショ

ン・プログラムのユーザー ID やトランザクション ID を変更するには、該当するコンテキスト・コンテナーの内容を変更します。

コンテナーに格納される情報の一部は、サービス・プロバイダーにのみ、またはサービス・リクエスターにのみ適用されるため、すべてのコンテキスト・コンテナーが両方で利用できるわけではありません。

コンテナー DFHWS-URI

サービス・プロバイダーの場合、このコンテナーは CICS が入力メッセージから抽出するサービスの URI を使用して初期化されます。

コンテナー DFHWS-TRANID

コンテナー DFHWS-TRANID は、パイプラインが稼働中のタスクのトランザクション ID を使用して初期化されます。

端末ハンドラーが CICS 提供の SOAP ハンドラーの 1 つであるサービス・プロバイダー・パイプラインでこのコンテナーの内容を変更した場合 (かつ変更のタイミングが、ターゲット・アプリケーション・プログラムに制御が渡される前である場合)、ターゲット・アプリケーションは、新規のトランザクション ID を使用して新規のタスク内で実行されます。

コンテナー DFHWS-USERID

コンテナー DFHWS-USERID は、パイプラインが稼働中のタスクに関連したユーザー ID を使用して初期化されます。

端末ハンドラーが CICS 提供の SOAP ハンドラーの 1 つであるサービス・プロバイダー・パイプラインでこのコンテナーの内容を変更した場合 (かつ変更のタイミングが、ターゲット・アプリケーション・プログラムに制御が渡される前である場合)、ターゲット・アプリケーションは、新規のユーザー ID と関連した新規のタスクで実行されます。

コンテナー DFHWS-APPHANDLER

サービス・プロバイダー・パイプラインの場合、このコンテナーはパイプライン構成ファイルの `<apphandler>` エレメントの内容によって初期化されます。

パイプラインの端末ノードの場合、CICS 提供の SOAP ハンドラーは、このコンテナーからターゲット・アプリケーション・プログラムの名前を取得します。

メッセージ・ハンドラーまたはヘッダー処理プログラムでこのコンテナーの内容を変更することができます。

CICS は、サービス・リクエスター・パイプラインではこのコンテナーを提供しません。

コンテナー DFHWS-PIPELINE

コンテナー DFHWS-PIPELINE は、プログラムが実行される PIPELINE の名前を格納します。

このコンテナーの内容を変更することはできません。

コンテナー DFHWS-WEBSERVICE

サービス・プロバイダー・パイプラインでのみ、コンテナー DFHWS-WEBSERVICE は一致した WEBSERVICE の RDO 名を (存在する場合) を格納します。

サービス・プロバイダー・パイプラインでは、Web サービスが存在する場合、Web サービスの RDO 名が一致します。 送でない場合、Web サービスは存在しません。

CICS は、サービス・リクエスター・パイプラインではこのコンテナーを提供しません。

コンテナー DFH-SERVICEPLIST

コンテナー DFH-SERVICEPLIST は、パイプライン構成ファイルの `<service_parameter_list>` エレメントの内容を格納します。

パイプライン構成ファイルのサービス・パラメーター・リストを指定していなかった場合、コンテナーは空になります (つまり、コンテナーの長さはゼロです)。

このコンテナーの内容を変更することはできません。

コンテナー DFH-HANDLERPLIST

コンテナー DFH-HANDLERPLIST は CICS 提供の SOAP ハンドラーが呼び出されると必ず初期化され、その内容はパイプライン構成ファイルの適切な `<handler_parameter_list>` エレメントになります。

パイプライン構成ファイルのハンドラー・パラメーター・リストを指定していなかった場合、コンテナーは空になります (つまり、コンテナーの長さはゼロです)。

このコンテナーの内容を変更することはできません。

ユーザー・コンテナー

ここには、あるメッセージ・ハンドラーが別のメッセージ・ハンドラーに渡す必要がある情報が格納されます。 ユーザー・コンテナーの使用は、全面的にメッセージ・ハンドラーに委ねられます。これらのコンテナーには独自の名前を選択することができますが、 `DFH` で始まる名前は使用できません。

第 13 章 Web Services Atomic Transaction のサポート

Web Services Atomic Transaction (または WS-AtomicTransaction) 仕様では、複数のトランザクション処理システムを Web サービス環境で同時に運用できる短期間のトランザクション向けプロトコルが定義されます。 WS-AtomicTransaction を使用するトランザクションには、原子性、一貫性、独立性および耐久性という ACID 特性があります。

WS-AtomicTransaction 仕様は、
http://www.ibm.com/developerworks/library/specification/ws-tx/ で参照できます。

Web サービス・プロバイダーまたは Web サービス・リクエスターとして配置される CICS アプリケーションは、WS-AtomicTransaction をサポートするその他の Web サービス実装環境との分散トランザクションに参加できます。

登録サービス

登録サービスとは、アプリケーションを調整プロトコルに登録するための WS-AtomicTransaction モデルの一部のことです。 分散トランザクションでは、参加しているシステム内で複数の登録サービスが互いに対話し、接続されているアプリケーションがこうしたプロトコルで参加できるようになります。

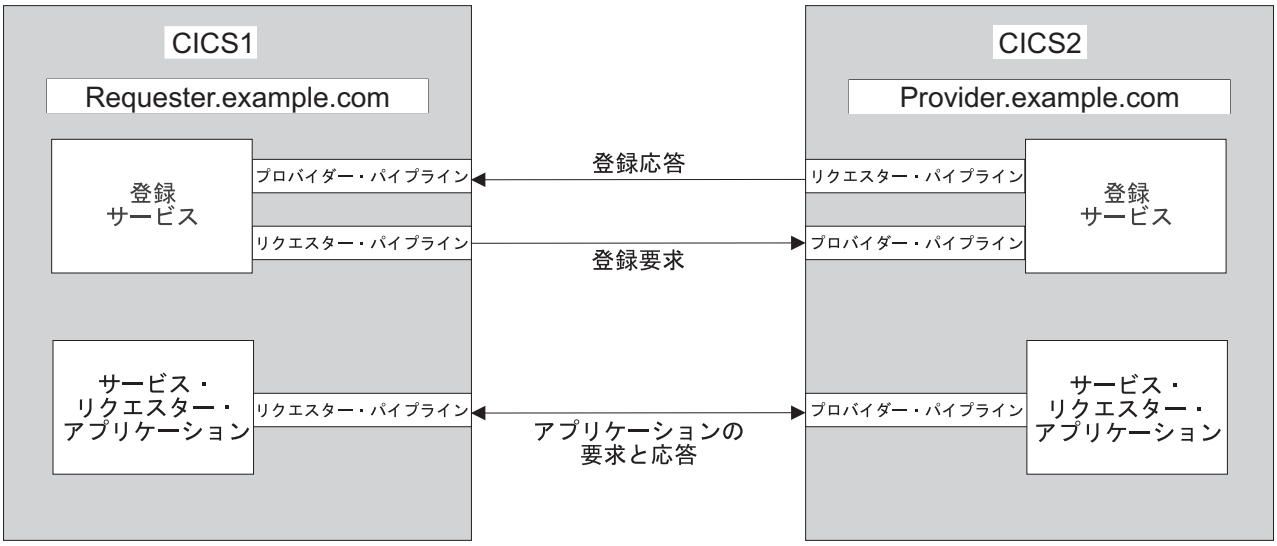


図 25. 登録サービス

図 25 には、2 つの CICS システムである CICS1 および CICS2 を示します。 CICS1 のサービス・リクエスター・アプリケーションが、CICS2 のサービス・プロバイダー・アプリケーションを呼び出します。 2 つの CICS 領域と 2 つのアプリケーションは、2 つのアプリケーションが WS-AtomicTransaction プロトコルを使用して 1 つの分散トランザクションに参加できるように構成されます。サービス・リクエスター・アプリケーションはコーディネーターで、サービス・プロバイダー・アプリケーションは参加プログラムです。

これらのプロトコルの補助として、2 つの CICS 領域の登録サービスがトランザクションの開始時とトランザクションの終了時に対話します。これらの対話中、2 つの領域の登録サービスは、サービス・プロバイダーおよびサービス・リクエスターとして、それぞれ別の時間に動作できます。したがって、各領域では、登録サービスがサービス・プロバイダー・パイプラインおよびサービス・リクエスター・パイプラインを使用します。パイプラインは、PIPELINE および関連リソースと共に CICS に定義されます。

各領域の登録サービスは、エンドポイント・アドレスと関連付けられます。このため、この例では、CICS1 の登録サービスには `requester.example.com` というエンドポイント・アドレスがあり、CICS2 の登録サービスには `provider.example.com` というエンドポイント・アドレスがあります。

CICSplex では、登録サービス・プロバイダー・パイプラインをさまざまな領域に分散できます。このことは、図 26 に示します。

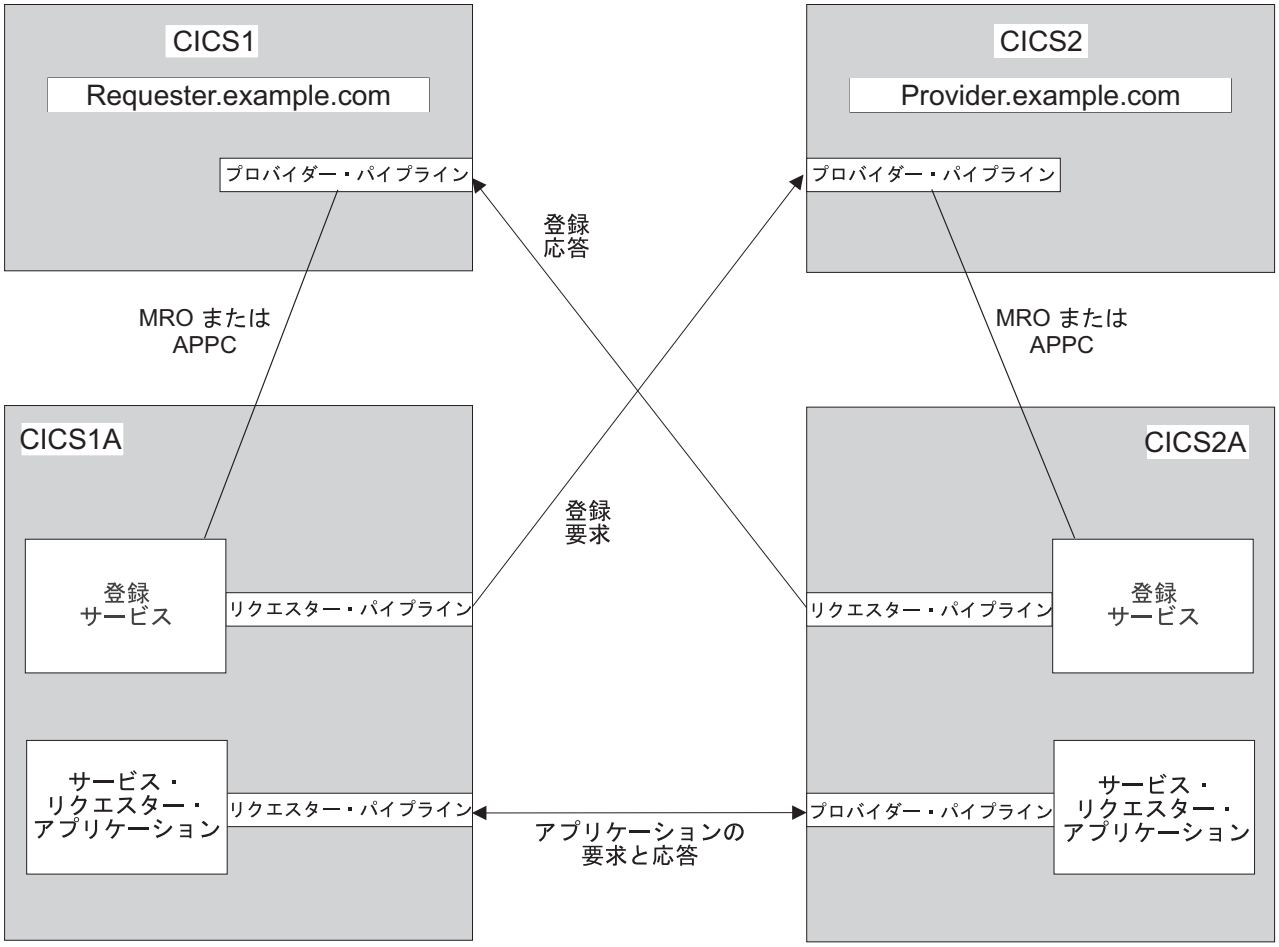


図 26. CICSPlex での登録サービス

この構成では、プロバイダー・パイプラインは MRO または APPC を使用して登録サービスと対話します。 登録サービス・リクエスター・パイプラインは、アプリケーションのリクエスター・パイプラインと同じ領域に残る必要があります。

この構成は、サービス・リクエスター・アプリケーションおよびサービス・プロバイダー・アプリケーションが多数の領域にまたがって分散している場合に便利です。アプリケーションのパイプラインが WS-AtomicTransaction に参加するように構成する場合は、登録サービス・プロバイダー・パイプラインの IPアドレスとポート番号を入力して、登録サービスのエンドポイントに関する情報を提供する必要があります。エンドポイントを 1 つにすると、すべてのパイプラインに同じ情報が格納されるため、構成を単純化できます。例えば、 130 ページの図 26 では、アプリケーションのリクエスター・パイプラインに指定する IP アドレスは、`requester.example.com` になります。

サービス・プロバイダー・アプリケーションにも同じ引数が適用されます。この例では、プロバイダー・アプリケーションのパイプラインに指定する IP アドレスは `requester.example.com` になります。

WS-AtomicTransaction に合わせた CICS の構成

Web サービス・リクエスター・アプリケーションおよび Web サービス・プロバイダー・アプリケーションが WS-AtomicTransaction に参加するには、それに応じて CICS を構成する必要があります。このためには、いくつかの CICSリソースをインストールします。

この作業を実行するには、その前に、WS-AtomicTransaction をサポートするために CICS が提供するパイプライン構成ファイルの場所を確認する必要があります。 デフォルトでは、構成ファイルはディレクトリー `/usr/lpp/cicsts/cicsts31/pipeline/configs`に置かれますが、デフォルトのファイル・パスは CICS のインストール時に変更されている場合があります。

WS-AtomicTransaction 対応の CICS サポートでは、CICS 提供の登録サービス・サービス・プロバイダーおよびサービス・リクエスターが使用されるため、これらの両方にリソースをインストールする必要があります。

重要: アプリケーションがすべてサービス・プロバイダーまたはサービス・リクエスターである場合でも、両方をインストールしなければなりません。

サービス・プロバイダー・アプリケーションおよびサービス・リクエスター・アプリケーションの実行時に呼び出されるヘッダー・ハンドラー・プログラムのプログラム定義もインストールする必要があります。

すべてのリソースは、グループ DFHWSAT 内に CICS によって提供されるため、リソースを変更することはできません。グループ DFHWSAT は、 リスト DFHLIST には*組み込まれない* ため、自動的にはインストールされません。

1. グループ DFHWSAT の内容を他のグループにコピーします。 すべてのリソースは、グループ DFHWSAT 内に CICS によって提供されるため、リソースを変更することはできません。ただし、PIPELINE リソース内の CONFIGFILE 属性の変更は必要になります。
2. CICS 提供の登録サービス・プロバイダー PIPELINE リソースを変更します。この PIPELINE には DFHWSATP という名前が付けられ、この PIPELINE によ

って、パイプライン構成ファイル
/usr/lpp/cicsts/cicsts31/pipeline/configs/registrationservicePROV.xml が
CONFIGFILE 属性に指定されます。

a. システム内のファイルの場所を反映するように CONFIGFILE 属性を変更します。

b. その他の属性は未変更のままにしておきます。

3. PIPELINE リソースをインストールします。 登録サービス・プロバイダー PIPELINE リソースは、サービス・リクエスター・アプリケーションやサービス・プロバイダー・アプリケーションと同じ CICS 領域に置く必要はありませんが、適切な MRO 接続または APPC 接続により、同じ CICS 領域に接続しておく必要があります。
4. 登録サービス・プロバイダーが使用する URIMAP を、PIPELINE と同じ領域に変更しないでインストールします。 この URIMAP には、DFHRSURI という名前が付けられます。
5. CICS 提供の登録サービス・リクエスター PIPELINE リソースを変更します。この PIPELINE には DFHWSATR という名前が付けられ、この PIPELINE によって、パイプライン構成ファイル
/usr/lpp/cicsts/cicsts31/pipeline/configs/registrationserviceREQ.xml が
CONFIGFILE 属性に指定されます。

a. システム内のファイルの場所を反映するように CONFIGFILE 属性を変更します。

b. その他の属性は未変更のままにしておきます。

6. PIPELINE リソースをインストールします。

重要: 登録サービス・リクエスター PIPELINE リソースは、サービス・リクエスター・アプリケーションおよびサービス・プロバイダー・アプリケーションと同じ CICS 領域に置く必要があります。

7. 登録サービス・プロバイダー・パイプラインが使用するプログラムを、PIPELINE リソースと同じ領域にインストールします。 このプログラムとは、DFHWSATX、DFHWSATR、および DFHPIRS です。 2 つの PIPELINE リソースが異なる領域に存在する場合は、これらのプログラムを両方の領域にインストールする必要があります。
8. ヘッダー・ハンドラー・プログラムの PROGRAM リソース定義をインストールします。 このプログラムには、DFHWSATH という名前が付けられます。PROGRAM は、サービス・プロバイダー・アプリケーションとサービス・リクエスター・アプリケーションが稼働する領域にインストールします。

CICS は、これで、サービス・プロバイダー・アプリケーションおよびサービス・リクエスター・アプリケーションが、WS-AtomicTransaction プロトコルを使用して分散トランザクションに参加できるように構成されました。

今度は、参加する各アプリケーションを個別に構成する必要があります。

WS-AtomicTransaction に合わせたサービス・プロバイダーの構成

サービス・プロバイダー・アプリケーションの目的が、WS-AtomicTransaction プロトコルを使用して分散トランザクションに参加することである場合、パイプライン構成ファイルには、<headerprogram> および <service_parameter_list> を指定する必要があります。

サービス・プロバイダー・アプリケーションが WS-AtomicTransaction に参加できるようにするには、サービス・プロバイダー・アプリケーションが SOAP プロトコルを使用してサービス・リクエスターと通信し、かつパイプラインを構成して、CICS 提供の SOAP メッセージ・ハンドラーのいずれかを使用する必要があります。 サービス・プロバイダー・アプリケーションを正しく構成した場合でも、サービス・リクエスター・アプリケーションの参加が設定されていると、サービス・プロバイダー・アプリケーションは、サービス・リクエスターとの WS-AtomicTransaction にのみ参加します。

アプリケーションに固有のパイプライン構成情報に加えて、構成ファイルには、アプリケーションが WS-AtomicTransaction に必ず参加するように、CICS が使用する情報が格納されている必要があります。

CICS では、この情報が格納されているパイプライン構成ファイルの例を、ファイル /usr/lpp/cicsts/apobsf/samples/pipelines/wsatprovider.xml で提供しています。

1. 端末ハンドラーの定義で、<headerprogram> エレメントを <cics_soap_1.1_handler> エレメントまたは <cics_soap_1.2_handler> エレメントの内部にコーディングします。 <program_name>、<namespace>、<localname>、 <mandatory> の各エレメントを、次に示す例のとおり正確にコーディングします。 例を次に示します。

   ```
   <terminal_handler>
     <cics_soap_1.1_handler>
       <headerprogram>
         <program_name>DFHWSATH</program_name>
         <namespace>http://schemas.xmlsoap.org/ws/2004/10/wscoor</namespace>
         <localname>CoordinationContext</localname>
         <mandatory>false</mandatory>
       </headerprogram>
     </cics_soap_1.1_handler>
   </terminal_handler>
   ```

   その他の <headerprogram> エレメントがアプリケーションに必要な場合は、それらも指定できます。

2. <registration_service_endpoint> エレメントを <service_parameter_list> の内部にコーディングします。 <registration_service_endpoint> を、次のようにコーディングします。

   ```
   <registration_service_endpoint>
   address:port/cicswsat/RegistrationService
   </registration_service_endpoint>
   ```

   ここで

   address は、登録サービス・プロバイダー・パイプラインがインストールされている CICS 領域の IP アドレスです。

port は、登録サービス・プロバイダー・パイプラインが使用するポート番号です。

その他はすべて表示どおりに正確にコーディングします。ストリング cicswsat/RegistrationService は、URIMAP DFHRSURI の PATH 属性に対応します。 例を次に示します。

```
<registration_service_endpoint>
provider.example.com:7160/cicswsat/RegistrationService
</registration_service_endpoint>
```

WS-AtomicTransaction に合わせたサービス・リクエスター・アプリケーションの構成

サービス・リクエスター・アプリケーションの目的が、WS-AtomicTransaction プロトコルを使用して分散トランザクションに参加することである場合、パイプライン構成ファイルには、<headerprogram> および <service_parameter_list> を指定する必要があります。

サービス・リクエスター・アプリケーションが WS-AtomicTransaction に参加できるようにするには、サービス・リクエスター・アプリケーションが SOAP プロトコルを使用してサービス・プロバイダーと通信し、かつパイプラインを構成して、CICS 提供の SOAP メッセージ・ハンドラーのいずれかを使用する必要があります。 サービス・リクエスター・アプリケーションを正しく構成した場合でも、サービス・プロバイダー・アプリケーションの参加が設定されていると、サービス・リクエスター・アプリケーションは、サービス・プロバイダーとの WS-AtomicTransaction にのみ参加します。

アプリケーションに固有のパイプライン構成情報に加えて、構成ファイルには、アプリケーションが WS-AtomicTransaction に必ず参加するように、CICS が使用する情報が格納されている必要があります。

CICS では、この情報が格納されているパイプライン構成ファイルの例を、ファイル /usr/lpp/cicsts/apobsf/samples/pipelines/wsatrequester.xml で提供しています。

1. <headerprogram> エレメントを <cics_soap_1.1_handler> エレメントまたは <cics_soap_1.2_handler> エレメントの内部にコーディングします。
   <program_name>、<namespace>、<localname>、 <mandatory> の各エレメントを、次に示す例のとおり正確にコーディングします。 例を次に示します。

   ```
   <cics_soap_1.1_handler>
     <headerprogram>
       <program_name>DFHWSATH</program_name>
       <namespace>http://schemas.xmlsoap.org/ws/2004/10/wscoor</namespace>
       <localname>CoordinationContext</localname>
       <mandatory>true</mandatory>
     </headerprogram>
   </cics_soap_1.1_handler>
   ```

   その他の <headerprogram> エレメントがアプリケーションに必要な場合は、それらも指定できます。

2. <registration_service_endpoint> エレメントを <service_parameter_list> の内部にコーディングします。 <registration_service_endpoint> を、次のようにコーディングします。

```
<registration_service_endpoint>
address:port/cicswsat/RegistrationService
</registration_service_endpoint>
```

ここで

address は、登録サービス・プロバイダー・パイプラインがインストールされている CICS 領域の IP アドレスです。

port は、登録サービス・プロバイダー・パイプラインが使用するポート番号です。

その他はすべて表示どおりに正確にコーディングします。 例を次に示します。

```
<registration_service_endpoint>
requester.example.com:7159/cicswsat/RegistrationService
</registration_service_endpoint>
```

第 14 章 CICS 実例アプリケーション

CICS 実例アプリケーションとは、CICS アプリケーションを外部のクライアントおよびサーバーに接続するときの最良事例を示すために設計された実用的な COBOL アプリケーションのことです。

実例アプリケーションは、簡単な販売カタログと注文処理のアプリケーションで構成されており、ここでは、エンド・ユーザーが以下の機能を実行できます。

- カタログ内の品目をリストする。
- カタログ内の個々の品目について問い合わせる。
- カタログを基に品目を注文する。

カタログは VSAM ファイルとして実装されています。

ベース・アプリケーションは 3270 ユーザー・インターフェースを備えていますが、明確に定義されたコンポーネント間インターフェースを備えたモジュラー構造により、コンポーネントを追加できます。特に、このアプリケーションは Web サービス・サポートに付属しており、既存のアプリケーションを Web サービス環境に拡張する方法を示す目的で設計されています。

ベース・アプリケーション

ベース・アプリケーションと、その 3270 ユーザー・インターフェースによって提供される機能を使用すると、保管カタログの内容をリスト表示し、このリストから品目を選択して、注文数量を入力できます。 アプリケーションは、モジュラー設計になっています。これにより、アプリケーションを拡張して Web サービスなどの新技術をサポートすることが簡単になります。

図 27 は、ベース・アプリケーションの構造を示しています。

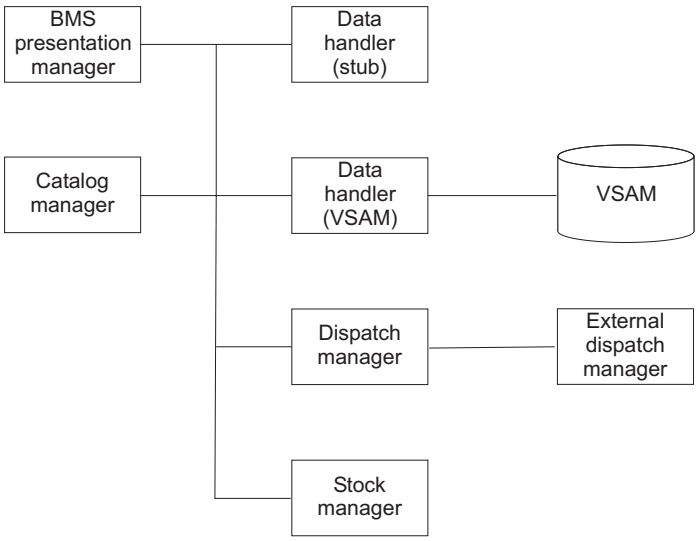


図 27. ベース・アプリケーションの構造

ベース・アプリケーションのインストールおよびセットアップ

ベース・アプリケーションを実行するには、その前に 2 つの VSAM データ・セットを定義して取り込み、2 つのトランザクション定義をインストールする必要があります。

VSAM データ・セットの作成および定義

実例アプリケーションでは、定義とデータ取り込みの対象となる 2 つの KSDS VSAM データ・セットが使用されます。 一方のデータ・セットには、実例アプリケーションの構成情報が格納されています。 もう一方には、販売カタログが格納されています。

1. JCL を検索して、VSAM データ・セットを作成します。 CICS のインストール時に、JCL は、階層ファイル・システム (HFS) の samples/webservices/JCL ディレクトリーに置かれます。
   - ファイル EXMPCONF には、構成データ・セットを生成するための JCL が記述されています。
   - ファイル EXMPCAT には、カタログ式データ・セットを生成するための JCL が記述されています。
2. JCL とアクセス方式サービス・プログラム・コマンドを変更します。
   a. 有効な JOB カードを入力します。
   b. アクセス方式サービス・プログラム・コマンドのデータ・セット名に、適切な高位修飾子を入力します。 JCL は、高位修飾子の HLQ の部分に、入力に応じた内容を使用します。

   以下のコマンドでは、カタログ・ファイルが定義されます。

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

   以下のコマンドでは、構成ファイルが定義されます。

```
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) -
          )
```

   ここで、*hlq* は、選択した高位修飾子を表します。

3. 両方のジョブを実行して、データ・セットを作成し、データを取り込みます。
4. CEDA トランザクションを使用して、カタログ・ファイルの FILE 定義を作成します。
   a. CEDA DEF FILE(EXMPCAT) G(EXAMPLE) と入力します。
   b. 以下の追加属性を入力します。

      DSNAME(*hlq*.EXMPLAPP.EXMPCAT)
      ADD(YES)
      BROWSE(YES)
      DELETE(YES)
      READ(YES)
      UPDATE(YES)

   c. その他の属性には、すべてデフォルト値を使用します。
5. CEDA トランザクションを使用して、構成ファイルの FILE 定義を作成します。
   a. CEDA DEF FILE(EXMPCAT) G(EXAMPLE) と入力します。
   b. 以下の追加属性を入力します。

      DSNAME(*hlq*.EXMPLAPP.EXMPCONF)
      ADD(YES)
      BROWSE(YES)
      DELETE(YES)
      READ(YES)
      UPDATE(YES)

   c. その他の属性には、すべてデフォルト値を使用します。

3270 インターフェースの定義

実例アプリケーションには、アプリケーションを実行してカスタマイズするための 3270 ユーザー・インターフェースが用意されています。 このユーザー・インターフェースは、EGUI と EGFG の 2 つのトランザクションで構成されます。

CEDA トランザクションを使用して、両方のトランザクションの TRANSACTION 定義を作成します。

1. トランザクション EGUI を定義するには、CEDA DEF TRANS (EGUI) G(EXAMPLE) PROG(DFH0XGUI) と入力します。
2. トランザクション EGFG を定義するには、CEDA DEF TRANS (EGFG) G(EXAMPLE) PROG(DFH0XCFG) と入力します。

その他の属性には、すべてデフォルト値を使用します。

注: 実例アプリケーションが正常に動作するかどうかは、トランザクションの名前には依存しないため、必要に応じて別の名前を使用できます。

インストールの完了

インストールを完了するには、リソース定義が格納されている RDO グループをインストールします。

CICS 端末で、CEDA I G(EXAMPLE) というコマンドを入力します。

これで、Web サービスとしてアプリケーションにアクセスする準備ができました。

実例アプリケーションに対する Web サービス・サポート

Web サービス・サポートは、実例アプリケーションを拡張して、Web クライアント・フロントエンドと Web サービス・エンドポイントを、オーダー・ディスパッチャー・コンポーネントに提供します。

Web クライアント・フロントエンドおよび Web サービス・エンドポイントは、以下の環境で稼働するエンタープライズ・アーカイブ (EAR) として提供されます。

- WebSphere Application Server バージョン 5 リリース 1 以上
- WebSphere 単体テスト環境を備えた WebSphere Studio Application Developer バージョン 5 リリース 1 以上
- WebSphere 単体テスト環境を備えた WebSphere Studio Enterprise Developer バージョン 5 リリース 1 以上

Web サービス・サポートのインストール

実例アプリケーションに対して Web サービス・サポートを実行するには、その前に 2 つの HFS ディレクトリーを作成し、いくつかの CICS リソース定義を作成してインストールしておく必要があります。

HFS ディレクトリーの作成

実例アプリケーションの Web サービス・サポートでは、階層ファイル・システム (HFS) にシェルフ・ディレクトリーとピックアップ・ディレクトリーが必要です。

シェルフ・ディレクトリーは、WEBSERVICE リソースに関連付けられている Web サービス・バインディング・ファイルを格納するために使用します。各 WEBSERVICE リソースは、順に PIPELINE に関連付けられます。 シェルフ・ディレクトリーは PIPELINE リソースに管理されるため、この内容は直接変更しないでください。CICS では、PIPELINE ごとにシェルフ・ディレクトリーの下位に固有のディレクトリー構造が確保されるため、いくつかの PIPELINES が同じシェルフ・ディレクトリーを使用できます。

ピックアップ・ディレクトリーとは、PIPELINE と関連付けられている Web サービス・バインディング・ファイルが格納されているディレクトリーのことです。PIPELINE がインストールされると、または PERFORM PIPELINE SCAN コマンドに対する対応として、バインディング・ファイルの情報が使用され、PIPELINE に関連した WEBSERVICE 定義や URIMAP 定義が動的に作成されます。

実例アプリケーションは、シェルフ・ディレクトリーとして /var/cicsts を使用します。 これは、CICS 付属のサンプル・ディレクトリーであり、ここには、WSDIR の Web サービス・バインディング・ファイル例と、付属のサンプル構成ファイルが格納されています。

パイプラインは、XML 構成ファイル内をインストール時に読み取ります。このため、これらのファイルの格納先ディレクトリーを定義することも有益です。

PIPELINE 定義の作成

パイプラインの完全な定義は、PIPELINE リソースとパイプライン構成ファイルで構成されます。 このファイルには、Web サービス要求および Web サービス応答がパイプラインをパススルーするときに、これらに作用するメッセージ・ハンドラーの詳細が記述されています。

実例アプリケーションでは、CICS 提供の SOAP 1.1 ハンドラーを使用して、インバウンド要求およびアウトバウンド要求の SOAP エンベロープを処理します。CICS には、サービス・プロバイダーおよびサービス・リクエスターで使用できるサンプルのパイプライン構成ファイルが用意されています。

複数の WEBSERVICE が 1 つの PIPELINE を共用できるため、実例アプリケーションのインバウンド要求に定義するパイプラインは 1 つのみにする必要があります。ただし、1 つの PIPELINE を、同時にプロバイダー・パイプラインとリクエスター・パイプラインの両方になるように構成することはできないため、アウトバウンド要求に対しては 2 番目の PIPELINE を定義する必要があります。

1. CEDA トランザクションを使用して、サービス・プロバイダーの PIPELINE 定義を作成します。
   a. `CEDA DEF PIPE(EXPIPE01) G(EXAMPLE)` と入力します。
   b. 以下の追加属性を入力します。

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

      HFS の項目では大/小文字が区別され、デフォルトの CICS HFS インストール・ルートは `/usr/lpp/cicsts` であることが前提になっていることに注意してください。
2. CEDA トランザクションを使用して、サービス・リクエスターの PIPELINE 定義を作成します。
   a. `CEDA DEF PIPE(EXPIPE02) G(EXAMPLE)` と入力します。
   b. 以下の追加属性を入力します。

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

      HFS の項目では大/小文字が区別され、デフォルトの CICS HFS インストール・ルートは `/usr/lpp/cicsts` であることが前提になっていることに注意してください。

TCPIPSERVICE の作成

クライアントは HTTP トランスポートを介して Web サービスに接続するため、TCPIPSERVICE を定義してインバウンド HTTP トラフィックを受信する必要があります。

インバウンド HTTP 要求を処理するには、CEDA トランザクションを使用して TCPIPSERVICE 定義を作成します。

1. `CEDA DEF TCPIPSERVICE(SOAPPORT) G(EXAMPLE)` と入力します。
2. 以下の追加属性を入力します。

   `URM(NONE)`

   `PORTNUMBER(`*port*`)` ここで、*port* は CICS システムにおける未使用のポート番号です。

   `PROTOCOL(HTTP)`

   `TRANSACTION(CWXN)`
3. その他の属性には、すべてデフォルト値を使用します。

WEBSERVICE リソースおよび URIMAP リソースの動的なインストール

Web サービスとして公開される各機能には、SOAP BODY の着信 XML とプログラムの COMMAREA インターフェース間をマップするための WEBSERVICE リソースと、着信要求を正しい PIPELINE および WEBSERVICE に送信する URIMAP リソースが必要です。RDO を使用して WEBSERVICE リソースと URIMAP リソースを定義してインストールできますが、PIPELINE リソースをインストールした場合は、CICS を使用しても、これらのリソースを動的に作成できます。

PIPELINE リソースをインストールします。 以下のコマンドを使用します。

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

各 PIPELINE リソースをインストールすると、CICS は、PIPELINE の WSDIR 属性で指定されたディレクトリー (ピックアップ・ディレクトリー) をスキャンします。このディレクトリーの Web サービス・バインディング・ファイルごとに、つまり、`.wsbind` という接尾部を持つファイルごとに、CICS は、WEBSERVICE および URIMAP のいずれかが存在しなかった場合、これらをインストールします。バインディング・ファイル内の情報の方が既存のリソースよりも新しい場合、既存のリソースは置き換えられます。

PIPELINE が後で使用不可になり、破棄されると、関連付けられていたすべての WEBSERVICE リソースおよび URIMAP リソースも破棄されます。

PIPELINE をインストール済みの場合は、PERFORM PIPELINE SCAN コマンドを発行して、PIPELINE のピックアップ・ディレクトリーのスキャンを開始してください。 PIPELINE をインストールすると、以下の WEBSERVICE およびその関連 URIMAP がシステムにインストールされます。

```
dispatchOrder
dispatchOrderEndpoint
inquireCatalog
```

```
inquireSingle

placeOrder
```

WEBSERVICE の名前は、Web サービス・バインディング・ファイルの名前から得られます。URIMAP の名前は動的に生成されます。 CEMT INQUIRE WEBSERVICE コマンドを使用すると、リソースを表示できます。

```
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)
```

この表示には、PIPELINE や URIMAP の名前、および各 WEBSERVICE に関連したターゲット・プログラムの名前が表示されています。この例では、WEBSERVICE はアウトバウンド要求を対象にしているため、WEBSERVICE(dispatchOrder) には URIMAP もターゲット・プログラムも表示されないことに注意してください。

WEBSERVICE(dispatchOrderEndpoint) は、注文発送サービスのローカル側 CICS 実装を表しています。

RDO による WEBSERVICE リソースの作成

PIPELINE スキャン機能を使用して WEBSERVICE リソースをインストールする代わりに、オンライン・リソース定義 (RDO) を使用して WEBSERVICE リソースを作成し、インストールすることができます。

重要: RDO を使用して WEBSERVICE リソースおよび URIMAP リソースを定義する場合は、Web サービス・バインディング・ファイルが PIPELINE のピックアップ・ディレクトリーに**存在しない**ことを確認する必要があります。

1. CEDA トランザクションを使用して、実例アプリケーションの INQUIRE 機能の WEBSERVICE 定義を作成します。
   a. `CEDA DEF WEBSERVICE(EXINQCWS) G(EXAMPLE)` と入力します。
   b. 以下の追加属性を入力します。

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

2. 実例アプリケーションの以下の機能ごとに、前のステップを繰り返します。

機能	WEBSERVICE 名	PIPELINE 属性	WSBIND 属性
INQUIRE SINGLE ITEM	EXINQSWS	EXPIPE01	/usr/lss/cicsts/samples /webservices/wsbind /provider/inquireSingle.wsbind
PLACE ORDER	EXORDRWS	EXPIPE01	/usr/lss/cicsts/samples /webservices/wsbind /provider/placeOrder.wsbind

機能	WEBSERVICE 名	PIPELINE 属性	WSBIND 属性
DISPATCH STOCK	EXODRQWS	EXPIPE01	/usr/lss/cicsts/samples /webservices/wsbind /provider/dispatchOrder.wsbind
DISPATCH STOCK endpoint (オプショナル)	EXODEPWS	EXPIPE01	/usr/lss/cicsts/samples /webservices/wsbind /requester/dispatchOrderEndpoint.wsbind

RDO による URIMAP リソースの作成

PIPELINE スキャン機能を使用して URIMAP リソースをインストールする代わりに、オンライン・リソース定義 (RDO) を使用して URIMAP リソースを作成し、インストールすることができます。

重要: RDO を使用して WEBSERVICE リソースおよび URIMAP リソースを定義する場合は、Web サービス・バインディング・ファイルが PIPELINE のピックアップ・ディレクトリーに**存在しない**ことを確認する必要があります。

1. CEDA トランザクションを使用して、実例アプリケーションの INQUIRE 機能の URIMAP 定義を作成します。
   a. `CEDA DEF URIMAP(INQCURI) G(EXAMPLE)` と入力します。
   b. 以下の追加属性を入力します。

      ```
      USAGE(PIPELINE)
      HOST(*)
      PATH(/exampleApp/inquireCatalog)
      TCPIPSERVICE(SOAPPORT)
      PIPELINE(EXPIPE01)
      WEBSERVICE(EXINQCWS)
      ```
2. 実例アプリケーションの残りの機能ごとに、前のステップを繰り返します。URIMAP には、以下の名前を使用します。

機能	URIMAP 名
INQUIRE SINGLE ITEM	INQSURI
PLACE ORDER	ORDRURI
DISPATCH STOCK	必要なし
DISPATCH STOCK endpoint (オプショナル)	ODEPURI

   a. URIMAP ごとに以下に示す別個の属性を指定します。

機能	URIMAP 名	PATH	WEBSERVICE
INQUIRE SINGLE ITEM	INQSURI	/exampleApp/inquireSingle	EXINQSWS
PLACE ORDER	ORDRURI	/exampleApp/placeOrder	EXORDRWS
ORDER DISPATCH endpoint (オプショナル)	ODEPURI	/exampleApp/dispatchOrder	EXODEPWS

b. 以下の追加属性を入力します。これらの属性は、すべての URIMAP で同じです。

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

インストールの完了

インストールを完了するには、リソース定義が格納されている RDO グループをインストールします。

CICS 端末で、`CEDA I G(EXAMPLE)` というコマンドを入力します。

これで、Web サービスとしてアプリケーションにアクセスする準備ができました。

実例アプリケーションの構成

ベース・アプリケーションには、実例アプリケーションを構成するために使用できるトランザクション (ECFG) が組み込まれています。

構成トランザクションでは、大/小文字混合情報を使用します。大/小文字混合情報を正しく処理できる端末を使用する必要があります。

トランザクションにより、実例アプリケーションのいくつかの性質を指定できます。この内容は次のとおりです。

- Web サービスの使用など、アプリケーションの全体的な構成
- アプリケーションの Web サービス・コンポーネントが使用するネットワーク・アドレス
- データ・ストアに使用されるファイルなどのリソースの名前
- アプリケーションの各コンポーネントに使用されるプログラムの名前

構成トランザクションでは、実例アプリケーションの CICS 提供コンポーネントを、アプリケーションを再始動することなく、 独自のコンポーネントに置き換えることができます。

1. トランザクション ECFG を入力して、構成アプリケーションを開始します。CICS では、次の画面が表示されます。

```
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. フィールドに値を入力します。

Datastore Type (データ・ストア・タイプ)
Data Store Stub プログラムを使用する場合は、STUB を指定します。

VSAM データ・ストアを使用する場合は、VSAM を指定します。

Outbound WebService (アウトバウンド WebService)
Order Dispatch (注文発送) 機能にリモート Web サービスを使用する場合は、YES を指定します。

Order Dispatch (注文発送) 機能にスタブ・プログラムを使用する場合は、NO を指定します。

Catalog Manager (カタログ・マネージャー)
Catalog Manager (カタログ・マネージャー) プログラムの名前を指定します。実例アプリケーションに付属するプログラムは DFH0XCMN です。

Data Store Stub (データ・ストア・スタブ)
「Datastore Type (データ・ストア・タイプ)」フィールドに STUB を指定した場合は、Data Store Stub (データ・ストア・スタブ) プログラムの名前を指定します。実例アプリケーションに付属するプログラムは DFH0XSDS です。

Data Store VSAM (データ・ストア VSAM)
「Datastore Type (データ・ストア・タイプ)」フィールドに VSAM を指定した場合は、VSAM データ・ストア・プログラムの名前を指定します。実例アプリケーションに付属するプログラムは DFH0XVDS です。

Order Dispatch Stub (注文発送スタブ)
Order Dispatch Stub (注文発送スタブ) プログラムの名前を指定します。実例アプリケーションに付属するプログラムは DFH0XSOD です。

Order Dispatch WebService (注文発送 WebService)
「Outbound WebService (アウトバウンド WebService)」フィールドに

YES を指定した場合は、サービス・リクエスターとして機能するプログラムの名前を指定します。実例アプリケーションに付属するプログラムは DFH0XWOD です。

Stock Manager (在庫マネージャー)
Stock Manager (在庫マネージャー) プログラムの名前を指定します。実例アプリケーションに付属するプログラムは DFH0XSSM です。

VSAM File Name (VSAM ファイル名)
「Datastore Type (データ・ストア・タイプ)」フィールドに VSAM を指定した場合は、CICS FILE 定義の名前を指定します。提供された実例アプリケーションで使用されている名前は、EXMPCAT です。

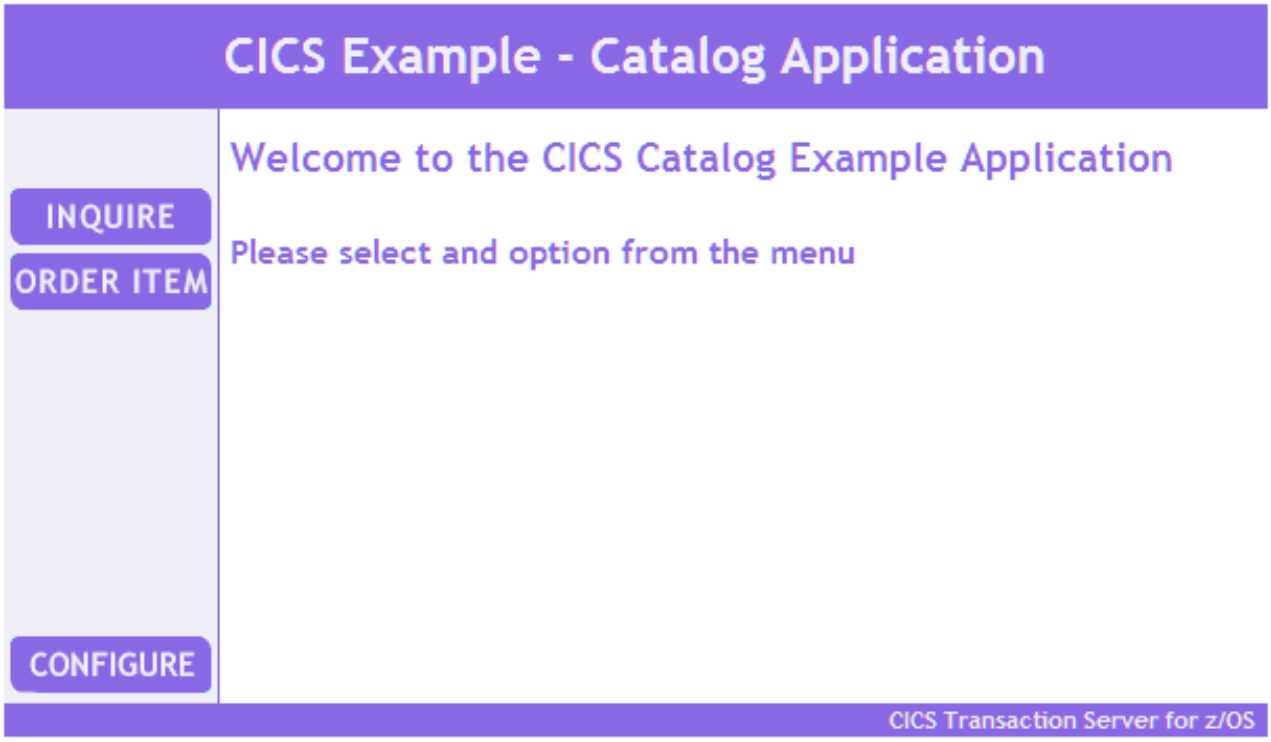
Server Address and Port (サーバー・アドレスおよびポート)

Outbound WebService URI (アウトバウンド WebService URI)
「Outbound WebService (アウトバウンド WebService)」フィールドに YES を指定した場合は、注文発送機能を実装する Web サービスのロケーションを指定します。提供された CICS エンドポイントを使用している場合は、これを `http://myserver:myport/exampleApp/dispatchOrder` に指定します。ここで、*myserver* および *myport* は、それぞれ使用している CICS サーバーのアドレスとポートです。

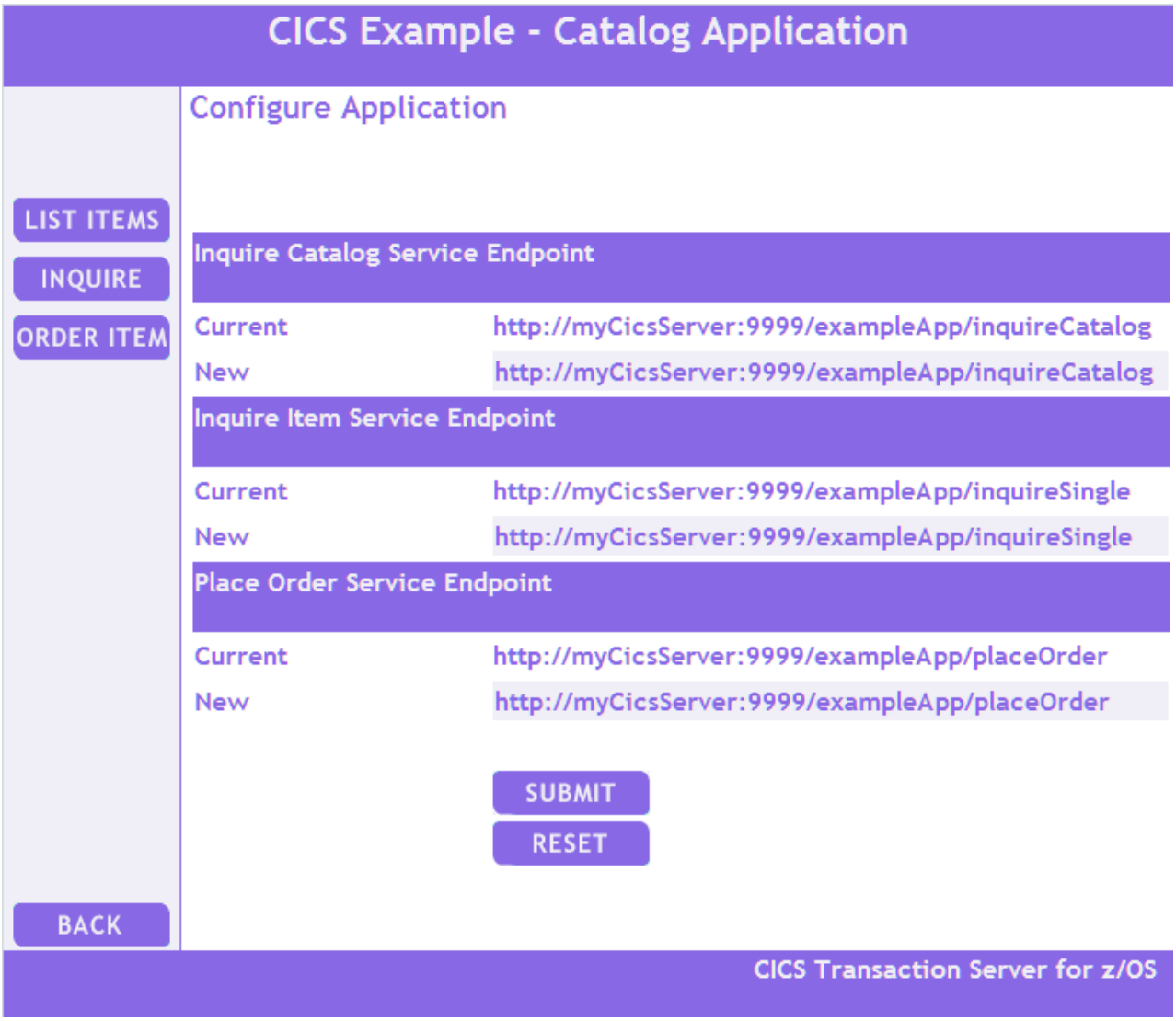
Web クライアントの構成

適切な CICS システムを呼び出すには、最初に Web クライアントを構成することが必要です。

1. Web ブラウザーで、次のように入力します。
   `http://myserver:9080/ExampleAppClientWeb/`。ここで、*myserver* は、Web サービス・クライアントのインストール先サーバーのホスト名です。 実例アプリケーションにより、次のページが表示されます。



2. **「構成 (CONFIGURE)」**ボタンをクリックして、構成ページを表示します。 構成ページが表示されます。



3. Web サービスの新しいエンドポイントを入力します。
   a. URL では、ストリング「myCicsServer」を、CICS が動作しているシステムの名前に置き換えます。
   b. ポート番号「9999」を、TCPIPSERVICE で構成したポート番号 (この例では、30000) に置き換えます。
4. この操作を、オーダーの発行と問い合わせの両方のエンドポイントについて実行します。
5. **「発注登録 (SUBMIT)」**ボタンをクリックします。

これで、Web アプリケーションを実行する準備ができました。

注: Web サービスによって呼び出される URL は、HTTP セッションで保管されます。したがって、ブラウザーが初めてクライアントに接続されるたびに、この構成ステップを繰り返す必要があります。

実例アプリケーションの実行

実例アプリケーションは、BMS インターフェースを使用する方法と Web クライアントを使用する方法の 2 つの方法で実行できます。

BMS インターフェースによる実例アプリケーションの実行

ベース・アプリケーションは、その BMS インターフェースを使用して起動できます。

1. CICS 端末からトランザクション EGUI を入力します。 実例アプリケーションにより、次のメニューが表示されます。

```
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
```

   メニューのオプションを選択すると、カタログ内の品目のリスト表示、品目の注文、アプリケーションの終了のいずれかを実行できます。

2. 「1」と入力し、ENTER キーを押して、「LIST ITEMS」オプションを選択します。 アプリケーションにより、カタログ内の品目リストが表示されます。

```
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. 「**ORDER**」列に「/」と入力し、ENTER キーを押して、品目を注文します。 アプリケーションにより、注文した品目の詳細が表示されます。

```
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. 注文を受けられるだけの十分な在庫がある場合は、以下の情報を入力します。
   a. 「ORDER QUANTITY (注文数量)」フィールドに値を入力します。 注文する品目の数を指定します。
   b. 「USERID (ユーザー ID)」フィールドに値を入力します。 1 から 8 文字のストリングを入力します。ベース・アプリケーションは、ここに入力された値を検査しません。
   c. 「CHARGE DEPT (課金部門)」フィールドに値を入力します。 1 から 8 文字のストリングを入力します。ベース・アプリケーションは、ここに入力された値を検査しません。
5. ENTER キーを押して注文をサブミットし、メインメニューに戻ります。

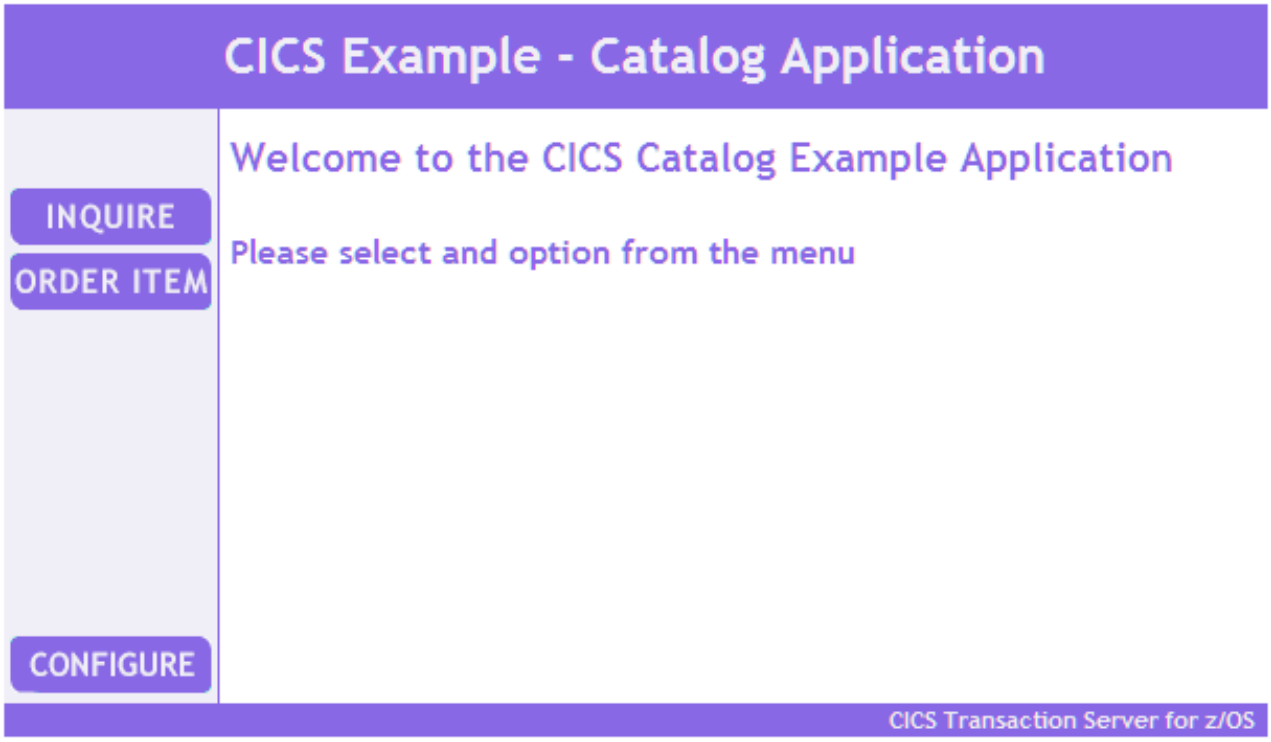
6. 「EXIT」オプションを選択して、アプリケーションを終了します。

Web サービス対応アプリケーション

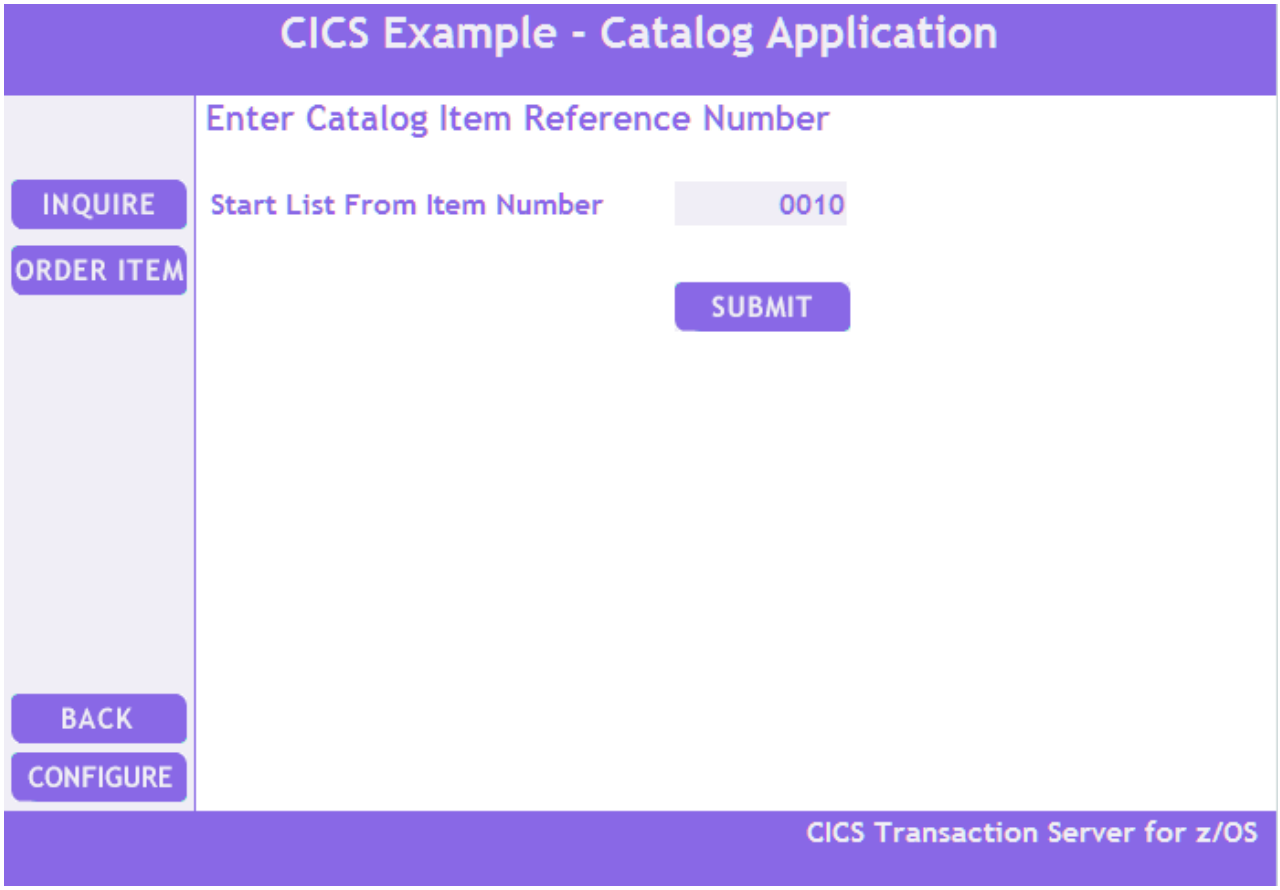
実例アプリケーションは Web ブラウザーから起動できます。

実例アプリケーションは、Web サービスとして動作するように構成する必要があります。

1. Web ブラウザーで、次のように入力します。
   `http://myserver:9080/ExampleAppClientWeb/`。ここで、*myserver* は、Web サービス・クライアントのインストール先サーバーのホスト名です。 実例アプリケーションにより、次のページが表示されます。



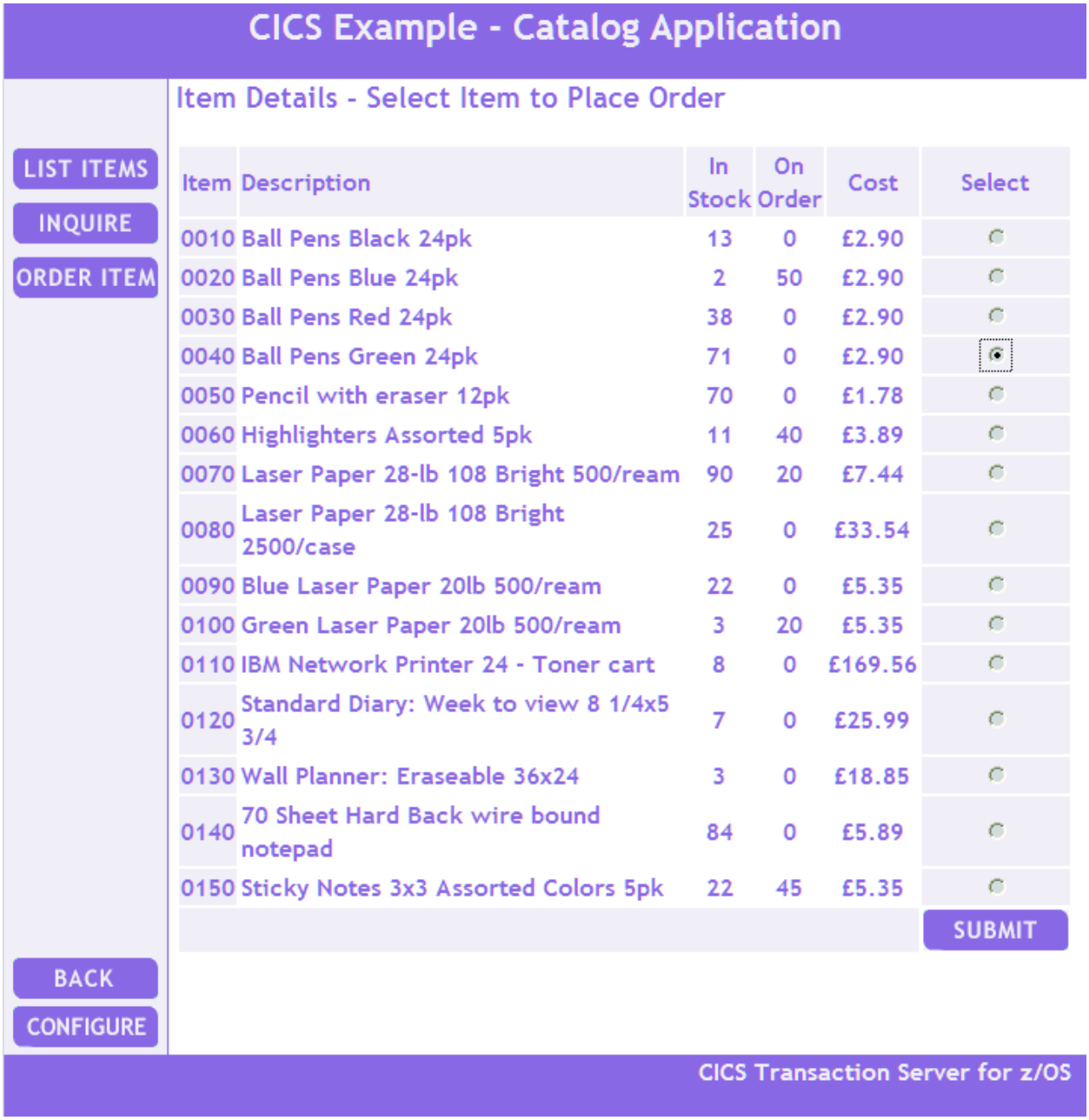
2. **「リスト項目 (LIST ITEMS)」**ボタンをクリックします。 実例アプリケーションにより、次のページが表示されます。



3. 品目番号を入力し、**「発注登録 (SUBMIT)」** ボタンをクリックします。

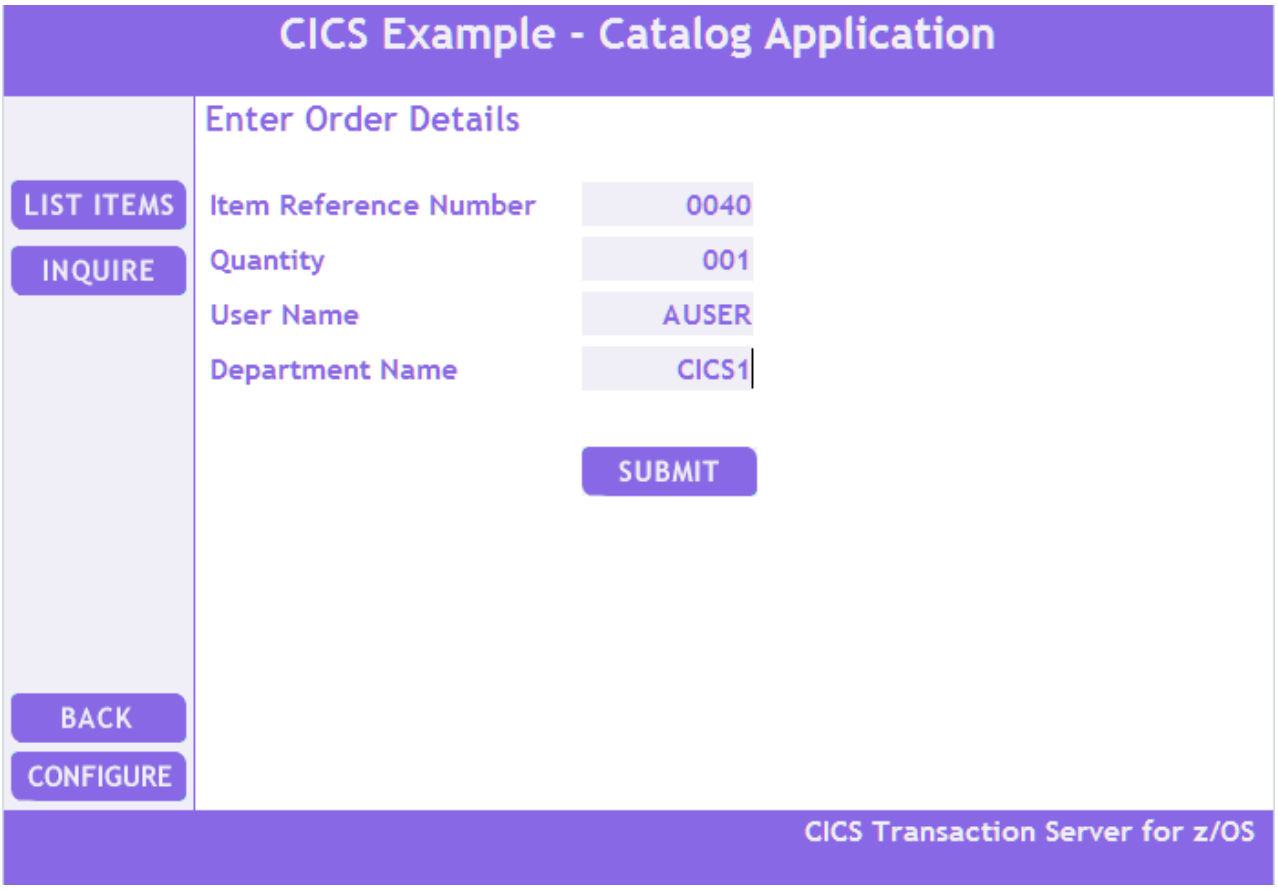
   ヒント: ベース・アプリケーションには、0010、0020、... の順に 0210 まで品目番号が付与されます。

   アプリケーションは、入力した品目番号から開始されたカタログの品目リストを含む、次のページを表示します。



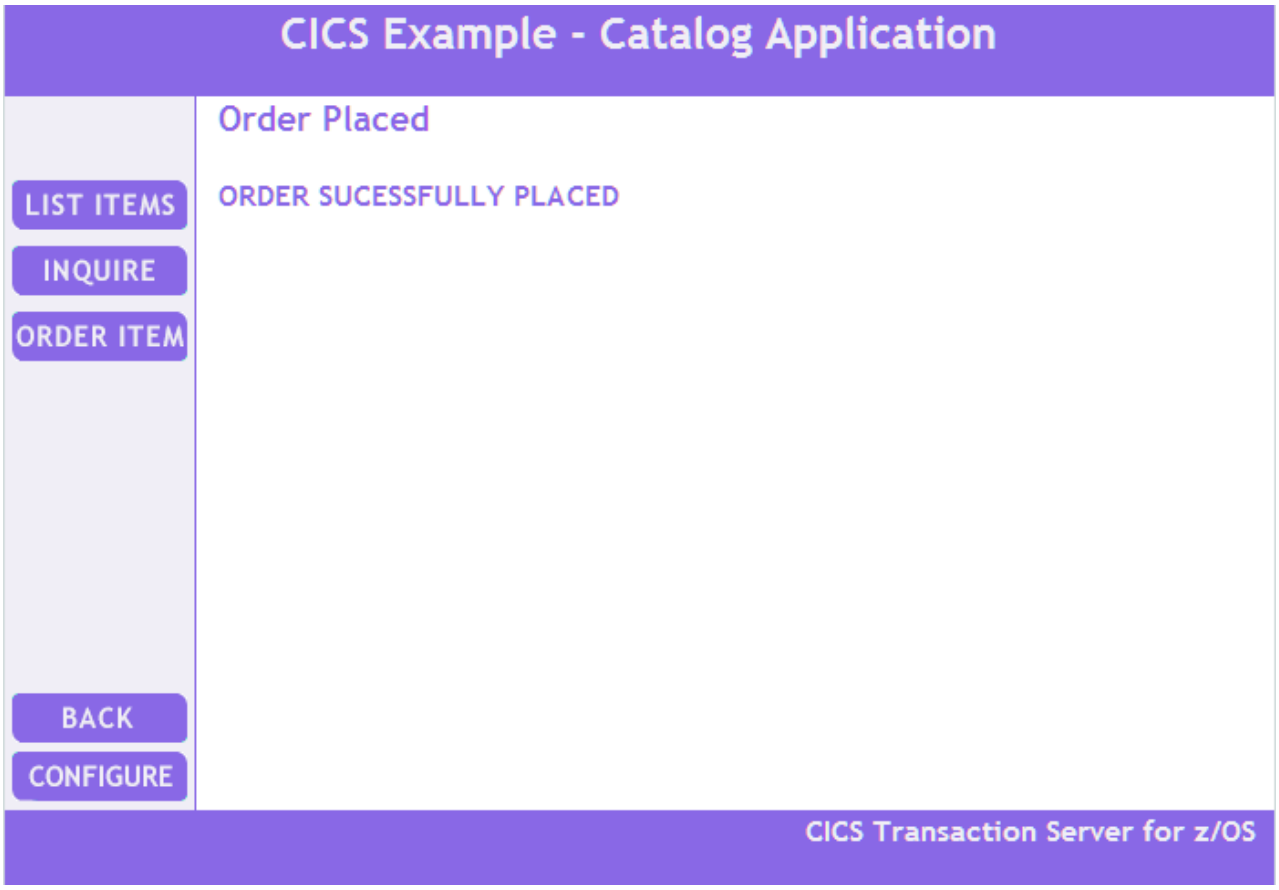
4. 注文する品目を選択します。
   a. 注文する品目の「**選択**」列のラジオ・ボタンをクリックします。
   b. 「**発注登録 (SUBMIT)**」ボタンをクリックします。

   アプリケーションにより、次のページが表示されます。



5. 発注するには、以下の情報を入力します。
   a. 「数量 (Quantity)」フィールドに値を入力します。 注文する品目の数を指定します。
   b. 「ユーザー名」フィールドに値を入力します。 1 から 8 文字のストリングを入力します。ベース・アプリケーションは、ここに入力された値を検査しません。
   c. 「部門名 (Department Name)」フィールドに値を入力します。 1 から 8 文字のストリングを入力します。ベース・アプリケーションは、ここに入力された値を検査しません。
   d. 「**発注登録 (SUBMIT)**」ボタンをクリックします。

   発注が行われたか確認するため、アプリケーションにより次のページが表示されます。



実例アプリケーションの配置

Web サービス・アシスタントを使用すると、実例アプリケーションの一部を Web サービスとして配置できます。 実例アプリケーションは、提供されたままの状態で、この作業を実行することなく動作しますが、独自のアプリケーションを配置して実例アプリケーションを拡張する場合は、類似の作業を実行する必要があります。

プログラム・インターフェースの抽出

CICS Web サービス・アシスタントを使用してプログラムを配置するには、プログラムの COMMAREA またはコンテナー・インターフェースに一致するコピーブックを作成する必要があります。

この例では、中央のカタログ・マネージャー・プログラム (DFH0XCMN) の INQUIRE SINGLE ITEM 機能が、Web サービスとして配置されます。このプログラムに対するインターフェースは、COMMAREA です。COMMAREAの構造は、コピーブック DFH0XCP1 で次のように定義されます。

```
*    カタログ COMMAREA 構造
           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).
```

```
*    Inquire Catalog (発注) で使用されているフィールド
     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).
*    Inquire Single (1 件の問い合わせ) で使用されているフィールド
     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).
*    Place Order (発注) で使用されているフィールド
     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).
```

コピーブックでは、INQUIRE CATALOG、 INQUIRE SINGLE ITEM、および PLACE ORDER の機能それぞれに対して、3 つの異なるインターフェースが定義されます。これらのインターフェースは、コピーブック内で互いにオーバーレイされています。ただし、DFHLS2WS ユーティリティーは、REDEFINES ステートメントをサポートしていません。 したがって、Inquire Single (1 件の問い合わせ) 機能に関連するセクションのみを結合コピーブックから抽出する必要があります。

```
*    カタログ COMMAREA 構造
     03 CA-REQUEST-ID            PIC X(6).
     03 CA-RETURN-CODE           PIC 9(2) DISPLAY.
     03 CA-RESPONSE-MESSAGE      PIC X(79).
*    Inquire Single (1 件の問い合わせ) で使用されているフィールド
     03 CA-INQUIRE-SINGLE.
         05 CA-ITEM-REF-REQ          PIC 9(4) DISPLAY.
         05 FILLER                   PIC X(4) DISPLAY.
         05 FILLER                   PIC X(3) DISPLAY.
         05 CA-SINGLE-ITEM.
             07 CA-SNGL-ITEM-REF     PIC 9(4) DISPLAY.
             07 CA-SNGL-DESCRIPTION  PIC X(40).
             07 CA-SNGL-DEPARTMENT   PIC 9(3) DISPLAY.
             07 CA-SNGL-COST         PIC X(6).
             07 IN-SNGL-STOCK        PIC 9(4) DISPLAY.
             07 ON-SNGL-ORDER        PIC 9(3) DISPLAY.
    05 FILLER                   PIC X(840).
```

再定義のエレメント CA-REQUEST-SPECIFIC は、除去され、Inquire Single (1 件の問い合わせ) 機能に対してこのエレメントを再定義したコピーブックの部分によって置き換えられます。これで、このコピーブックは、Web サービス・アシスタントとの組み合わせ使用に適合するようになりました。

このコピーブックは、コピーブック DFH0XCP4 として実例アプリケーションに付属しています。

Web サービス・アシスタント・プログラム DFHLS2WS の実行

CICS Web サービス・アシスタントは、2 つのバッチ・プログラムで構成されており、既存の CICS アプリケーションを Web サービスに変換するのに役立ちます。また、Web サービス・アシスタントを使用すると、CICS アプリケーションが、外部のプロバイダーによって提供された Web サービスを使用できるようになります。 プログラム DFHLS2WS は、言語構造を変換してWeb サービス・バインディング・ファイルと Web サービス記述を生成します。

1. 付属のサンプル JCL を適切な作業ファイルにコピーします。 JCL は、samples/webservice/LS2WS にあります。
2. 有効な JOB カードを JCL に追加します。
3. DFHLS2WS のパラメーターを指定します。 実例アプリケーションの INQUIRE SINGLE ITEM 機能に必要なパラメーターは、次のとおりです。

```
//INPUT.SYSUT1 DD *
LOGFILE=/u/exampleapp/wsbind/inquireSingle.log
PDSLIB=//CICSHLQ.SDFHSAMP
REQMEM=DFH0XCP4
RESPMEM=DFH0XCP4
LANG=COBOL
PGMNAME=DFH0XCMN
PGMINT=COMMAREA
URI=exampleApp/inquireSingle
WSBIND=/u/exampleapp/wsbind/inquireSingle.wsbind
WSDL=/u/exampleapp/wsdl/inquireSingle.wsdl
*/
```

パラメーターは以下のとおりです。

LOGFILE=/u/exampleapp/wsbind/inquireSingle.log
DFHLS2WS から診断情報を記録するときに使用するファイル。このファイルを使用するのは、通常、IBM のソフトウェア・サポート組織のみです。

PDSLIB=//CICSHLQ.SDFHSAMP
要求構造と応答構造を定義するコピーブックが、Web サービス・アシスタントによって検索される区分データ・セット (PDS) の名前。この例では、CICS によってインストールされたデータ・セットの SDFHSAMP になります。

REQMEM=DFH0XCP4
RESPMEM=DFH0XCP4
これらのパラメーターは、プログラムに対する要求と応答の言語構造を定義します。この例では、要求と応答の構造は同じであり、同じコピーブックによって定義されます。

LANG=COBOL
ターゲット・プログラムおよびデータ構造は、COBOL で記述されます。

PGMNAME=DFH0XCMN
Web サービス要求を受け取ると呼び出されるターゲット・プログラムの名前。

PGMINT=COMMAREA
ターゲット・プログラムは、COMMAREA インターフェースによって呼び出されます。

URI=exampleApp/inquireSingle

URI の固有の部分。この URI は、生成された Web サービス定義で使用され、着信要求を正しい Web サービスにマップする URIMAP リソースを作成するために使用されます。指定された値により、外部クライアントに対して、サービスが次の場所で利用可能になります。

`http://mycicsserver:myport/exampleApp/inquireSingle`

ここで、*mycicsserver* および *myport* は、この WEBSERVICE がインストールされている CICS サーバー・アドレスおよびポートを表します。

注: このパラメーターには、先頭に「/」が**ありません**。

WSBIND=/u/exampleapp/wsbind/inquireSingle.wsbind
Web サービス・バインディング・ファイルの書き込み先となる HFS 上の場所。

注: このファイルが PIPELINE スキャン機能と組み合わせて使用される場合、このファイルには拡張子 `.wsbind` が**必要**です。

WSDL=/u/exampleapp/wsdl/inquireSingle.wsdl
生成された Web サービス記述が格納されているファイルの場所。規則により、このファイルには拡張子 `.wsdl` が付きます。 Web サービス・バインディング・ファイルと、これに対応する Web サービス記述にマッチングする名前を使用する習慣をつけておくことをお勧めします。

Web サービス記述は、クライアントが Web サービスにアクセスするために必要な情報を提供します。ここには、要求と応答の XML スキーマ定義と、サービスのロケーション情報が格納されています。

Web サービス記述をクライアントに公開するには、その前にサービス・ロケーションをカスタマイズする必要があります。生成したままの状態では、`<service>` エレメントの内容は次のようになっています。

```
<service name="DFHCMNService">
<port binding="tns:DFHCMNHTTPSoapBinding" name="DFHCMNPort">
<soap:address location="http://my-server:my-port/exampleApp/inquireSingle"/>
</port>
</service>
```

Web サービス記述をカスタマイズするには、*my-server* と *my-port* を、それぞれ CICS サーバー・ロケーションとポート番号に置き換えます。

4. ジョブを実行します。

Web サービス記述と Web サービス・バインディング・ファイルが、指定のロケーションに作成されます。

Web サービス・バインディング・ファイルの配置

DFHLS2WS によって作成された Web サービス・バインディング・ファイルは、PIPELINE リソースのインストール時に CICS 領域に動的に配置されます。

PIPELINE スキャン・コマンドを実行すると、CICS は、CEMT P PIPELINE() SCAN を介して明示的に、または PIPELINE インストール時に自動的にピックアップ・ディレクトリーをスキャンして、Web サービス・バインディング・ファイル、つまり .wsbind という拡張子を持つファイル名を検索します。CICS は、見つかったバインディング・ファイルごとに、WEBSERVICE リソースをインストールするかどうかを判別します。

URIMAP リソースも JCL に用意されているように作成され、これにより、インストール済み WEBSERVICE と、WEBSERVICE のインストール先 PIPELINE に URI がマップされます。 スキャンされた WEBSERVICE が破棄されると、これに関連付けられている URIMAP も破棄されます。

1. プロバイダー・パイプラインの PIPELINE 定義である PIPELINE(EXPIPE01) を変更します。 WSDIR パラメーターを /u/exampleapp/wsbind に変更します。このピックアップ・ディレクトリーには、DFHLS2WS を使用して生成したWeb サービス・バインディング・ファイルが格納されています。
2. アプリケーションが使用するその他の Web サービス・バインディング・ファイルを同じディレクトリーにコピーします。 この例では、コピーの対象ファイルは次のとおりです。

   ```
   inquireCatalog
   placeOrder
   ```

   これらのファイルは、ディレクトリー /usr/lpp/cicsts/samples/webservices/wsbind/provider にあります。
3. PIPELINE をインストールします。 CICS は、Web サービス・バインディング・ファイルを基にして、WEBSERVICE リソースと URIMAP リソースを作成します。

ベース・アプリケーションのコンポーネント

表 2. アプリケーションのコンポーネント

ファイル名	タイプ	コメント
DFH0XCMN	COBOL ソース	カタログ・マネージャー・アプリケーションのソース・コード
DFH0XVDS	COBOL ソース	VSAM データ・ストア・モジュールのソース・コード
DFH0XSDS	COBOL ソース	スタブ化データ・ストア・モジュールのソース・コード
DFH0XSOD	COBOL ソース	注文発送モジュールのスタブ化版のソース・コード
DFH0XWOD	COBOL ソース	アウトバウンドの Web サービス要求を行う注文発送モジュールのソース・コード
DFH0XODE	COBOL ソース	注文発送エンドポイントのスタブ化版のソース・コード

表 2. アプリケーションのコンポーネント (続き)

ファイル名	タイプ	コメント
DFH0XSSM	COBOL ソース	在庫マネージャー・モジュールのスタブ化版のソース・コード
DFH0XGUI	COBOL ソース	BMS インターフェース・コントローラー・アプリケーションのソース・コード
DFH0XCFG	COBOL ソース	アプリケーション構成モジュールのソース・コード
DFH0XCP1	コピーブック	カタログ・マネージャー問い合わせ操作および発注操作の COBOL コピーブック定義
DFH0XCP2	コピーブック	注文発送操作と在庫マネージャー操作の COBOL コピーブック定義
DFH0XM1	コピーブック	BMS インターフェース用の COBOL コピーブック
DFH0XM2U	コピーブック	BMS 問い合わせインターフェースのカスタマイズ済み COBOL コピーブック
DFH0XM3	コピーブック	構成モジュールに対する BMS インターフェース用の COBOL コピーブック
DFH0XS1	BMS マップ・セット	アプリケーション・ユーザー・インターフェース用の BMS マップ・セット
DFH0XS2	BMS マップ・セット	アプリケーション・ユーザー・インターフェースの問い合わせ操作用の BMS マップ・セット
DFH0XS3	BMS マップ・セット	構成モジュール用の BMS マップ・セット

表 3. CICS リソース定義

ファイル名	タイプ	コメント
EXAMPLE	CICS リソース定義グループ	実例アプリケーションに必要な CICS リソース定義
EGUI	TRANSACTION	プログラム DFH0XGUI を呼び出して、このプログラムに対する BMS インターフェースを開始するためのトランザクション (カスタマイズ可能)

表 3. CICS リソース定義 (続き)

ファイル名	タイプ	コメント
ECFG	TRANSACTION	プログラム DFH0XCFG を呼び出して実例構成 BMS インターフェースを開始するためのトランザクション (カスタマイズ可能)
EXMPCAT	FILE	アプリケーション・カタログを求めて EXMPCAT VSAM ファイルを参照するためのファイル定義 (カスタマイズ可能)
EXMPCONF	FILE	EXMPCONF アプリケーション構成ファイルを参照するためのファイル定義。パフォーマンス向上のため CICS データ・テーブルとして定義されたリソース (リソース名必須、固定)

カタログ・マネージャー・プログラム

カタログ・マネージャーは、実例アプリケーションのビジネス・ロジックの制御プログラムであり、実例アプリケーションとのすべての対話は、カタログ・マネージャーをパススルーします。

プログラム・ロジックが単純であることを確認するため、カタログ・マネージャーが制限したのは、型検査とエラー・リカバリーのみでした。

カタログ・マネージャーは、いくつかの操作をサポートしています。各操作の入出力パラメーターは、COMMAREA 内のプログラムとの間で受け渡される単一のデータ構造に定義されます。

COMMAREA 構造

```
*    カタログ COMMAREA 構造
          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).
      *    Inquire Catalog (発注) で使用されているフィールド
          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).
      *    Inquire Single (1 件の問い合わせ) で使用されているフィールド
          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).
      *    Place Order (発注) で使用されているフィールド
           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).


      *    発送プログラム/在庫マネージャーの COMMAREA 構造
           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).
      *    発送プログラムで使用されているフィールド
           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).
      *    在庫マネージャーで使用されているフィールド
           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).
```

戻りコード

カタログ・マネージャーの各操作では、いくつかの戻りコードが戻る場合があります。

タイプ	コード	説明
一般	00	機能はエラーが発生することなく実行されました。
ファイル	20	品目の参照番号が見つかりませんでした。
	21	一般的なファイル・エラーです。
	22	ファイルの更新エラーです。
構成	50	構成ファイルのオープン時エラーです。
	51	データ・ストア・タイプ - フォーマットの誤り
	52	アウトバウンドの Web サービスの切り替え - フォーマットの誤り

タイプ	コード	説明
アプリケーション	99	要求が不明です。
	98	注文数量が無効です。
	97	注文を完了するには在庫が不足しています。

INQUIRE CATALOG 操作

この操作を実行すると、呼び出し元が指定した品目を先頭に、最大 15 品目のカタログ品目リストが戻されます。

入力パラメーター

CA-REQUEST-ID
操作を指定するストリング。 INQUIRE CATALOG コマンドの場合、このストリングには「01INQC」が含まれます。

CA-LIST-START-REF
戻される最初の品目の参照番号。

出力パラメーター

CA-RETURN-CODE

CA-RESPONSE-MESSAGE
「*num* ITEMS RETURNED」を含む、人が読めるストリング。ここで、*num* は、戻される品目の数を表します。

CA-LAST-ITEM-REF
戻された最後の品目の参照番号。

CA-ITEM-COUNT
戻された品目の数。

CA-CAT-ITEM
戻されたカタログ品目のリストを含む配列。この配列のエレメントの数は 15 です。戻された品目数が 15 未満の場合、残りの配列エレメントには空白が入ります。

INQUIRE SINGLE ITEM 操作

この操作を実行すると、呼び出し元が指定した単一のカタログ品目が戻ります。

入力パラメーター

CA-REQUEST-ID
操作を指定するストリング。 INQUIRE SINGLE ITEM コマンドの場合、このストリングには「01INQS」が含まれます。

CA-ITEM-REF-REQ
戻される品目の参照番号。

出力パラメーター

CA-RETURN-CODE

CA-RESPONSE-MESSAGE
「RETURNED ITEM: REF=*item-reference*」が含まれる、人が読めるストリング。ここで、*item-reference* は、戻された品目の参照番号を表します。

CA-CAT-ITEM
戻されたカタログ項目がその最初のエレメントに含まれている配列。

PLACE ORDER 操作

この操作を実行すると、単一品目が発注されます。必要な数量が入力されていないと、ユーザーにメッセージが戻されます。 注文が正常に実行されると、在庫マネージャーが呼び出され、注文された品目とその数量が通知されます。

入力パラメーター

CA-REQUEST-ID
操作を指定するストリング。 PLACE ORDER 操作の場合、このストリングには「01ORDR」が入ります。

CA-USERID
発送および請求のためにアプリケーションが使用する 8 文字のユーザー ID。

CA-CHARGE-DEPT
発送および請求のためにアプリケーションが使用する 8 文字の部門 ID。

CA-ITEM-REF-NUMBER
注文された品目の参照番号。

CA-QUANTITY-REQ
品目の必要数。

出力パラメーター

CA-RETURN-CODE

CA-RESPONSE-MESSAGE
「ORDER SUCCESSFULLY PLACED」を含む、人が読めるストリング。

DISPATCH STOCK 操作

この操作を実行すると、在庫発送プログラムが呼び出され、このプログラムによって注文品が顧客に順に発送されます。

入力パラメーター

CA-REQUEST-ID
操作を指定するストリング。 DISPATCH ORDER 操作の場合、このストリングには「01DSPO」が入ります。

CA-USERID
発送および請求のためにアプリケーションが使用する 8 文字のユーザー ID。

CA-CHARGE-DEPT
発送および請求のためにアプリケーションが使用する 8 文字の部門 ID。

CA-ITEM-REF-NUMBER
注文された品目の参照番号。

CA-QUANTITY-REQ
品目の必要数。

出力パラメーター

CA-RETURN-CODE

NOTIFY STOCK MANAGER 操作

この操作では、在庫の補充が必要かどうかを判断するために、出された注文の詳細を取り込みます。

入力パラメーター

CA-REQUEST-ID
操作を指定するストリング。 NOTIFY STOCK MANAGER 操作の場合、このストリングには「01STKO」が入ります。

CA-ITEM-REF-NUMBER
注文された品目の参照番号。

CA-QUANTITY-REQ
品目の必要数。

出力パラメーター

CA-RETURN-CODE

BMS プレゼンテーション・マネージャー

プレゼンテーション・マネージャーは、3270 BMS パネルを介したエンド・ユーザーとの対話をすべて担っています。 このプログラムでは、ビジネス上の判断は行われません。

データ・ハンドラー

データ・ハンドラーは、カタログ・マネージャーとデータ・ストア間の抽象インターフェースを提供します。

実例アプリケーションには、以下の 2 種類のデータ・ハンドラーがあります。

- 最初のタイプでは、VSAM ファイルをデータ・ストアとして使用します。
- 2 番目のタイプは、問い合わせに対して常に同じデータを返し、更新要求の結果は保管しないダミー・プログラムです。

発送マネージャー

発送マネージャーは、注文が確認された場合に注文品を顧客へ発送する処理を担当します。

実例アプリケーションには、以下の 2 種類の発送マネージャー・プログラムがあります。

- 最初のタイプは、呼び出し元に正しい応答を戻すが、その他の動作はしないダミー・プログラムです。
- 2 番目のタイプは、構成ファイルに定義されたエンドポイント・アドレスに要求を行う Web サービス・リクエスター・プログラムです。

注文発送エンドポイント

注文発送プログラムとは、品目の顧客への発送を担当する Web サービス・プロバイダー・プログラムのことです。

この実例アプリケーションでは、注文発送プログラムは、呼び出し元に正しい応答を戻すが、その他の動作はしないダミー・プログラムです。このプログラムにより、実例 Web サービスのすべての構成が動作可能になります。

在庫マネージャー

在庫マネージャーは、在庫補充の管理を担当します。

この実例プログラムでは、在庫マネージャーは、呼び出し元に正しい応答を戻すが、その他の動作はしないダミー・プログラムです。

アプリケーションの構成

実例アプリケーションには、ベース・アプリケーションを構成できるプログラムが組み込まれています。

ファイル構造と定義

実例アプリケーションでは、2 つの VSAM ファイルが使用されます。1 つは、在庫になっているすべての品目の詳細とその在庫レベルが記録されているカタログ・ファイルで、もう 1 つは、アプリケーションのユーザー選択オプションが保持されている構成ファイルです。

カタログ・ファイル

カタログ・ファイルとは、製品の在庫に関連するすべての情報が格納されている KSDS VSAM ファイルのことです。

このファイルのレコードは、以下の構造になっています。

名前	COBOL データ・タイプ	説明
WS-ITEM-REF-NUM	PIC 9(4)	品目の参照番号
WS-DESCRIPTION	PIC X(40)	品目の説明
WS-DEPARTMENT	PIC 9(3)	部門識別番号
WS-COST	PIC ZZZ.99	品目の価格
WS-IN-STOCK	PIC 9(4)	品目の在庫数
WS-ON-ORDER	PIC 9(3)	品目の注文数

構成ファイル

構成ファイルとは、実例アプリケーションの構成に使用される情報が格納されている KSDS VSAM ファイルのことです。

構成ファイルは、3 つの異なるレコードを持つ KSDS VSAM ファイルです。

表 4. 一般情報レコード

名前	COBOL データ・タイプ	説明
PROGS-KEY	PIC X(9)	「EXMP-CONF」を含む一般情報レコードのキー・フィールド
充てん文字	PIC X	
DATASTORE	PIC X(4)	使用するデータ・ストア・プログラムのタイプを指定する文字ストリング。値は以下のとおりです。 「STUB」 「VSAM」
充てん文字	PIC X	
DO-OUTBOUND-WS	PIC X	発送マネージャーがアウトバウンドの Web サービス要求を行うかどうかを指定する文字。値は以下のとおりです。 「Y」 「N」
充てん文字	PIC X	
CATMAN-PROG	PIC X(8)	カタログ・マネージャー・プログラムの名前
充てん文字	PIC X	
DSSTUB-PROG	PIC X(8)	ダミーのデータ・ハンドラー・プログラムの名前
充てん文字	PIC X	
DSVSAM-PROG	PIC X(8)	VSAM データ・ハンドラー・プログラムの名前
充てん文字	PIC X	
ODSTUB-PROG	PIC X(8)	ダミーの注文発送モジュールの名前
充てん文字	PIC X	
ODWEBS-PROG	PIC X(8)	アウトバウンドのWeb サービス注文発送プログラムの名前
充てん文字	PIC X	
STKMAN-PROG	PIC X(8)	在庫マネージャー・プログラムの名前
充てん文字	PIC X(10)	

表 5. アウトバウンド URL レコード

名前	COBOL データ・タイプ	説明
URL-KEY	PIC X(9)	「OUTBNDURL」を含む一般情報レコードのキー・フィールド
充てん文字	PIC X	
OUTBOUND-URL	PIC X(255)	注文発送 Web サービス要求のアウトバウンド URL

表 6. カタログ・ファイル情報

名前	COBOL データ・タイプ	説明
URL-FILE-KEY	PIC X(9)	「VSAM-NAME」を含む一般情報レコードのキー・フィールド
充てん文字	PIC X	
CATALOG-FILE-NAME	PIC X(8)	カタログ・ファイルに使用された CICS FILE リソースの名前

参考文献

CICS Transaction Server for z/OS ライブラリー

CICS Transaction Server for z/OS® の公開情報は、次の形で提供されます。

CICS Transaction Server for z/OS Information Center

CICS Transaction Server のユーザー情報は、主に CICS Transaction Server for z/OS Information Center から提供されます。Information Center では、以下の情報を提供します。

- CICS Transaction Server に関する情報 (HTML 形式)
- CICS Transaction Server の使用許諾された資料および使用許諾対象外の資料 (Adobe PDF ファイル形式)。これらのファイルを使用して、資料のハードコピーを印刷することができます。詳しくは、『PDF のみの資料』 を参照してください。
- 関連製品に関する情報 (HTML 形式および PDF ファイル)

製品には、CICS Information Center のコピー 1 部 (CD-ROM) が付属しています。さらにコピーが必要な場合は、Information Center のフィーチャー番号である 7014 を指定すると、無償で注文することができます。

使用許諾された文書は、製品のライセンス所有者のみに提供されます。 使用許諾対象外の情報のみが含まれる Information Center は、資料の注文システム (資料番号 SK3T-6945) を使用して注文することができます。

同梱のハードコピー資料

製品には、以下のハードコピーの基本資料が付属しています。詳しくは、『同梱セット』 を参照してください。

同梱セット

CICS Transaction Server for z/OS、バージョン 3 リリース 1 をご注文の際、同梱セットに以下のハードコピーの資料が含まれています。

Memo to Licensees, GI10-2559
CICS Transaction Server for z/OS Program Directory, GI10-2586
CICS Transaction Server for z/OS リリース・ガイド, GD88-6377-00
CICS Transaction Server for z/OS インストール・ガイド, GD88-6381-00
CICS Transaction Server for z/OS Licensed Program Specification, GC34-6608

同梱セットの以下の資料は、上記の資料番号を使用して、コピーを追加注文することができます。

CICS Transaction Server for z/OS リリース・ガイド
CICS Transaction Server for z/OS インストール・ガイド
CICS Transaction Server for z/OS Licensed Program Specification

PDF のみの資料

以下の資料は、CICS Information Center から Adobe PDF ファイルの形で入手できます。

CICS Transaction Server for z/OS の CICS 資料

一般

CICS Transaction Server for z/OS Program Directory, GI10-2586
CICS Transaction Server for z/OS リリース・ガイド, GD88-6377-00
CICS Transaction Server for z/OS CICS TS V2.3 からのマイグレーション, GD88-6380-00
CICS Transaction Server for z/OS CICS TS V1.3 からのマイグレーション, GD88-6378-00
CICS Transaction Server for z/OS CICS TS V2.2 からのマイグレーション, GD88-6379-00
CICS Transaction Server for z/OS インストール・ガイド, GD88-6381-00

管理

CICS システム定義ガイド, SD88-6526-00
CICS Customization Guide, SC34-6429
CICS Resource Definition Guide, SC34-6430
CICS Operations and Utilities Guide, SC34-6431
CICS Supplied Transactions, SC34-6432

プログラミング

CICS Application Programming Guide, SC34-6433
CICS Application Programming Reference, SC34-6434
CICS System Programming Reference, SC34-6435
FEPI ユーザーズ・ガイド, SD88-6382-00
CICS C++ OO Class Libraries, SC34-6437
CICS Distributed Transaction Programming Guide, SC34-6438
CICS Business Transaction Services, SC34-6439
Java Applications in CICS, SC34-6440
JCICS Class Reference, SC34-6001

診断

CICS Problem Determination Guide, SC34-6441
CICS Messages and Codes, GC34-6442
CICS Diagnosis Reference, LY33-6110
CICS Data Areas, LY33-6107
CICS Trace Entries, SC34-6443
CICS Supplementary Data Areas, LY33-6108

通信

CICS Intercommunication Guide, SC34-6448
CICS External Interfaces Guide, SC34-6449
CICS Internet Guide, SC34-6450

特殊なトピック

CICS Recovery and Restart Guide, SC34-6451
CICS パフォーマンス・ガイド, SD88-6391-00
CICS IMS Database Control Guide, SC34-6453
CICS RACF Security Guide, SC34-6454
CICS Shared Data Tables Guide, SC34-6455
CICS DB2 Guide, SC34-6457
CICS Debugging Tools Interfaces Reference, LY33-6109

CICS Transaction Server for z/OS 用の CICSPlex SM の資料

一般

CICSPlex SM Concepts and Planning, SC34-6459
CICSPlex SM User Interface Guide, SC34-6460
CICSPlex SM Web User Interface Guide, SC34-6461

管理

CICSPlex SM Administration, SC34-6462
CICSPlex SM Operations Views Reference, SC34-6463
CICSPlex SM Monitor Views Reference, SC34-6464
CICSPlex SM Managing Workloads, SC34-6465
CICSPlex SM Managing Resource Usage, SC34-6466
CICSPlex SM Managing Business Applications, SC34-6467

プログラミング

CICSPlex SM Application Programming Guide, SC34-6468
CICSPlex SM Application Programming Reference, SC34-6469

診断

CICSPlex SM Resource Tables Reference, SC34-6470
CICSPlex SM Messages and Codes, GC34-6471
CICSPlex SM Problem Determination, GC34-6472

CICS ファミリーの資料

通信

CICS プロダクト間通信ガイド, SD88-6384-00
S/390 CICS からの通信, SD88-6385-00

ライセンス出版物

Information Center の使用許諾対象外の資料には、以下のライセンス出版物は含まれていません。

CICS Diagnosis Reference, LY33-6110
CICS Data Areas, LY33-6107
CICS Supplementary Data Areas, LY33-6108
CICS Debugging Tools Interfaces Reference, LY33-6109

CICS のその他の資料

以下の資料では、CICS に関する詳細情報を記載しています。これらの資料は、CICS Transaction Server for z/OS、バージョン 3 リリース 1 には含まれていません。

Designing and Programming CICS Applications	SR23-9692
CICS Application Migration Aid Guide	SC33-0768
CICS ファミリー: API の構成	SC88-7261-02
CICS ファミリー: クライアント・サーバー プログラミング	SC88-7429-04
CICS Transaction Gateway (OS/390 版) 管理の手引き バージョン 3.1	SD88-7246-01
CICS Family: General Information	GC33-0155-05
CICS 4.1 Sample Applications Guide	SC33-1173
CICS/ESA 3.3 XRF Guide	SC33-0661-02

最新の資料の確認

IBM® では、各資料を新規情報および変更情報を反映して定期的に更新しています。通常資料は、初回発行時には、ハードコピー版と BookManager® のソフトコピー版が一致しています。ハードコピーの資料は印刷と配布に時間が必要であるため、BookManager 版に発行前に行われた直前の変更内容が含まれることが多くなります。

それ以降の更新情報は、たいていの場合、ハードコピーとして提供される前にソフトコピーで入手可能になります。つまり、リリース後は常に、ソフトコピー版を最新の情報と考える必要があります。

CICS Transaction Server の資料の場合、ソフトコピーの更新情報が *Transaction Processing and Data Collection Kit* CD-ROM (SK2T-0730-xx) として定期的に提供されています。このコレクション・キットの再発行は、資料番号の接尾部 (-xx 部分) の更新によって示されます。例えば、コレクション・キット SK2T-0730-06 は、SK2T-0730-05 より新しい情報となります。コレクション・キットのカバーには、日付も明記されています。

ソフトコピーへの更新は、変更内容の左側にある改訂コード (通常は # 文字) で明示しています。

アクセシビリティ

アクセシビリティ機能は、運動障害または視覚障害など身体に障害を持つユーザーがソフトウェア・プロダクトを快適に使用できるようにサポートします。

CICS システムのセットアップ、実行、および保守に関するほとんどの作業は、以下のいずれかの方法で実行できます。

- CICS にログオンした 3270 エミュレーターを使用する
- TSO にログオンした 3270 エミュレーターを使用する
- MVS™ システム・コンソールとして 3270 エミュレーターを使用する

IBM パーソナル・コミュニケーションズは、身体に障害のあるユーザーのためのアクセシビリティ機能を備えた 3270 エミュレーションを提供します。この製品を使用すると、ご使用の CICS システムで必要なアクセシビリティ機能が利用可能になります。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[ハ行]

[ヤ行]

D

S

W

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。 本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

本書には、技術的に正確でない記述や誤植がある場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。IBM United Kingdom Laboratories, MP151, Hursley Park, Winchester, Hampshire, England, SO21 2JN 本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

商標

以下は、IBM Corporation の商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

プログラム番号: 5655-M15

SD88-6383-00

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12